Microsoft

OFFICIAL MICROSOFT LEARNING PRODUCT

10987C

Performance Tuning and Optimizing SQL Databases

Product Number: 10987C

Part Number (if applicable): X21-64450

Released: 11/2017

MICROSOFT LICENSE TERMS
MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS. IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

a. "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.

b. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.

c. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.

d. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of a MPN Member, or (iii) a Microsoft full-time employee.

e. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.

f. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.

g. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.

h. "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.

i. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.

j. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.

k. "MPN Member" means an active Microsoft Partner Network program member in good standing.

l. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.

m. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.

n. "Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.

o. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.

2. USE RIGHTS. The Licensed Content is licensed not sold. The Licensed Content is licensed on a ***one copy per user basis***, such that you must acquire a license for each individual that accesses or uses the Licensed Content.

2.1 Below are five separate sets of use rights. Only one set of rights apply to you.

a. **If you are a Microsoft IT Academy Program Member:**
   i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
   ii. For each license you acquire on behalf of an End User or Trainer, you may either:
      1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
      2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
      3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,

   provided you comply with the following:
   iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
   iv. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
   v. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
   vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

vii. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
viii. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
ix. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

b. **If you are a Microsoft Learning Competency Member**:
i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
ii. For each license you acquire on behalf of an End User or Trainer, you may either:
    1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
    2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
    3. you will provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,

    provided you comply with the following:
iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
iv. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
v. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for your Authorized Training Sessions,
viii. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
x. you will only provide access to the Trainer Content to Trainers.

c. **If you are a MPN Member**:

i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

ii. For each license you acquire on behalf of an End User or Trainer, you may either:

1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content,

provided you comply with the following:

iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,

iv. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,

v. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

vi. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,

vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,

viii. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,

ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and

x. you will only provide access to the Trainer Content to Trainers.

d. **If you are an End User:**

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

e. **If you are a Trainer.**

i. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

ii. You may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "*customize*" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

2.2 **Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.

2.3 **Redistribution of Licensed Content**. Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.

2.4 **Third Party Notices**. The Licensed Content may include third party code tent that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code ntent are included for your information only.

2.5 **Additional Terms**. Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY. If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("**Pre-release**"), then in addition to the other provisions in this agreement, these terms also apply:

a. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.

b. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.

c. **Pre-release Term**. If you are an Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("**Pre-release term**"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.

4. **SCOPE OF LICENSE**. The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
   - access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
   - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
   - modify or create a derivative work of any Licensed Content,
   - publicly display, or make the Licensed Content available for others to access or use,
   - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
   - work around any technical limitations in the Licensed Content, or
   - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.

5. **RESERVATION OF RIGHTS AND OWNERSHIP**. Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.

6. **EXPORT RESTRICTIONS**. The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.

7. **SUPPORT SERVICES**. Because the Licensed Content is "as is", we may not provide support services for it.

8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.

9. **LINKS TO THIRD PARTY SITES**. You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.

10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.

11. **APPLICABLE LAW.**
    a. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.

b. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.

12. LEGAL EFFECT. This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.

13. DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

14. LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection dues consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 $ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.
Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised July 2013

Welcome!

Thank you for taking our training! We've worked together with our Microsoft Certified Partners for Learning Solutions and our Microsoft IT Academies to bring you a world-class learning experience—whether you're a professional looking to advance your skills or a student preparing for a career in IT.

- **Microsoft Certified Trainers and Instructors**—Your instructor is a technical and instructional expert who meets ongoing certification requirements. And, if instructors are delivering training at one of our Certified Partners for Learning Solutions, they are also evaluated throughout the year by students and by Microsoft.

- **Certification Exam Benefits**—After training, consider taking a Microsoft Certification exam. Microsoft Certifications validate your skills on Microsoft technologies and can help differentiate you when finding a job or boosting your career. In fact, independent research by IDC concluded that 75% of managers believe certifications are important to team performance[1]. Ask your instructor about Microsoft Certification exam promotions and discounts that may be available to you.

- **Customer Satisfaction Guarantee**—Our Certified Partners for Learning Solutions offer a satisfaction guarantee and we hold them accountable for it. At the end of class, please complete an evaluation of today's experience. We value your feedback!

We wish you a great learning experience and ongoing success in your career!

Sincerely,

Microsoft Learning
www.microsoft.com/learning

Microsoft | Learning

[1] *IDC,* Value of Certification: Team Certification and Organizational Performance, *November 2006*

Acknowledgements

Microsoft Learning would like to acknowledge and thank the following for their contribution towards developing this title. Their effort at various stages in the development has ensured that you have a good classroom experience.

Aaron Johal – Content Developer

Aaron Johal is a Microsoft Certified Trainer who splits his time between training, consultancy, content development, contracting and learning. Since he moved into the non-functional side of the Information Technology business. He has presented technical sessions at SQL Pass in Denver and at sqlbits in London. He has also taught and worked in a consulting capacity throughout the UK and abroad, including Africa, Spain, Saudi Arabia, Netherlands, France, and Ireland. He enjoys interfacing functional and non-functional roles to try and close the gaps between effective use of Information Technology and the needs of the Business.

Caroline Eveleigh – Content Developer

Caroline Eveleigh is a Microsoft Certified Professional and SQL Server specialist. She has worked with SQL Server since version 6.5 and, before that, with Microsoft Access and dBase. Caroline works on database development and Microsoft Azure projects for both corporates, and small businesses. She is an experienced business analyst, helping customers to re-engineer business processes, and improve decision making using data analysis. Caroline is a trained technical author and a frequent blogger on project management, business intelligence, and business efficiency. Between development projects, Caroline is a keen SQL Server evangelist, speaking and training on SQL Server and Azure SQL Database.

Rachel Horder – Content Developer

Rachel Horder graduated with a degree in Journalism and began her career in London writing for The Times technology supplement. After discovering a love for programming, Rachel became a full-time developer, and now provides SQL Server consultancy services to businesses across a wide variety of industries. Rachel is MCSA certified, and continues to write technical articles and books, including What's New in SQL Server 2012. As an active member of the SQL Server community, Rachel organizes the Bristol SQL Server Club user group, runs the Bristol leg of SQL Relay, and is a volunteer at SQLBits.

Simon Butler – Content Developer

Simon Butler FISTC is a highly-experienced Senior Technical Writer with nearly 30 years' experience in the profession. He has written training materials and other information products for several high-profile clients. He is a Fellow of the Institute of Scientific and Technical Communicators (ISTC), the UK professional body for Technical Writers/Authors. To gain this, his skills, experience and knowledge have been judged and assessed by the Membership Panel. He is also a Past President of the Institute and has been a tutor on the ISTC Open Learning course in Technical Communication techniques. His writing skills are augmented by extensive technical skills gained within the computing and electronics fields.

Geoff Allix – Technical Reviewer

Geoff Allix is a Microsoft SQL Server subject matter expert and professional content developer at Content Master—a division of CM Group Ltd. As a Microsoft Certified Trainer, Geoff has delivered training courses on SQL Server since version 6.5. Geoff is a Microsoft Certified IT Professional for SQL Server and has extensive experience in designing and implementing database and BI solutions on SQL Server technologies, and has provided consultancy services to organizations seeking to implement and optimize database solutions.

Lin Joyner – Technical Reviewer

Lin is an experienced Microsoft SQL Server developer and administrator. She has worked with SQL Server since version 6.0 and previously as a Microsoft Certified Trainer, delivered training courses across the UK. Lin has a wide breadth of knowledge across SQL Server technologies, including BI and Reporting Services. Lin also designs and authors SQL Server and .NET development training materials. She has been writing instructional content for Microsoft for over 15 years.

Contents

Lab Answer Keys

About This Course

This section provides a brief description of the course, audience, suggested prerequisites, and course objectives.

Course Description

This four-day instructor-led course provides students who manage and maintain SQL Server databases with the knowledge and skills to performance tune and optimize their databases.

Audience

The primary audience for this course is individuals who administer and maintain SQL Server databases and are responsible for optimal performance of SQL Server instances that they manage. These individuals also write queries against data and need to ensure optimal execution performance of the workloads.

The secondary audiences for this course are individuals who develop applications that deliver content from SQL Server databases.

Student Prerequisites

In addition to their professional experience, students who attend this training should already have the following technical knowledge:

- Basic knowledge of the Microsoft Windows operating system and its core functionality.
- Working knowledge of database administration and maintenance.
- Working knowledge of Transact-SQL.

Course Objectives

After completing this course, students will be able to:

- Describe the high level architectural overview of SQL Server and its various components.
- Describe the SQL Server execution model, waits and queues.
- Describe core I/O concepts, Storage Area Networks and performance testing.
- Describe architectural concepts and best practices related to data files for user databases and TempDB.
- Describe architectural concepts and best practices related to Concurrency, Transactions, Isolation Levels and Locking.
- Describe architectural concepts of the Optimizer and how to identify and fix query plan issues.
- Describe architectural concepts, troubleshooting scenarios and best practices related to Plan Cache.
- Describe architectural concepts, troubleshooting strategy and usage scenarios for Extended Events.
- Explain data collection strategy and techniques to analyze collected data.
- Understand techniques to identify and diagnose bottlenecks to improve overall performance.

Course Outline

The course outline is as follows:

- Module 1: "SQL Server architecture, scheduling, and waits" covers high level architectural overview of SQL Server and its various components. It dives deep into SQL Server execution model, waits and queues.
- Module 2: "SQL server IO" covers core I/O concepts, Storage Area Networks and performance testing. It focuses on SQL Server I/O operations and how to test storage performance.
- Module 3: "Database structures" covers Database Structures, Data File and TempDB Internals. It focuses on architectural concepts and best practices related to data files for user databases and TempDB.
- Module 4: "SQL Server memory" covers Windows and SQL Server memory internals. It focuses on architectural concepts and best practices related to SQL Server memory configuration.
- Module 5: "SQL Server concurrency" covers Transactions and Locking Internals. It focuses on architectural concepts and best practices related to Concurrency, Transactions, Isolation Levels and Locking.
- Module 6: "Statistics and index internals" covers Statistics and Index Internals. It focuses on architectural concepts and best practices related to Statistics and Indexes.
- Module 7: "Query execution and query plan analysis" covers Query Execution and Query Plan Analysis. It focuses on architectural concepts of the Optimizer and how to identify and fix query plan issues.
- Module 8: "Plan caching and recompilation" covers Plan Caching and Recompilation. It focuses on architectural concepts, troubleshooting scenarios and best practices related to Plan Cache.
- Module 9: "Extended events" covers Extended Events. It focuses on architectural concepts, troubleshooting strategy and usage scenarios for Extended Events.
- Module 10: "Monitoring, Tracing, and baselining" covers tools and techniques to monitor, trace and baseline SQL Server performance data. It focuses on data collection strategy and techniques to analyze collected data.

Course Materials

The following materials are included with your kit:

- ***Course Handbook:*** a succinct classroom learning guide that provides the critical technical information in a crisp, tightly-focused format, which is essential for an effective in-class learning experience.
  - **Lessons**: guide you through the learning objectives and provide the key points that are critical to the success of the in-class learning experience.
  - **Labs**: provide a real-world, hands-on platform for you to apply the knowledge and skills learned in the module.
  - **Module Reviews and Takeaways**: provide on-the-job reference material to boost knowledge and skills retention.
  - **Lab Answer Keys**: provide step-by-step lab solution guidance.

Additional Reading: Course Companion Content on the http://www.microsoft.com/learning/en/us/companion-moc.aspx Site*:* searchable, easy-to-browse digital content with integrated premium online resources that supplement the Course Handbook.

- **Modules**: include companion content, such as questions and answers, detailed demo steps and additional reading links, for each lesson. Additionally, they include Lab Review questions and answers and Module Reviews and Takeaways sections, which contain the review questions and answers, best practices, common issues and troubleshooting tips with answers, and real-world issues and scenarios with answers.
- **Resources**: include well-categorized additional resources that give you immediate access to the most current premium content on TechNet, MSDN®, or Microsoft® Press®.

Additional Reading: Student Course files on the http://www.microsoft.com/learning/en/us/companion-moc.aspx Site: includes the Allfiles.exe, a self-extracting executable file that contains all required files for the labs and demonstrations.

- ***Course evaluation:*** at the end of the course, you will have the opportunity to complete an online evaluation to provide feedback on the course, training facility, and instructor.
- To provide additional comments or feedback on the course, send email to mcspprt@microsoft.com. To inquire about the Microsoft Certification Program, send an email to mcphelp@microsoft.com.

Virtual Machine Environment

This section provides the information for setting up the classroom environment to support the business scenario of the course.

Virtual Machine Configuration

In this course, you will use Microsoft® Hyper-V™ to perform the labs.

Note: At the end of each lab, you must revert the virtual machines to a snapshot. You can find the instructions for this procedure at the end of each lab

The following table shows the role of each virtual machine that is used in this course:

Virtual machine	Role
10987C-MIA-DC	MIA-DC1 is domain controller and has Dynamic Host Configuration Protocol (DHCP), Domain Name System (DNS), and Active Directory Certificate Services (AD CS) roles installed.
10987C-MIA-SQL	10987C-MIA-SQL is a SQL Server and SharePoint Server.

Software Configuration

The following software is installed on the virtual machines:

- Microsoft Windows Server 2016
- Microsoft SQL Server 2017
- Microsoft SharePoint Server 2016
- Microsoft Visual Studio 2017

Course Files

The files associated with the labs in this course are located in the D:\Labfiles folder on the 10987C-MIA-SQL virtual machine.

Classroom Setup

Each classroom computer will have the same virtual machine configured in the same way.

Course Hardware Level

To ensure a satisfactory student experience, Microsoft Learning requires a minimum equipment configuration for trainer and student computers in all Microsoft Learning Partner classrooms in which Official Microsoft Learning Product courseware is taught.

- Processor: Intel Virtualization Technology (Intel VT) or AMD Virtualization (AMD-V)
- Hard Disk: Dual 120 GB hard disks 7200 RM SATA or better (Striped)
- RAM: 12GB or higher. 16 GB or more is recommended for this course.
- DVD/CD: DVD drive

- Network adapter with Internet connectivity
- Video Adapter/Monitor: 17-inch Super VGA (SVGA)
- Microsoft Mouse or compatible pointing device
- Sound card with amplified speakers

Additionally, the instructor's computer must be connected to a projection display device that supports SVGA 1024×768 pixels, 16-bit colors.

Module 1

SQL Server Architecture, Scheduling, and Waits

Contents:

Module Overview

This module gives a high level architectural overview of the Microsoft® SQL Server® Database Engine and its various components. It dives deep into the SQL Server execution model, in addition to waits, and queues.

Objectives

After completing this module, you will be able to:

- Describe the SQL Server architecture.
- Describe Microsoft Azure™ SQL Database.
- Describe and monitor SQL Server scheduling.
- Analyze wait statistics.

Lesson 1
SQL Server Components and SQLOS

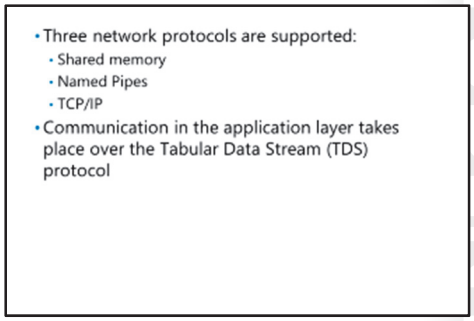
The SQL Server Database Engine is a complex software product. Knowledge of database engine architecture can be critical to understanding and addressing performance issues in a logical fashion. This lesson covers several important database engine components and how these components interact when a Transact-SQL query is processed. The lesson will also cover the Dynamic Management Objects (DMOs) that you can use to monitor the components of the database engine.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the different components of the database engine and the SQL Server Operating System (SQLOS).
- Explain worker scheduling in SQL Server.
- Monitor database engine behavior by using waits and queues.

Connection Protocols



Client applications will connect to an instance of the SQL Server Database Engine through endpoints that are exposed by the server process. Endpoints can be configured by using one of three transport-layer/session-layer protocols:

- Shared memory
- Named Pipes
- TCP/IP

Note: From SQL Server 2005, communication over the Virtual Interface Adapter (VIA) protocol was supported.
The VIA protocol is now deprecated, and will be removed in future versions of SQL Server. It is however, still supported in SQL Server 2016, although not for failover clusters.
Microsoft recommend that future development does not use the VIA protocol, and that existing applications are modified so they will continue to work after it is removed from SQL Server.

After a session is established, the client and the server communicate by using the Tabular Data Stream (TDS) application-layer protocol.

For more information about SQL Server network protocols, see the topic *Choosing a Network Protocol* in SQL Server Technical Documentation.

Choosing a Network Protocol

http://aka.ms/u9g8kj

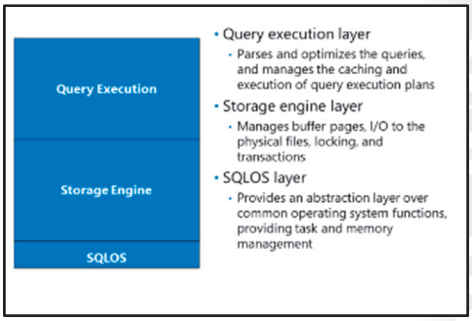
For full details of the TDS protocol, including a full specification, see the topic *[MS-TDS]: Tabular Data Stream Protocol* on MSDN.

[MS-TDS]: Tabular Data Stream Protocol

http://aka.ms/orhw9u

Database Engine

The SQL Server Database Engine is responsible for processing, storing, and securing data in SQL Server; Transact-SQL statements are serviced by the database engine. The database engine consists of a set of subcomponents that work together. Three general categories of components exist within the database engine and are structured as layers:



- The query execution layer (also referred to as the relational engine).
- The storage engine layer.
- The SQLOS layer.

Query Execution Layer

In addition to managing the query optimization process, the query execution layer manages connections and security. A series of subcomponents helps it to work out how to execute your queries:

- The parser checks that you have followed the rules of the Transact-SQL language. It then generates a syntax tree, which is a logical representation of the queries to be executed.
- The algebrizer converts the syntax tree into a relational algebra tree, where operations are represented by logic objects rather than words. The aim of this phase is to take the list of what you want to achieve, and convert it to a logical series of operations that represent the work that needs to be performed.
- The query optimizer then considers the different ways in which your queries could be executed and finds an acceptable plan. The query optimizer also manages a query plan cache to avoid the overhead of performing all of this work when another similar query is received for execution.

Storage Engine Layer

The storage engine layer manages the data that is held within databases. The main responsibilities of the storage engine layer are to manage:

- How data is stored, both on disk and in memory.
- How data is cached for reuse.
- The consistency of data through locking and transactions.

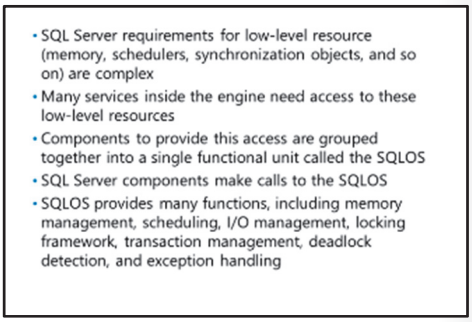
The main subcomponents of the storage engine layer are as follows:

- The access methods component is used to manage how data is accessed. For example, the access methods component works with scans, seeks, and lookups.
- The page cache manages the storage of cached copies of data pages. Caching of data pages is used to minimize the time that it takes to access them. The page cache places data pages into memory so that they are present when they are needed for query execution.
- The locking and transaction management components work together to maintain the consistency of your data. This includes the maintenance of transactional integrity, with the help of the database log file.

SQL Server Operating System Layer

SQLOS is the layer of SQL Server that provides operating system functionality to the SQL Server components. All SQL Server components use programming interfaces that SQLOS provides to access memory, schedule tasks, or perform I/O. SQLOS is discussed further in the next topic.

SQLOS



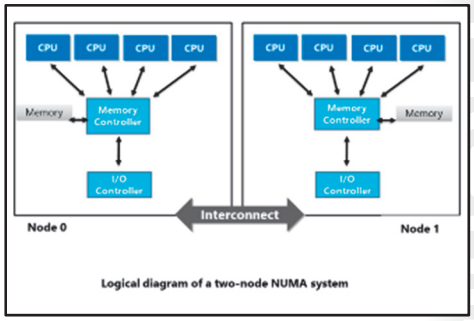
The SQL Server Operating System (SQLOS), which was first introduced in SQL Server 2005, is an operating system inside SQL Server. Prior to SQL Server 2005, there was a thin layer of interfaces between SQL Server and Windows® to manage tasks such as memory allocation and scheduling. As the requirement for these low-level functions became more complex, a single application layer was designed to cater to these requirements instead of making changes to different engines separately. The abstraction layer that is provided by SQLOS avoids the need for resource-related code to be present throughout the SQL Server Database Engine code.

SQLOS is not a Windows service; rather, it is a user-mode operating system. It consists of operating system components such as non-preemptive scheduling, memory management, resource monitoring, exception handling, synchronization, deadlock detection, extended events, and asynchronous I/O. SQLOS exploits the scalability and performance features of Windows to enhance the performance of SQL Server. Many of the tasks that SQLOS performs would typically be managed by the host operating system for most applications.

SQLOS provides highly detailed reporting and metrics to give administrators insight into how a SQL Server instance is performing through Dynamic Management Views (DMVs). You will learn how to use DMVs to assist in performance tuning in several modules of this course.

Multiple CPUs—SMP and NUMA

Symmetric Multiprocessing System



Logical diagram of a two-node NUMA system

In a typical symmetric multiprocessing (SMP) system, two or more identical processors connect to a shared main memory with full access to all I/O devices, controlled by a single operating system instance that treats all processors equally. Having more than one CPU gives SMP better performance characteristics than one-processor systems. However, as the number of CPUs increases, the limitation of a single processor bus results in a scalability bottleneck. Generally, SMP systems degrade in performance when they have eight or more CPUs.

SQL Server can be configured to use some or all of the processors that are available in an SMP system through the processor affinity mask configuration setting. By default, all available processors are used, up to the limit that is imposed by the edition of SQL Server that you are using.

Non-Uniform Memory Access

The scalability issue in SMP system design can be resolved by interconnecting two or more SMP systems with a high-speed interconnect. This enables the individual groups or nodes to function as a single system. This hardware design is called non-uniform memory access (NUMA). NUMA enables the CPUs to access local memory and memory from the other node. The access to memory from another node is slower than that of local memory, which is why it is called non-uniform memory access.

SQLOS is NUMA-aware. Each of the NUMA nodes is mapped to an internal scheduler node. The memory node is separate from the scheduler node. The **sys.dm_os_nodes** and **sys.dm_os_schedulers** Dynamic Management Views provide information about NUMA configuration. SQLOS tries to reduce the need for remote memory access. Whenever possible, the memory objects for a task are created within the same NUMA node in which a task is running—this reduces the need to access memory from another node.

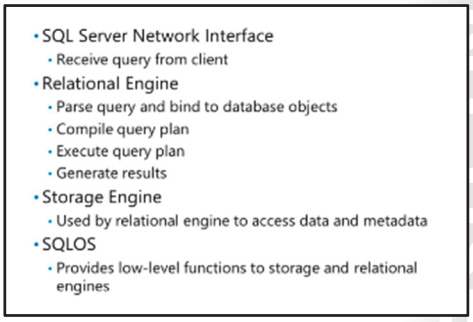
The processor affinity mask setting can be used to control which processors SQL Server will use at NUMA node level, and at the level of individual processors.

It is possible to configure the SQL Server Network Interface layer to associate an IP address with a specific NUMA node.

Advantages of NUMA

The main advantage of NUMA is scalability. As previously mentioned, traditional SMP systems are difficult to scale past eight CPUs. In SMP systems, all memory access is sent to the same shared memory bus. The shared bus works fine with a limited number of CPUs; however, it becomes a bottleneck with 10s or 100s of CPUs competing for a single shared bus. NUMA removes this bottleneck by limiting the number of CPUs on one memory bus and connecting the different nodes via high-speed interconnect.

The Query Life Cycle



During the execution of a SELECT query, the components of the database engine interact as follows.

1. Parsing and Binding

The query is received from the client as a TDS packet by the SQL Server Network Interface (SNI). The SNI identifies and extracts the SELECT statement from the TDS packet and passes it to the relational engine.

A subcomponent of the relational engine, the command parser, validates the statement's syntax and parses it into a logical query tree.

The references in the logical query tree are then bound to actual database objects; if any of the objects that are referenced in the query do not exist, an "invalid object name" error will be raised.

2. Plan Compilation

The relational engine generates a hash of the logical query tree and compares the hash against plan hashes that are already stored in the query plan cache. If a match is found, the cached query plan is reused.

If no matching query plan is found, another subcomponent of the relational engine, the query optimizer, is employed to generate a suitable query plan. The query optimizer uses various techniques to attempt to generate an effective query plan in a reasonable amount of time; the optimizer is not guaranteed to find the best plan, but attempts to find a plan that is sufficiently good. The new plan will normally be stored in the query plan cache for potential reuse later.

3. Query Execution

The relational engine carries out the query plan with the query executor subcomponent. The query executor will use the storage engine to retrieve data pages either from the data cache (if the page is already held in memory) or from storage by using disk I/O. Data pages that are retrieved from storage may be added to the data cache by the storage engine's buffer manager.

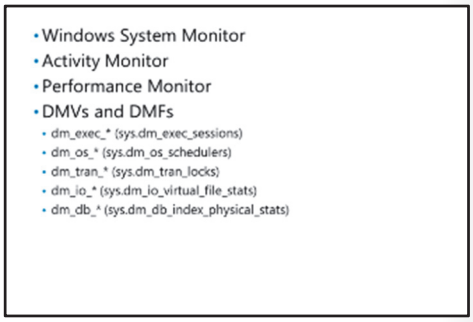
4. Result Generation

When the query executor returns the result set, the relational engine formats the result set as a table and encapsulates it into a TDS packet. The TDS packet is then dispatched to the client.

Monitoring Engine Behavior

SQL Server provides several methods of monitoring database engine behavior, including:

- Activity Monitor.
- Performance Monitor.
- Dynamic Management Objects.



Note: All of these monitoring methods offer a view of the performance counters and metrics that SQLOS exposes.

Activity Monitor

Activity Monitor is a tabbed document window that can be displayed in SQL Server Management Studio to display SQL Server resource utilization details. The information is displayed in the following expandable and collapsible panes:

- **Overview**. Gives a graphical view of processor time, waiting task count, database I/O (MB/sec) and batch requests/sec.
- **Processes**. Provides details about SQL Server sessions.
- **Resource Waits**. Displays wait statistics.
- **Data File I/O**. Gives I/O usage (read and write) for data files across databases.
- **Recent Expensive Queries**. Lists details of expensive queries for which details exist in the plan cache.

For more information about Activity Monitor, see the topic *Open Activity Monitor (SQL Server Management Studio)* in Microsoft Docs:

Open Activity Monitor (SQL Server Management Studio)

http://aka.ms/k8h7kb

Performance Monitor

Performance Monitor is a Windows management snap-in you use to analyze system performance. It can be used for real-time monitoring of current system performance and to log data to be analyzed later. It provides different counters that are categorized into objects to monitor Windows and other applications that are running on the Windows operating system.

SQL Server makes a large number of performance counters available to Performance Monitor to make it easier for you to monitor SQL Server activity and Windows activity from one place. Some examples of SQL Server performance counters that are typically used when examining SQL Server performance are:

- SQL Server: Buffer Manager: Buffer Cache Hit Ratio
- SQL Server: Buffer Manager: Page Life Expectancy
- SQL Server: SQL Statistics: Batch Requests/Sec
- SQL Server: SQL Statistics: SQL Compilations/Sec
- SQL Server: SQL Statistics: SQL Recompilations/sec

For more information about Performance Monitor, see the topic *Monitor Resource Usage (System Monitor)* in Microsoft Docs.

Monitor Resource Usage (System Monitor)

http://aka.ms/gobdkk

Dynamic Management Objects

Dynamic Management Views (DMVs) and Dynamic Management Functions (DMFs) can provide insight into the internal state, health, and performance of SQL Server through Transact-SQL queries. They provide useful information that can be used to identify, diagnose, and fix SQL Server performance issues from counters and metrics that SQLOS collects. Each Dynamic Management Object (DMO) will return data either at server scope or database scope.

There are more than 200 DMOs in SQL Server. DMOs can be found in the **sys** schema; their names start with "dm". They are further divided into other categories based on the kind of information that they return. Some of the most useful DMO categories that you can use when performance tuning SQL Server are listed here:

- **sys.dm_exec_***. This set of DMVs contains information that relates to the execution of user queries. They are useful in diagnosing blocking, deadlock, long-running queries, CPU-intensive queries, and so on. For example, **sys.dm_exec_requests** returns information about currently executing requests.
- **sys.dm_os_***. This set of DMVs gives details about SQLOS operations; that is, memory management and scheduling. For example, **sys.dm_os_wait_stats** returns aggregate information about all of the waits that have been incurred on an instance of SQL Server. The **sys.dm_os_schedulers** DMV returns one row per scheduler and is used to monitor the health and condition of the scheduler, or to identify runaway tasks.
- **sys.dm_tran_***. This set of DMVs provides details about current transactions and isolation. For example, the **sys.dm_tran_active_transactions** DMV returns transaction details for a SQL Server instance.
- **sys.dm_io_***. This set of DMVs provides insight into the I/O activity in SQL Server. For example, **sys.dm_io_pending_io_requests** identifies pending I/O requests that SQL Server has initiated.
- **sys.dm_db_***. This set of DMVs provides details about the database and database-level objects such as tables and indexes. For example, **sys.dm_db_index_physical_stats** is a DMF that returns fragmentation information across all indexes in a database or for a specific index.

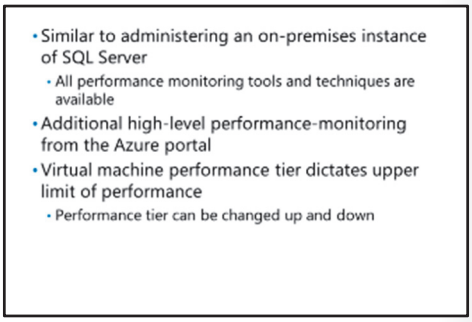
For a complete list of DMOs in SQL Server, see the topic *Dynamic Management Views and Functions (Transact-SQL)* in Microsoft Docs.

Dynamic Management Views and Functions (Transact-SQL)

http://aka.ms/yv48h3

Performance and Scalability in a SQL Server Instance on an Azure Virtual Machine

When you run SQL Server on an Azure virtual machine, Microsoft provides compute and storage resources; you configure and administer Windows and SQL Server. Apart from the infrastructure on which it is hosted, using SQL Server on Azure virtual machines is almost identical to running SQL Server on hardware that you own; you must manage all aspects of server and database administration, including performance tuning.



Performance Tools

Whether you run SQL Server on an Azure virtual machine or on an on-premises SQL Server instance, all of the same performance-monitoring tools that are covered in this course are available to you.

Some high-level performance reports are available from the Azure portal.

Performance Tiers

In the same way that you define the hardware specification for a physical server or define an on-premises virtual machine, you can define hardware characteristics for an Azure virtual machine. The hardware characteristics that you select will impose a limit on the performance of the virtual machine and any instance of SQL Server that it hosts.

When you configure an Azure virtual machine, you choose from several performance tiers that have defined CPU, memory, and disk performance characteristics. The higher the performance tier, the higher the performance (and the greater the per-minute billing cost).

You can change the performance tier of an Azure virtual machine while it is running to enable you to scale performance up or down in response to demand. Each virtual machine may have its performance tier adjusted up to four times in a 24-hour period.

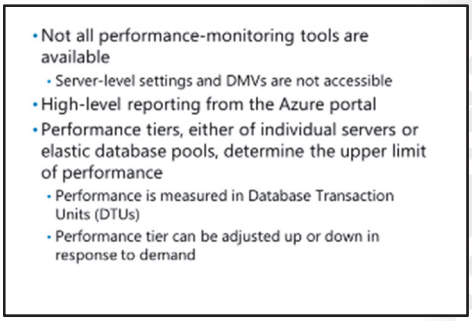
For current information about performance characteristics and pricing for Azure virtual machines, and licensing costs for platform-provided SQL Server images, see *Virtual Machines Pricing* in the Azure documentation.

Virtual Machines Pricing

http://aka.ms/x7cmej

Performance and Scalability in Azure SQL Database

Azure SQL Database aims to minimize administration costs for using SQL Server. Microsoft manages the operating system, SQL Server instance, and most aspects of database administration (including upgrades, backups, and high availability); you are only responsible for the data that is held in the database.



Performance Tools

Most server-level DMOs are not accessible when you use Azure SQL Database because you do not manage the SQL Server instance. However, most database-level DMOs are accessible.

Some high-level performance reports are available from the Azure portal.

Performance Tiers

As with Azure virtual machines, the performance of Azure SQL Database is determined by the performance tier that is selected for the database. Unlike virtual machines, which have specified CPU, memory, and disk performance characteristics, performance tiers for Azure SQL Database are measured in a special unit that is called the Database Transaction Unit (DTU). This is a measure that combines CPU, memory, and disk performance to give a measure of transactions per second that the Azure SQL Database instance can support:

- 1 DTU = 1 transaction per second

The higher the DTU value for an Azure SQL Database service tier, the more transactions per second it will support.

The performance tier of an instance of Azure SQL Database can be adjusted up and down in response to demand.

Instances of Azure SQL Database may also be assigned to an elastic database pool, which enables several databases to share a fixed set of resources. The performance of elastic database pools is measured in elastic Database Transaction Units (eDTUs). As with a DTU, one eDTU is equivalent to one transaction per second. A distinction is made between DTUs and eDTUs because eDTUs are only allocated as needed.

Performance tiers in Azure SQL Database and elastic database pools also impose limits on maximum database size.

For more information about Azure SQL Database performance tiers, see the topic *SQL Database options and performance: Understand what's available in each service tier*, in the Azure documentation.

SQL Database options and performance: Understand what's available in each service tier

http://aka.ms/dr66jn

Demonstration: Monitoring Engine Behavior

In this demonstration, you will see how to monitor aspects of database engine behavior.

Demonstration Steps

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the D:\Demofiles\Mod01 folder as Administrator. In the **User Account Control** dialog box, click **Yes**, and wait for the script to complete.
3. Start SQL Server Management Studio, and then connect to the **MIA-SQL** database engine instance by using Windows authentication.
4. Open the **Project.ssmsslnproj** solution in the D:\Demofiles\Mod01\Project folder.
5. In Solution Explorer, expand, **Queries**, and then open the **Demo1 - Monitor_Engine.sql** script file.
6. In the D:\Demofiles\Mod01 folder, right-click **start_load_1.ps1**, and then click **Run with PowerShell**. If the **Execution Policy change** message appears, type **y**, and then press Enter.

   Leave the script running, and continue with the demo.
7. In SQL Server Management Studio, highlight the script below **Step 2**, and then click **Execute**.
8. Highlight the script below **Step 3**, and then click **Execute**.
9. Highlight the script below **Step 4**, and then click **Execute**.
10. Highlight the script below **Step 5**, and then click **Execute**.
11. Highlight the script below **Step 6**, and then click **Execute**.
12. Highlight the script below **Step 7**, and then click **Execute**.
13. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
Which database engine component is responsible for thread management?	
Select the correct answer.	
	The SQL Server Network Interface
	The query optimizer
	The relational engine
	The SQL Server Operating System (SQLOS)
	The storage engine

Lesson 2
Windows Scheduling vs. SQL Server Scheduling

The SQL Server User Mode Scheduler is based on a non-preemptive scheduling architecture. A thorough understanding of the non-preemptive execution model in SQL Server is critical to troubleshoot performance issues that are related to CPU pressure and scheduling.

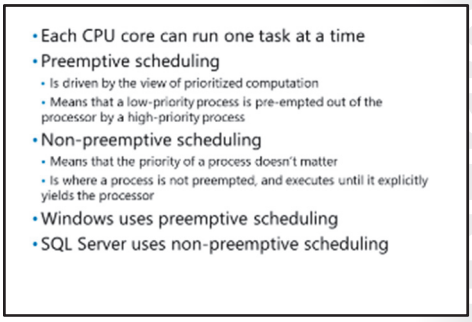
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the differences between preemptive and non-preemptive scheduling.
- Describe the SQL Server User Mode Scheduler.
- Describe the SQL Server execution model.
- Explain the life cycle of a user request.

Preemptive vs. Non-Preemptive Scheduling

In computing, the term "scheduling" refers to the management of allocation of resources to execute tasks—this is typically performed by an operating system. In the context of this lesson, you will learn about how SQL Server schedules CPU time for processes. A CPU can process one task at a time. A CPU scheduling decision is required when a process:



- Switches from running to waiting state.
- Switches from running to runnable state.
- Switches from waiting to runnable state.
- Is terminated.

Preemptive Scheduling

Preemptive scheduling is a priority-based type of scheduling. Each process is given a priority. A process that has a higher priority is given preference over a process that has a lower priority. Whenever a high-priority process requires CPU time, the low-priority process is preempted or forced out of the processor in the middle of its execution. In preemptive mode, the scheduler decides to move processes in and out of different states, and the client application has no control over the scheduler. The main advantages of preemptive scheduling are that:

- It is not possible for a process or an application to monopolize the CPU.
- The operating system makes sure that the CPU is shared among all executing processes.

Non-Preemptive Scheduling

Non-preemptive scheduling, or cooperative scheduling, is not a priority scheduling mechanism. Processes do not have levels of priority when compared to one another. A process that is running on CPU executes until it terminates or switches to a waiting state. One main disadvantage of non-preemptive scheduling is that a misbehaving process may hold the CPU, which prevents other processes from running. Some of the advantages of non-preemptive scheduling are that:

- The application has more control over CPU utilization.
- Non-preemptive scheduling is easy to implement.

Windows has used preemptive scheduling since Windows 95/Windows NT 4.

SQLOS uses a non-preemptive scheduling model. To avoid individual SQL Server processes occupying CPU resources for too long, SQLOS has a setting that defines the longest continuous period for which a process may use CPU without voluntarily yielding. This period is called the quantum. At the time of writing, the quantum value in SQL Server is 4 milliseconds.

 **Note:** You can see the quantum value in the **sys.dm_os_sys_info** DMV, in the column **os_quantum**.

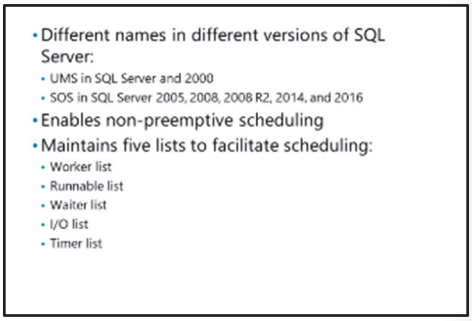
For more information about sys.dm_os_sys_info, see Microsoft Docs:

 **sys.dm_os_sys_info (Transact-SQL)**

https://aka.ms/Hnvncw

SQL Server on OS Scheduler and User Mode Scheduler

Versions of SQL Server up to and including SQL Server 6.5 relied on the host Windows operating system to perform scheduling. This imposed a performance limit on SQL Server because the scheduling needs that are specific to a relational database system could not be met by the general-purpose operating system scheduler that Windows used. The performance of SQL Server could be unpredictable because a process could be preempted by a higher-priority thread at any time. This issue was especially evident on SQL Server systems that had many concurrent client sessions, and on systems that had four or more CPUs.



To address this limitation, SQL Server 7.0 and 2000 have their own private scheduler, which is called the User Mode Scheduler (UMS). This component enables SQL Server to use a non-preemptive scheduling model, as opposed to the preemptive model that Windows used. In later versions of SQL Server (SQL Server 2005, 2008, 2008 R2, 2012, 2014, and 2016), the scheduler is referred to as the SQL Server on OS scheduler (SOS); this is an enhanced version of the UMS. Both UMS and SOS operate by attempting to keep the scheduling in SQL Server in user mode (as opposed to kernel mode).

The SOS/UMS Scheduler

The major components of SOS and UMS are sufficiently similar that they can be discussed together.

As discussed above, SOS/UMS is a SQLOS process that performs process scheduling for SQL Server. There is one SOS/UMS scheduler for each CPU in the system. The SOS/UMS scheduler works on non-preemptive or cooperative scheduling methodology. Its job is to accept a task and assign it to a worker. It assigns one worker at a time to a CPU. After a worker is assigned to a CPU, it works to complete the task that has been assigned without the scheduler's intervention. Each scheduler maintains five different lists for the purpose of thread scheduling:

- **The Worker List**. The worker list contains all available SOS/UMS workers. An SOS/UMS worker is a logical representation of an operating system thread or fiber within SQL Server. A worker is responsible for performing one task at a time, so it will switch to another task after it has finished the current task.
- **The Runnable List**. The runnable list contains workers that are ready to execute a task or a work request. A worker stays in the runnable list until it is signaled.
- **The Waiter List**. The waiter list has workers that are waiting for a resource. When a worker needs a resource that is owned by another worker, it puts itself in the waiter list. When the worker frees up a resource, it checks the waiter list for workers that are waiting on that resource and moves one to the runnable list as appropriate.
- **The I/O List**. The I/O list manages outstanding I/O requests. It scans the I/O list and removes completed I/O requests.
- **The Timer List**. The timer list manages SOS/UMS timer requests. When a worker waits for a resource for a particular time before timing out, the request is said to be a timer request and is added to the timer list.

For details of the operation of UMS, see the topic *Inside the SQL Server 2000 User Mode Scheduler* on MSDN. (Note that this white paper refers to the UMS in SQL Server 2000. There is no equivalent document that specifically covers the more recent SOS, but the core concepts are the same.)

Inside the SQL Server 2000 User Mode Scheduler

http://aka.ms/peagm9

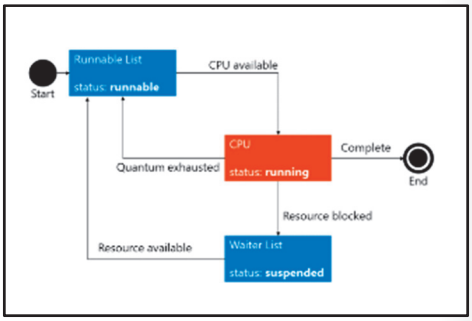
Note: SQL Server instances are most commonly configured to run workers on threads. You will see many references to threads as if they are the only worker type that SQL Server supports. In fact, SQL Server can be configured to use worker fibers rather than threads, although this is not generally recommended. This course concentrates on analyzing SQL Server instances by using threads.
To understand more about why threads are the recommended (and default) worker type, see the topic *The Perils of Fiber Mode* on TechNet.

The Perils of Fiber Mode

http://aka.ms/bs0i7a

SQL Server Execution Model

When a client application connects to SQL Server, session memory and a session identifier are assigned to the connection. The session identifier is used to track all of the server-side activity that is linked to a connection, and bring together all of the resources that are required to fulfill queries that the client application issues.



Note: The session identifier may be referred to as either a session ID or a server process ID (SPID). Both of these terms refer to the same value, and may be used interchangeably.

The SOS scheduler manages the execution of user requests. There is one scheduler instance for each CPU core. For example, a server that has four cores would have four schedulers by default.

An execution request from a client over a session is divided into one or more tasks and a worker is assigned to each task for its duration.

Each worker can be in one of the following three states:

- **Running**. A worker that is currently executing on a processor is said to be in a **running** state.
- **Suspended**. A worker is in a **suspended** state while it waits for a resource.
- **Runnable**. When a worker has finished waiting and is ready to execute again, it is said to be in a **runnable** state.

The **suspended** and **runnable** states correspond to two of the lists maintained by SOS that you learned about in the last topic:

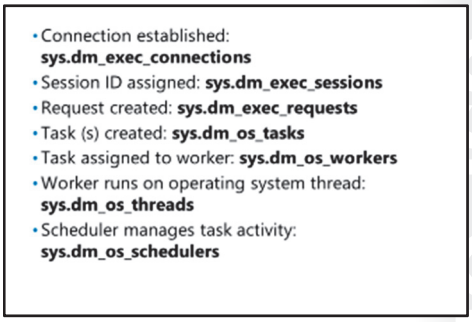
- **Waiter list**. For workers in the **suspended** state.
- **Runnable list**. For workers in the **runnable** state.

A worker will typically start its life at the bottom of the runnable list, which behaves as a First in, First out (FIFO) queue, with the **runnable** status. When the worker reaches the top of the runnable list, it will be assigned to a CPU, at which point it is removed from the runnable list and changes to the **running** status. The worker will work until it must wait for a resource, at which point it yields the CPU, is added to the waiter list, and its status is changed to **suspended**. The waiter list is unordered; workers may leave the waiter list at any time, after the resource that the worker needs becomes available. When the worker leaves the waiter list, it is returned to the bottom of the runnable list with a status of **runnable**. This cycle continues until the worker's task is complete.

Note: A worker may go from the **running** state to the bottom of the runnable list (with a **runnable** state) without visiting the waiter list if it yields the processor after using it for the duration of the quantum (4 milliseconds).

User Request Life Cycle

When an application or user sends an execution request to SQL Server, the database engine does the following:



1. SQL Server authenticates the connection request, establishes a connection with the client, and assigns a unique session ID. Connection details are available in the **sys.dm_exec_connections** DMV and session details are available in the **sys.dm_exec_sessions** DMV.
2. A request is sent from the client. The request details are available in the **sys.dm_exec_requests** DMV. It also displays the session status—**running**, **suspended**, or **runnable**—as discussed in the previous topic, *SQL Server Execution Model*.
3. SQL Server creates one or more tasks against the request. The **sys.dm_os_tasks** DMV returns one row for each active task. A task can be in any of the following states—**Running**, **Runnable**, **Pending**, **Suspended**, **Done**, or **Spinlock**.
4. A task is assigned to an available worker. A worker can be said to be the logical representation of a thread and is responsible for carrying out a SQL Server task. The worker information is available in the **sys.dm_os_workers** DMV.
5. A worker is assigned an available operating system thread for execution. The **sys.dm_os_threads** DMV provides details of threads that are running under the SQL Server process. The thread is then assigned a CPU by the SOS scheduler.
6. The user request is executed and the result is returned back to the user or client.

Demonstration: Analyzing the Life Cycle of a Thread

In this demonstration, you will see how to analyze the life cycle of a thread.

Demonstration Steps

1. Ensure that you have completed the previous demonstration in this module. Alternatively, start the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the D:\Demofiles\Mod01 folder as Administrator. In the **User Account Control** dialog box, click **Yes**, and wait for the script to complete.
3. If it is not already running, start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance by using Windows authentication.
4. If it is not already open, open the **Project.ssmsslnproj** solution in the D:\Demofiles\Mod01\Project folder.
5. In Solution Explorer, open the **Demo2i - Create hanging transaction.sql** script file, and click **Execute**.
6. Open the **Demo2ii - Start blocked transaction.sql** script file, and click **Execute**.
7. Open the **Demo2iii - DMV queries.sql** script file.

8. Substitute the values of **update_session_id** and **select_session_id** collected in the last two steps into the **VALUES** clause below **step 3**, click **Execute**.
9. Highlight the script below **Step 4**, and then click **Execute**.
10. Highlight the script below **Step 5**, and then click **Execute**.
11. Highlight the script below **Step 6**, and then click **Execute**.
12. Highlight the script below **Step 7**, and then click **Execute**.
13. Highlight the script below **Step 8**, and then click **Execute**.
14. Highlight the script below **Step 9**, and then click **Execute**.
15. On the **Demo2i - Create hanging transaction.sql** query, uncomment and execute the ROLLBACK command at the end of the file.
16. On the **Demo2ii - Start blocked transaction.sql,** note that the query is no longer blocked and results have been returned.
17. At the end of the demonstration, close SQL Server Management Studio without saving changes.

Question: Under what circumstances will a worker thread enter the runnable queue?

Question: When will a worker leave the runnable queue?

Lesson 3
Waits and Queues

As you learned in the previous lesson, whenever a task is not running, its status is either **Suspended** or **Runnable**.

For each current task that is not running, SQLOS records how long the task has been waiting—and the type of resource for which it is waiting—as a metadata value that is known as a wait type. This information is recorded for currently waiting tasks, and statistics are aggregated by wait type to enable the analysis of the most common wait types on an instance of SQL Server Database Engine.

An understanding of wait types and their meanings can be an invaluable tool when troubleshooting performance problems in SQL Server—because you can gain an insight into overall server performance, in addition to the performance of individual tasks.

Lesson Objectives

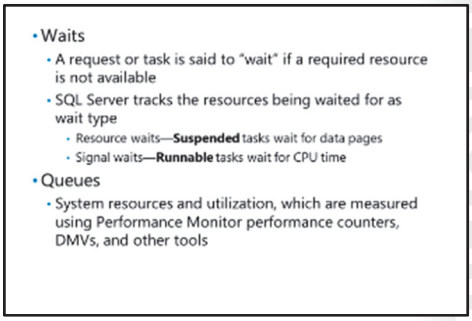
After completing this lesson, you will be able to:

- Describe waits and queues.
- Explain the usage of wait statistics.
- Describe wait types that are commonly found when performance tuning:
  - LCK_M_*
  - PAGELATCH_*
  - PAGEIOLATCH_*
  - CXPACKET
  - WRITELOG
- Describe other common wait types.
- Use waits and queues to investigate SQL Server performance.

Overview of Waits and Queues

Waits

A SQL Server wait records how long the task has been waiting, and the type of resource for which it is waiting.



Waits fall into two categories:

- **Resource waits**. When a task is on the waiter list and has a **Suspended** state, it is waiting for a resource (for example, an I/O read or write, or a lock that is held by another task). This is known as a resource wait.
- **Signal waits**. When a task is on the runnable list and has a **Runnable** state, it is waiting for CPU time. This is known as a signal wait.

Note: Remember that it is normal for tasks to wait during their execution, especially on a busy server. All SQL Server instances will experience some waiting. You must consider the cause and duration of a wait before treating it as a symptom of a performance problem.

Queues

When waits are considered on their own, they do not give a complete picture of overall system performance. They require some context in the form of information about the performance of the host operating system. This information is usually in the form of performance counters that are accessed either from Performance Monitor or through DMVs. These counters will show the levels of pressure on system resources, such as disk queues and processor utilization, and are generically referred to as queues.

Associating waits and queues provides a way to identify the resources that are under most pressure, which will help you to determine where to concentrate your performance-tuning efforts.

Using waits and queues as a performance-tuning method was first outlined in the SQL Server 2005 best practices article, *SQL Server 2005 Waits and Queues*. This document gives an excellent high-level overview of the waits and queues methodology.

SQL Server Best Practices Article – SQL Server 2005 Waits and Queues

http://aka.ms/nzz69y

Note: The document has not been updated for more recent versions of SQL Server, so some of the information that it provides about specific wait types and troubleshooting methods may be incomplete or no longer accurate.

Viewing Wait Statistics

- When a thread waits on a resource, SQL Server tracks wait information
- The aggregated wait statistics for all wait types are recorded in the **sys.dm_os_wait_stats** DMV
- The **sys.dm_os_waiting_tasks** DMV shows current wait statistics
- The aggregated wait statistics for current active sessions are recorded in **sys.dm_exec_session_wait_stats** (SQL Server 2016 only)

SQL Server regularly gathers statistics for waits that occur in the system. Whenever a thread waits on a resource, SQL Server records the resources that are being waited for, the wait time, and other details.

Cumulative statistics on the number, duration, and type of all of the waits that have been recorded since the server last started are known as wait statistics, or wait stats. The wait statistics for a SQL Server instance can be viewed using the **sys.dm_os_wait_stats** DMV.

Note: Cumulative wait statistics can be reset while a server is running, by using a DBCC command:

```
DBCC SQLPERF('sys.dm_os_wait_stats', CLEAR);
```

Real-time waiting information can be viewed by using the **sys.dm_os_waiting_tasks** DMV, which displays the waiter list. It lists wait type, wait duration, the session ID that is associated with the waiting tasks, a description of the resource for which the thread is waiting, and other details for all of the currently waiting tasks in the waiter list.

Wait statistics information for individual active sessions can be viewed by using the **sys.dm_exec_session_wait_stats** DMV. This offers a view of the same cumulative data that is shown in the **sys.dm_os_wait_stats** DMV, partitioned by currently active sessions.

 **Note: sys.dm_exec_session_wait_stats** is a DMV that was new in SQL Server 2016 and is not available in earlier versions.

The columns that are returned by **sys.dm_os_wait_stats** and **sys.dm_exec_session_wait_stats** include the total cumulative overall wait time for each wait type (**wait_time_ms**) and the cumulative signal wait time for each wait type (**signal_wait_time_ms**). These two values enable the resource wait time to be calculated.

```
Resource wait time (ms) = wait_time_ms - signal_wait_time_ms
```

Each of the following DMVs can be used to understand SQL Server performance in different ways:

- **sys.dm_os_wait_stats** can be used to identify frequent wait types and to tune or diagnose the system accordingly. One way to do this is to record and establish a wait statistics baseline, and then troubleshoot issues by analyzing the changes in baseline.
- **sys.dm_os_waiting_tasks** can be used to find out details of the resources for which tasks that are currently in the waiter list are waiting. This information can be useful when you are troubleshooting a performance problem while it is happening.
- **sys.dm_exec_session_wait_stats** can be used to understand the wait types that affect a particular Transact-SQL batch or statement. This information can be useful when comparing the performance of the same command on two different systems.

For more information about **sys.dm_os_wait_stats**, see the topic *sys.dm_os_wait_stats* in Microsoft Docs:

 **sys.dm_os_wait_stats**

http://aka.ms/kvkoru

For more information about **sys.dm_os_waiting_tasks**, see the topic *sys.dm_os_waiting_tasks* in Microsoft Docs:

 **sys.dm_os_waiting_tasks**

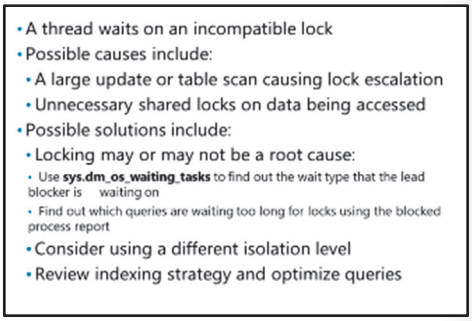
http://aka.ms/h36d40

For more information about **sys.dm_exec_session_wait_stats**, see the topic *sys.dm_exec_session_wait_stats* in Microsoft Docs:

 **sys.dm_exec_session_wait_stats**

http://aka.ms/njm5rg

LCK_M_* Wait Types



There are more than 60 wait types in this category; their names all begin with "LCK_M_."

This wait type indicates that a task is waiting to place a lock on a resource that is currently locked by another task. The characters at the end of the wait type name indicate the type of lock that the task is waiting to acquire. Some examples of wait types in this category are:

- **LCK_M_S**. The task is waiting to acquire a shared lock.
- **LCK_M_U**. The task is waiting to acquire an update lock.
- **LCK_M_X**. The task is waiting to acquire an exclusive lock.

You might be able to find more detail about the causes of these waits by examining **sys.dm_os_waiting_tasks**, which will give details of the resource for which the task is waiting, and which task is blocking it.

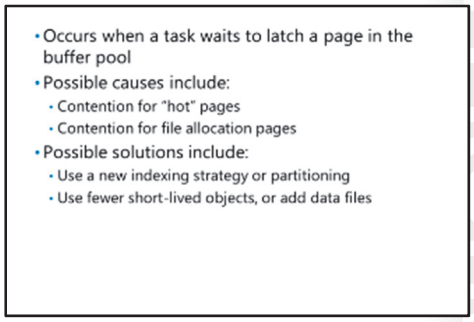
The blocked process report may also help you to identify the lead blocker in a larger blocking chain.

Some possible causes of wait types in this category include:

- **A large update or table scan that causes lock escalation**. This can be fixed by revisiting the indexing strategy; updating statistics; and using snapshot isolation, locking hints, and other strategies to avoid lock escalations.
- **Unnecessary shared locks on data being accessed**. This can be resolved by using a locking hint or a different isolation level.

Note: You will learn more about the role that locks play in the SQL Server concurrency model in Module 5 of this course, *SQL Server Concurrency*.

PAGELATCH_* Wait Types



The six wait types that belong to this category have names that begin with "PAGELATCH_."

In SQL Server, a latch is a lightweight locking mechanism that is used to protect a data structure in memory. A PAGELATCH wait indicates that a task is waiting to access a data page from the buffer pool that is already latched by another task. The characters at the end of the wait type name indicate the type of latch that the task is waiting to acquire. For example:

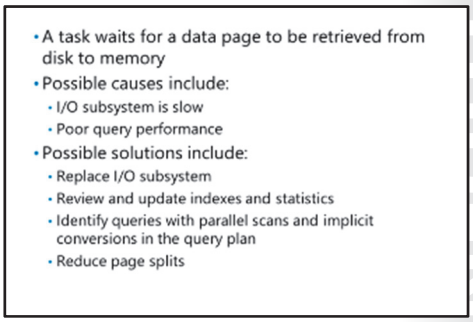
- **PAGELATCH_SH**. The task is waiting to acquire a shared latch.
- **PAGELATCH_UP**. The task is waiting to acquire an update latch.
- **PAGELATCH_EX**. The task is waiting to acquire an exclusive latch.

Note: You will learn more about the role that latches play in the SQL Server concurrency model in Module 5 of this course, *SQL Server Concurrency*.

There are many causes of PAGELATCH waits, but relatively few of them are likely to be the source of a performance problem. Some possible causes that might have performance impacts are:

- **Contention for "hot" pages**. If many tasks are writing to a table that has a clustered index on an identity column, there may be heavy contention for the data page that contains the rows that have been most recently added to the table. This kind of data page is sometimes referred to as a "hot" page. The contention might be addressed by a new indexing strategy, or a switch to table partitioning.
- **Contention for file allocation pages**. Special pages in a data file are used to track the allocation of data pages to database objects. If large numbers of objects are being created and dropped, there may be contention for the allocation pages; this is most typically a problem seen in **tempdb**. The contention might be addressed by changing the application design to use fewer short-lived objects, or (in the case of **tempdb**) by adding data files to the database.

PAGEIOLATCH_* Wait Types



The six wait types that belong to this category have names that begin with "PAGEIOLATCH_."

A PAGEIOLATCH wait indicates that a task is waiting to access a data page that is not in the buffer pool, so it must be read from disk to memory. The characters at the end of the wait type name indicate the type of latch that the task is waiting to acquire. For example:

- **PAGEIOLATCH_SH**. The task is waiting to acquire a shared latch.
- **PAGEIOLATCH_UP**. The task is waiting to acquire an update latch.
- **PAGEIOLATCH_EX**. The task is waiting to acquire an exclusive latch.

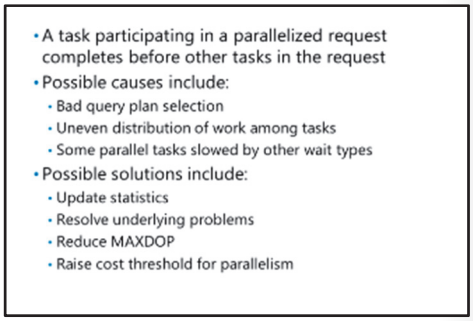
Note: You will learn more about how SQL Server manages I/O in Module 2 of this course, *SQL Server I/O*.

A high wait time for PAGEIOLATCH wait types may indicate an I/O bottleneck. It may further indicate that the I/O subsystem is slow to return the pages to SQL Server. However, just like any wait type, one should not jump to this conclusion without identifying the root cause of the bottleneck. Some of the possible root causes of PAGEIOLATCH waits and their solutions are:

- **Poorly performing queries**. Poorly written queries, mismanaged indexes, and out-of-date statistics result in far more I/O than necessary. This can be resolved by finding and tuning such queries and revisiting the indexing strategy.
- **Page splits, parallel scans, and implicit conversions**. Find and fix queries that are causing page splits, parallel scans, and implicit conversions. Revisit indexes and the FILLFACTOR setting to reduce page splits—in turn, reducing fragmentation.

CXPACKET Wait Type

The CXPACKET wait indicates that a task belonging to a request that has a parallel query plan has completed its work before other tasks that are working on other portions of the same request. CXPACKET waits can only occur on instances of SQL Server where parallel query plans are being executed. Any system that is running parallel queries will experience some CXPACKET waits. CXPACKET waits are not always an indicator that a performance issue is caused by parallelism; these waits can simply be an indicator that parallel plans are being used.

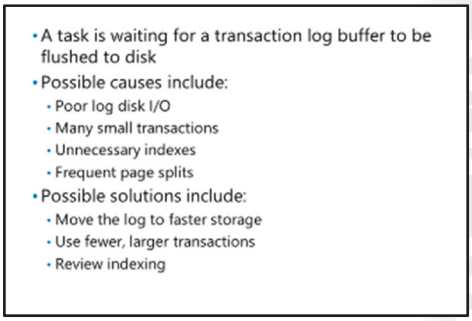


There are three root causes for CXPACKET waits:

- **A parallel query plan is not optimal**. There are circumstances where the query optimizer will select a parallel query plan when a serial query plan might give better performance. This is typically because the statistics on which the query optimizer bases its decisions are missing or out of date. Updating statistics may resolve the situation.
- **Uneven distribution of work among parallel tasks**. In this scenario, a parallel plan has been correctly identified as optimal for a request, but the way in which work has been divided among the tasks causes some of the tasks to do a disproportionate amount of the work. This is typically because of out-of-date or missing statistics.
- **Some parallel tasks are slowed by other wait types**. In this scenario, a parallel plan is optimal and work has been evenly divided among the parallel tasks, but some of the tasks are slower to complete because they must wait for another wait type. For example, in a parallel request that has two tasks, one task might be able to access all of the data pages that it needs from the buffer pool; whereas the other task might need to wait for all of the data pages that it requires to be retrieved from disk. Effectively, the second task is delayed by a PAGEIOLATCH wait. Identifying and resolving the underlying wait will resolve the CXPACKET wait.

It is possible to change the behavior of parallelism at server level, either by limiting the **maximum degree of parallelism** (MAXDOP) or by raising the **cost threshold for parallelism** setting. Adjusting these settings may reduce the incidence of CXPACKET waits, but application performance may not improve, because many queries can get performance gains from using a parallel plan.

WRITELOG Wait Type

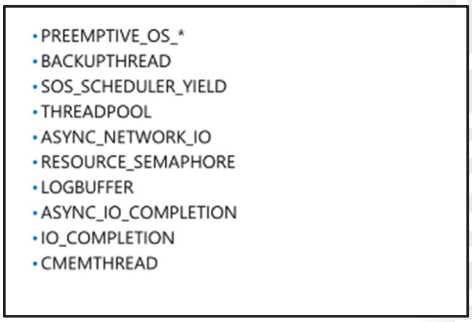


The WRITELOG wait type indicates that a task is waiting for a transaction log block buffer to be flushed to disk. A common operation that causes the flush of log block buffers to disk is when a transaction is committed. SQL Server implements a write-ahead logging mechanism to ensure the durability of the transactions. The transaction details are written to a transaction log synchronously for each committed transaction. The volume of transactions may exceed the capacity of the I/O subsystem to keep up with writes to the transaction log; in this scenario, WRITELOG waits may occur.

Causes for large numbers of WRITELOG waits include:

- **Many small transactions**. If an online transaction processing (OLTP) system writes many small transactions that have frequent commits, this increases the likelihood of WRITELOG waits occurring. This could be addressed by combining data changes into fewer, larger transactions.
- **Unnecessary indexes**. The greater the number of indexes on a table, the more data must be written to the transaction log each time the table changes. If indexes are unused by queries, they can be removed from the table.
- **Frequent page splits**. Frequent page splits that generate excessive log writes may result in WRITELOG waits. Tuning the index FILLFACTOR setting will help to reduce the number of page splits, which should in turn reduce the amount of transaction log activity and WRITELOG waits.
- **Slow log disks**. If the I/O subsystem that holds the transaction log files cannot keep up with the volume of log writes, WRITELOG waits will occur. Moving the transaction log files to faster storage is one solution.

Other Common Wait Types



SQL Server defines more than 800 wait types, not all of which are fully documented. The following is a short list of wait types that you might see on SQL Server instances for which you are responsible. They are seen less frequently than the wait types that have already been covered in this lesson:

- **PREEMPTIVE_OS_***. This wait type occurs when SQL Server calls out to the operating system for a specific task. In this case, the calling thread waits on any of the preemptive wait types that are depending on the task.
- **BACKUPTHREAD**. This wait type typically occurs when a task is waiting for a backup task to finish.

- **SOS_SCHEDULER_YIELD**. When a task voluntarily yields the CPU at the end of its quantum (4 milliseconds) and returns to the runnable list to resume execution, the wait type that is assigned SOS_SCHEDULER_YIELD. SOS_SCHEDULER_YIELD is always a signal wait.
- **THREADPOOL**. This wait type occurs when a task is waiting for a worker to be assigned. It can indicate a thread starvation situation or that the maximum worker setting is too low.
- **ASYNC_NETWORK_IO**. A task is put on the ASYNC_NETWORK_IO wait type when it is waiting for a client response; for example, when a client application is receiving a large result set.
- **RESOURCE_SEMAPHORE**. This wait type occurs when a task is waiting for a grant of memory for an operation that requires more memory than the task currently has assigned. A high incidence of this wait type may be seen on systems that are running concurrent queries that require a lot of memory.
- **LOGBUFFER**. The task is waiting for a log buffer to become available when it is flushing the log contents.
- **ASYNC_IO_COMPLETION**. The task is waiting for an I/O operation that does not relate to a data file to complete—for example, initializing a transaction log file or writing to backup media.
- **IO_COMPLETION**. This wait type occurs when the task is waiting for a synchronous I/O operation that does not relate to a data file to complete; for example, reading a transaction log for a rollback or for transactional replication.
- **CMEMTHREAD**. The task is waiting on a thread-safe memory object—generally, for inserting or removing a query execution plan from the plan cache. This may be a symptom of plan cache bloat.

The **sys.dm_os_wait_stats** in Microsoft Docs lists a brief definition for many wait types.

sys.dm_os_wait_stats

http://aka.ms/kvkoru

Demonstration: Monitoring Common Wait Types

In this demonstration, you will see how to monitor three common wait types:

- LCK_M_S
- WRITELOG
- PAGELATCH_*

Demonstration Steps

1. Ensure that you have completed the previous demonstration in this module. Alternatively, start the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the D:\Demofiles\Mod01 folder as Administrator. In the **User Account Control** dialog box, click **Yes**, and wait for the script to complete.
3. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance by using Windows authentication.
4. Open the **Project.ssmsslnprof** solution in the D:\Demofiles\Mod01\Project folder.
5. In Solution Explorer, expand **Queries**, open the **Demo2i - Create hanging transaction.sql** script file, and then click **Execute**.

6. Open the **Demo2ii - Start blocked transaction.sql** script file, and click **Execute**. Note the select_session_id.
7. Open the **Demo3 - instructions.sql** script file.
8. Highlight the script below **Step 3**, and then click **Execute**.
9. Add the **select_session_id** value collected in step 6 into the clause below **step 4**, and then click **Execute**.
10. On the **Demo2i - Create hanging transaction.sql** query, uncomment and execute the **ROLLBACK** command at the end of the file.
11. In **Demo3 - instructions.sql**, highlight the script below **Step 4**, and then click **Execute**. Note that this time, the LCK_M_S wait is included in the results because the session has finished waiting.
12. Highlight the script below **Step 7**, and then click **Execute**.
13. Highlight the script below **Step 8**, and then click **Execute**.
14. On the Start menu, type **Resource Monitor**, and then click **Resource Monitor**.
15. In Resource Monitor, on the **Disk** tab, notice that the **Disk Queue** graph for the D: drive (where the database files are located) is near zero.
16. In File Explorer, browse to **D:\Demofiles\Mod01**, right-click **start_load_2.ps1**, and then click **Run with PowerShell** to start the load. The load will run for approximately five minutes before stopping, and then press Enter.
17. Wait 30 seconds for the load to start.
18. In SQL Server Management Studio, select the query under the comment **Step 11**, and click **Execute**. Repeat this step several times over the course of a minute whilst the load is running.
19. In Resource Monitor, observe that the **Disk Queue** graph for the D: drive is elevated above zero.
20. At the end of the demonstration, close Resource Monitor, close SSMS without saving changes, and then close PowerShell.

Check Your Knowledge

Question	
Why is signal_wait_time_ms an important metric to review during performance issues?	
Select the correct answer.	
	If signal wait time is a high percentage of total wait time, this may indicate memory pressure.
	If signal wait time is a high percentage of total wait time, this may indicate I/O pressure.
	If signal wait time is a high percentage of total wait time, this may indicate CPU pressure.
	If signal wait time is a high percentage of total wait time, this may indicate a bottleneck in tempdb.

Lab: SQL Server Architecture, Scheduling, and Waits

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. The owners of the company have decided to start a new direct marketing arm. This has been created as a new company named Proseware Inc. Even though Proseware Inc. has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that can support both the existing workload and the workload from the new company.

Objectives

At the end of this lab, you will be able to:

- Explore the configuration of database engine components, schedulers, and NUMA.
- Monitor workload pressure on schedulers and observe the life cycle of a thread.
- Monitor and record wait statistics.

Estimated Time: 60 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Recording CPU and NUMA Configuration

Scenario

A new instance of SQL Server has been installed by the IT department at Adventure Works. In the first exercise, you need to document CPU and NUMA configuration.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Record CPU Configuration
3. Record CPU-Related Configuration Settings
4. Record NUMA Configuration
5. Record Distribution of Schedulers Across NUMA Nodes

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. Run Setup.cmd in the D:\Labfiles\Lab01\Starter folder as Administrator.

▶ Task 2: Record CPU Configuration

1. Start SQL Server Management Studio, and then open the project file **D:\Labfiles\Lab01\Starter\Project\Project.ssmssln** and the Transact-SQL file **Lab Exercise 01 - CPU and NUMA.sql**.
2. Under the heading for Task 1, write a query to return details of the CPU and hyperthreading configuration of the server that is hosting the MIA-SQL instance. Hint: look for CPU count and hyperthreading ratio values in the output of the **sys.dm_os_sys_info** DMV.

▶ Task 3: Record CPU-Related Configuration Settings

- Edit the query under the heading for Task 2 to return the following additional configuration values:
  - Max degree of parallelism
  - Max worker threads
  - Priority boost

▶ Task 4: Record NUMA Configuration

- Under the heading for Task 3, write a query to return details of the NUMA configuration for this server. Hint: the **sys.dm_os_nodes** DMV will provide the information that you need.

▶ Task 5: Record Distribution of Schedulers Across NUMA Nodes

- Under the heading for Task 4, write a query to return details of how user schedulers are distributed across NUMA nodes for this SQL Server instance. Hint: you will need to join **sys.dm_os_nodes** with **sys.dm_os_schedulers** on **node_id** and **parent_node_id**.

Results: At the end of this exercise, you will be able to:

Record CPU configuration.

Record NUMA configuration.

Exercise 2: Monitoring Schedulers and User Requests

Scenario

The new instance of SQL Server supports the existing workload and the workload from the new company. In this exercise, you will monitor the scheduling in SQL Server. You will monitor workload pressure on schedulers and the life cycle of threads.

The main tasks for this exercise are as follows:

1. Start the Workload
2. Monitor Workload Pressure on Schedulers
3. Monitor Task Status for User Requests
4. Stop the Workload

▶ Task 1: Start the Workload

- In the D:\Labfiles\Lab01\Starter folder, execute **start_load_exercise_02.ps1** by using Windows PowerShell®.

▶ Task 2: Monitor Workload Pressure on Schedulers

1. In Solution Explorer, open the **Lab Exercise 02 - Monitor Schedulers.sql** query file.
2. Under the heading for Task 2, write a query to return details of the visible online schedulers.
3. How do the column values change as the workload runs?
4. Can you make any deductions about the level of CPU pressure?

▶ Task 3: Monitor Task Status for User Requests

1. Under the heading for Task 3, write a query to return details of active user requests. Hint: to filter the output, only include sessions that have a **session_id** value of greater than 50. Sessions that have an ID of less than 50 are likely to be system sessions.
2. Which wait type are the user requests waiting for?
3. What does the wait type indicate?

▶ Task 4: Stop the Workload

- Highlight the code under the heading for Task 4, and then execute it.

Results: At the end of this exercise, you will be able to:

Monitor workload pressure on schedulers.

Monitor thread status for user requests.

Exercise 3: Monitoring Waiting Tasks and Recording Wait Statistics

Scenario

The additional workload is causing the new SQL Server instance to respond slowly. Users are occasionally complaining of poor performance and general slowness. In this exercise, you will monitor and record wait statistics.

The main tasks for this exercise are as follows:

1. Clear Wait Statistics
2. Check Current Wait Statistics
3. Start the Workload
4. Monitor Waiting Tasks While the Workload Is Running
5. Record Wait Statistics for Analysis
6. Stop the Workload

▶ Task 1: Clear Wait Statistics

1. In Solution Explorer, open the **Lab Exercise 03 - Waits.sql** query file.
2. Under the heading for Task 1, write a query to clear wait statistics. Hint: review the topic *Viewing Wait Statistics* in Lesson 3 of this module for assistance with this task.

▶ Task 2: Check Current Wait Statistics

- Under the heading for Task 2, write a query to select all rows and columns from the wait statistics DMV.

▶ Task 3: Start the Workload

- In the D:\Labfiles\Lab01\Starter folder, execute **start_load_exercise_03.ps1** by using Windows PowerShell.

▶ Task 4: Monitor Waiting Tasks While the Workload Is Running

- Under the heading for Task 4, write a query to view the waiter list. Hint: to filter the output, only include sessions that have a **session_id** value of greater than 50. Sessions that have an ID that is less than 50 are likely to be system sessions.

Note: Note the wait type(s) for which the tasks are waiting.

▶ Task 5: Record Wait Statistics for Analysis

1. Execute the first query under the heading for Task 5 to capture a snapshot of wait statistics into a temporary table called **#wait_stats_snapshot**.
2. The second query under the heading for Task 5 compares the snapshot that was captured in **#wait_stats_snapshot** with the current wait statistics.
   Amend this query to order the results by the change to **wait_time_ms** between the snapshot and the current wait statistics, descending, and exclude wait types where there is no change.

▶ Task 6: Stop the Workload

- Highlight the code under the heading for Task 6, and then execute it.

Results: At the end of this exercise, you will be able to:

Monitor the waiting tasks list.

Capture and review wait statistics.

Question: The output from **sys.dm_os_schedulers** (accessed in the last task in Exercise 1) shows some schedulers as HIDDEN ONLINE. What is meant by hidden schedulers?

Module Review and Takeaways

In this module, you learned about SQL Server architecture and the various components of the database engine. You learned about preemptive and non-preemptive scheduling. You learned how to monitor the user request life cycle and workload pressure. You learned how to view CPU and NUMA configuration. You also learned about the importance of wait statistics and how to analyze them.

 **Best Practice:** It is good practice to keep a baseline of wait statistics data; this makes it much easier to determine when a trend has changed.

Sequencing Activity

Put the following steps of the query life cycle in order by numbering each to indicate the correct order.

	Steps
	Command received from client application.
	Command is parsed into a parse tree.
	Parse tree is bound to database objects.
	Plan hash is generated.
	Check for existing query plan.
	Query Optimizer selects a suitable plan.
	Query plan is executed.
	Results are returned to the client application.

Tools

Several SQL Server MVPs have published useful tools for tracking waiting tasks and wait statistics:

1. Adam Machanic's **sp_WhoIsActive**.
2. Glenn Berry's performance monitoring scripts.
3. Jonathan Kehayias and Erin Stellato at sqlskills.com have written an unofficial update to the *SQL Server 2005 Waits and Queues* document, which is called *SQL Server Performance Tuning Using Wait Statistics: A Beginner's Guide*. It is available as a PDF file from sqlskills.com.

Module 2

SQL Server I/O

Contents:

Module Overview

In the course of normal operation, Microsoft® SQL Server® relies heavily on reading and writing data to and from disk. As a consequence, the performance characteristics of the I/O subsystem are critical to SQL Server performance. This module covers core I/O concepts, storage area networks (SANs), and performance testing—all focusing on SQL Server I/O operations.

Objectives

After completing this module, you will be able to:

- Describe core I/O concepts.
- Select a storage solution for a SQL Server installation.
- Test storage performance by using the Diskspd utility.

Lesson 1
Core Concepts of I/O

This lesson focuses on I/O terminology and different concepts that are related to storage systems. To approach an I/O performance issue logically, knowledge of storage concepts is important.

You will learn about three measures of storage I/O performance: I/O operations per second (IOPS), disk throughput, and disk latency factor. The lesson concludes with a discussion of disk technology and common redundant array of independent disks (RAID) levels.

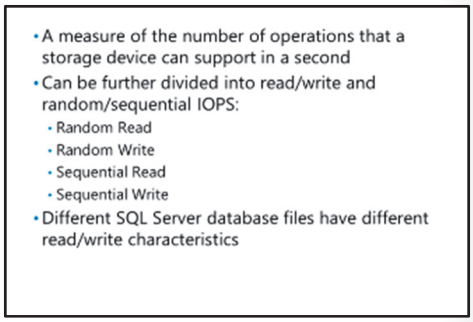
Lesson Objectives

After completing this lesson, you will be able to:

- Use IOPS to assess storage performance.
- Use throughput to assess storage performance.
- Use latency factor to assess storage performance.
- Understand the differences between magnetic disk and solid-state drive (SSD).
- Describe common RAID levels.

Input/Output Operations Per Second

IOPS measures the number of physical transfers that a storage device—such as a hard disk drive (HDD), an SSD, or a SAN—can perform in one second. For example, a device that has 10,000 IOPS can perform 10,000 transfers in one second. The higher the IOPS, the better the performance of the Windows® I/O subsystem.



The two main performance characteristics that are measured are sequential operation and random operation. Sequential operations access storage devices in an ordered sequence; that is, one after the other. Random operations access storage devices in an impromptu manner. Measures of IOPS are further divided by reads and writes, as described in the following table:

IOPS Measure	Description
Total IOPS	Total number of I/O operations per second
Random Read IOPS	Average number of random read I/O operations per second
Random Write IOPS	Average number of random write I/O operations per second
Sequential Read IOPS	Average number of sequential read I/O operations per second
Sequential Write IOPS	Average number of sequential write I/O operations per second

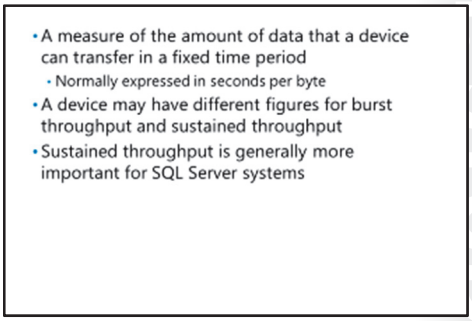
IOPS can be measured by using the following performance counters:

- **Physical Disk** or **Logical Disk: Disk Reads/Sec**. The current number of read I/O operations per second.
- **Physical Disk** or **Logical Disk: Disk Writes/Sec**. The current number of write I/O operations per second.

SQL Server uses a mixture of sequential and random reads and writes. Database data files will typically have a mixture of random reads and writes; the exact split between read activity and write activity will vary by workload. Transaction log file activity will typically be mostly sequential writes; new log entries are always added to the end of the file.

Throughput

In the context of I/O, throughput refers to the amount of data that a storage device can transfer in a known period of time—typically, one second. The greater the throughput figure, the more data the device can transfer.



A storage device may have different throughput figures for burst transfers and sustained transfers:

- Burst throughput measures the maximum throughput that is achievable for a small amount of data under ideal conditions, perhaps when the data is already in a cache on the device.
- Sustained throughput measures the maximum throughput that is achievable for amounts of data that are larger than the device's cache.

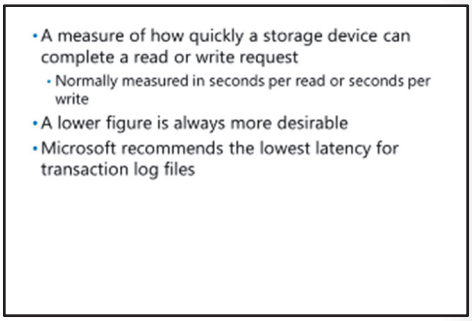
Throughput can be measured by using the following performance counters:

- **Physical Disk** or **Logical Disk: Disk Read Bytes/Sec**. The current rate at which bytes are transferred from the disk during read operations.
- **Physical Disk** or **Logical Disk: Disk Bytes/Sec**. The current rate at which bytes are transferred to or from the disk during write or read operations.
- **Physical Disk** or **Logical Disk: Disk Write Bytes/Sec**. The current rate at which bytes are transferred to the disk during write operations.

Sustained throughput figures will typically be of more importance when designing storage for SQL Server systems.

Latency Factor

In the context of I/O, latency factor refers to the time that has elapsed between an I/O request being issued and a response being received from the storage device. It is the time that a disk takes to begin to complete a read operation or a write operation:



- In the case of storage devices that have mechanical disks, the latency factor is directly related to the time taken to rotate the disk platter and to move the disk head into position to access sectors that contain the target data.
- In the case of flash memory devices such as SSDs (which have no moving parts), the latency factor is much lower than for mechanical disks, because it reflects the time taken for the disk to return data from memory chips. Higher latency factors may occur on flash memory devices if the demand for I/O operations exceeds the ability of the device to respond to them.

Whatever the design of the storage device, a lower latency factor figure is more desirable.

Latency can be measured by using the following performance monitor counters:

- **Physical Disk** or **Logical Disk: Avg. Disk sec/Read**. The average time, in seconds, of a read of data from the disk.
- **Physical Disk** or **Logical Disk: Avg. Disk sec/Write**. The average time, in seconds, of a write of data to the disk.

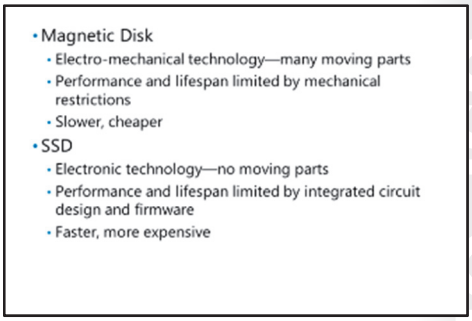
As part of a 2007 best-practices paper, Microsoft published the following guidance for target **Avg. Disk sec/Read** and **Average Disk sec/Write** figures on SQL Server systems:

SQL Server File Type	Ideal Values for Avg. Disk sec/Read and Average Disk sec/Write
Log files	1–5 ms (ideally 1 ms on arrays with cache)
OLTP data files	4–20 ms (ideally 10 ms or less)
Decision Support System/OLAP data files	30 ms or less—this figure comes with a note that latencies can vary significantly as system activity varies

Note: Remember, these figures are only guidelines. It is down to your organization to determine what constitutes acceptable performance for your applications.

Magnetic Disk and SSD

Magnetic disks (also referred to as hard disk drives or HDDs) consist of one or more circular disks that are coated with a magnetically sensitive metal oxide on one or both sides. These oxide-coated disks are referred to as platters. All of the platters in a disk are mounted on a central spindle, which is rotated by an electronically controlled motor at speeds of up to 15,000 rpm. Each magnetically sensitive side of a platter has an electromagnetic read/write head that can be moved across the surface of the platter to read and write data as the platter rotates.

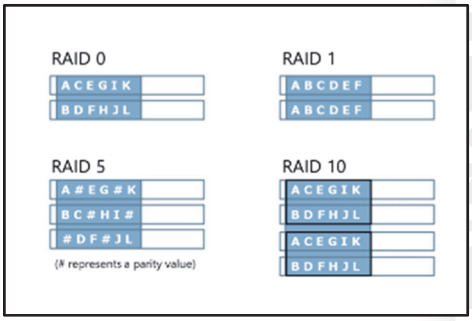


SSDs use integrated circuits to store data persistently on flash memory chips. Flash memory is an electronic persistent memory that can be electrically erased and reprogrammed. Unlike magnetic disks, SSDs have no moving parts. As a consequence, SSDs are generally faster than magnetic disks in all three areas of I/O performance that were discussed earlier in this lesson (IOPS, throughput, and latency factor). SSDs are not subject to issues of mechanical failure, but they have a lifespan that is limited by the number of read/write cycles that individual memory cells support.

Magnetic disks are the older, more established technology, so they have a lower cost per gigabyte of storage space, and wider availability, than comparable SSDs.

RAID Levels

Many storage solutions use RAID hardware to provide fault tolerance through data redundancy and, in some cases, to improve performance. You can also implement software-controlled RAID 0, RAID 1, and RAID 5 by using the Windows Server® operating system, and other levels may be supported by third-party SANs. Commonly used types of RAID include:



RAID 0

Disk striping. A stripe set consists of space from two or more disks that is combined into a single volume. The data is distributed evenly across all of the disks, which improves I/O performance, particularly when each disk device has its own hardware controller. RAID 0 offers no redundancy, and if a single disk fails, the volume becomes inaccessible.

RAID 1

Disk mirroring. A mirror set is a logical storage volume that is based on space from two disks, with one disk storing a redundant copy of the data on the other. Mirroring can provide good read performance, but write performance can suffer. RAID 1 is expensive in terms of storage because 50 percent of the available disk space is used to store redundant data.

RAID 5

Disk striping with parity. RAID 5 offers fault tolerance through the use of parity data that is written across all of the disks in a striped volume that consists of space from three or more disks. RAID 5 typically performs better than RAID 1. However, if a disk in the set fails, performance degrades. RAID 5 is less costly in terms of disk space than RAID 1 because parity data only requires the equivalent of one disk in the set to store it. For example, in an array of five disks, four are available for data storage, which represents 80 percent of the total disk space.

RAID 10

Mirroring with striping. In RAID 10, a nonfault-tolerant RAID 0 stripe set is mirrored. This arrangement delivers the excellent read/write performance of RAID 0, combined with the fault tolerance of RAID 1. However, RAID 10 can be expensive to implement because, like RAID 1, 50 percent of the total space is used to store redundant data.

Consider the following points when planning files on RAID hardware:

- Generally, RAID 10 offers the best combination of read/write performance and fault tolerance, but is the costliest solution.
- Write operations on RAID 5 can sometimes be relatively slow compared to RAID 1 because of the need to calculate parity data (RAID 5). If you have a high proportion of write activity, RAID 5 might not be the best candidate.
- Consider the cost per gigabyte. For example, implementing a 500 GB database on a RAID 1 mirror set would require (at least) two 500 GB disks. Implementing the same database on a RAID 5 array would require substantially less storage space.
- Many databases use a SAN, and the performance characteristics can vary between SAN vendors and architectures. For this reason, if you use a SAN, you should consult with your vendors to identify the optimal solution for your requirements. When considering SAN technology for SQL Server, always look beyond the headline I/O figures quoted and consider other characteristics such as latency.

Check Your Knowledge

Question	
Which I/O performance measure reflects the responsiveness of a storage device to an I/O request?	
Select the correct answer.	
	Throughput
	Latency factor
	IOPS

Lesson 2
Storage Solutions

There are several designs of storage solution in common use in IT environments. When assessing the performance characteristics of an I/O system, it is important to understand the principles of the underlying storage solution.

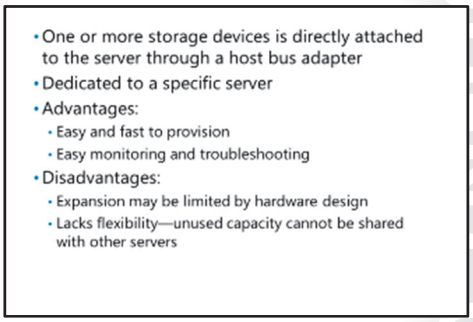
This lesson outlines common storage solutions, how these solutions differ, and some of their strengths and weaknesses.

Lesson Objectives

At the end of this lesson, you will be able to:

- Describe direct-attached storage (DAS).
- Describe a SAN.
- Describe Windows Storage Spaces.
- Describe SQL Server files in Azure Blob storage.
- Explain how you might go about choosing between storage solutions.

Direct-Attached Storage



DAS refers to a storage system in which one or more storage devices are attached directly to a server through a host bus adapter. DAS is only accessible to one computer and is not connected to another system. A typical DAS configuration consists of enclosures or external drive bays that hold several drives that are connected directly to the system.

DAS provides better performance than network storage because SQL Server does not have to cross a network to read and write data. DAS is recommended for some of the high-performance enterprise applications; for example, Microsoft recommends DAS for Microsoft Exchange Server.

Advantages

The following are some advantages of DAS:

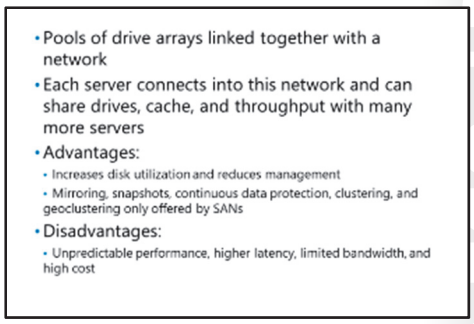
- Easy and fast to provision.
- Easy monitoring and troubleshooting.
- Easy to support.

Disadvantages

The following are some disadvantages of DAS:

- Inability to share unused storage. DAS is connected to only one server, so the unused storage cannot be shared with other computers.
- A server chassis can support a limited number of drive enclosures. Adding a dedicated array chassis outside the server adds capacity, but lacks flexibility.

Storage Area Network



A SAN is a pool of drive arrays that are linked together by using a network. A server can connect to a SAN, sharing drives, cache, and throughput with other connected servers. SANs are centrally managed and are based on Fiber Channel or Internet SCSI (iSCSI) technology. Typically, Fiber Channel SANs are complex and expensive. iSCSI encapsulates SCSI commands into IP packets for transmission over an Ethernet connection instead of a fiber connection. This reduces the cost, complexity, and maintainability that are associated with the Fiber Channel design, but the performance of the SAN can be limited by available network bandwidth.

A SAN consists of three main components: cables, a host bus adapter, and a switch. Each switch is interconnected to a storage system on the SAN. The physical connection must be good enough to handle the bandwidth requirement during peak data load.

The resources in the SAN can be shared, even down to the individual disk level. Pools of hard drives can be set and each server can be assigned individual spaces on each pools. For example, you can create a pool of 20 hard drives in a RAID 10 configuration, each with 1 terabyte of capacity, for a total capacity of 10 TB. The administrator can then assign five 2-TB units, called logical unit numbers (LUNs), to five different servers. The servers will share the same physical hard drives, but they will not be able to see each other's data; the SAN hardware manages that access.

Advantages

The following are the advantages of SAN:

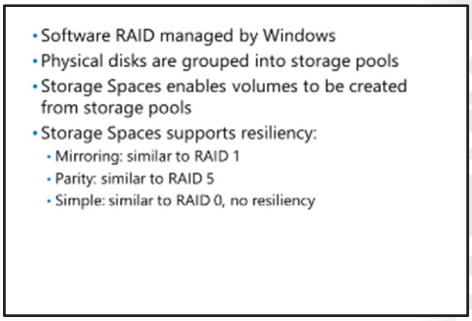
- Shared storage:
  - Increases disk utilization.
  - Reduces management by making it easier to create new volumes and dynamically allocate storage.
  - Means that you can create diskless servers that boot from SAN only.
  - Works well with hardware virtualization.
- Advanced features:
  - High-availability features, such as clustering and geoclustering, can be set up by using SAN.
- Performance:
  - An almost unlimited number of spindles, controllers, and caches can be put together to meet the requirements.

Disadvantages

The following are some of the disadvantages of using SAN:

- Unpredictable performance:
  - When you share your disks, controllers, and fiber switches between several servers, it is very difficult to have predictable performance.
- Higher latency:
  - The I/O needs to travel further; there are added layers of switches, cabling, and ports.
  - PCI Bus -> HBA -> FC switches -> FC ports -> array processors -> disks.

Windows Storage Spaces



Storage Spaces is a form of software-based RAID. Storage Spaces enables you to virtualize storage by grouping industry-standard disks into storage pools. You can then create virtual disks, called storage spaces, from the available capacity of the storage pool.

Storage pools can be created from magnetic disks and SSDs that have IDE, Serial ATA (SATA), serial attached SCSI (SAS), or USB interfaces. After creating a storage pool, if you run low on disk space, you can add more disks to the pool, increasing the available storage capacity without needing to copy or move data. You can implement storage tiers within storage pools, enabling you to move frequently accessed data to faster disks within the pool.

Storage Spaces is a variation of DAS and has similar advantages and limitations.

Resiliency is built into Storage Spaces, and there are three different levels available depending on your storage needs:

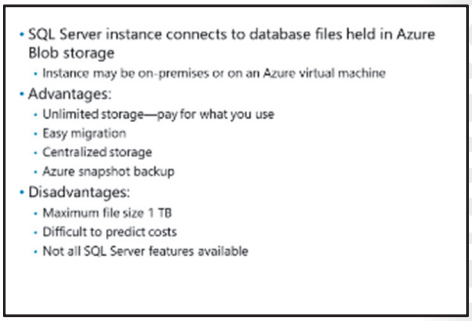
- **Mirroring**. Writes multiple copies of data across multiple disks in a similar way to a RAID 1 disk set. Mirroring offers maximum protection for your data in the event of a disk failure, and gives good read and write performance, but disk capacity is reduced.
- **Parity**. Writes a single copy of data striped across the disks along with a parity bit to enable data recovery. Gives good capacity and read performance, but write performance is generally slower due to the need to calculate parity bits. Parity is similar to a RAID 5 disk set.
- **Simple**. Stripes data across all disks as a single copy with no parity and is technically similar to a RAID 0 disk set. Simple maximizes storage capacity and gives high performance, but offers no resiliency. Losing a disk will mean that data is lost.

For more information about Storage Spaces, see the topic *Storage Spaces Overview* on TechNet.

Storage Spaces Overview

http://aka.ms/dyg4dk

SQL Server Data Files in Microsoft Azure



SQL Server 2014 and 2016 support the creation of databases where the database files are held inside the Microsoft Azure™ cloud computing service. The Azure Storage service enables you to store large binary files—known as "blobs"—in the Azure cloud. SQL Server can connect directly to database files that are held in Azure Blob storage as if they were local to the database server. Database instances that use this feature may run on-premises or from an Azure virtual machine.

Advantages

Advantages of storing SQL Server data files in Azure include:

- **Cost and flexibility**. There is no practical upper limit on the amount of data that you can add to Azure Blob storage, and you are only charged for what you use.
- **Stepping stone to cloud migration**. If your organization is hoping to move database resources to the cloud, using Azure Blob storage enables you to start the process in a way that is transparent to applications and users.
- **Centralized storage**. Keeping database files in Azure Blob storage may reduce the need for you to copy large database files between geographical locations.
- **Snapshot backup**. Azure snapshots enable almost instantaneous backup of database files.

Disadvantages

Disadvantages of storing SQL Server data files in Azure include:

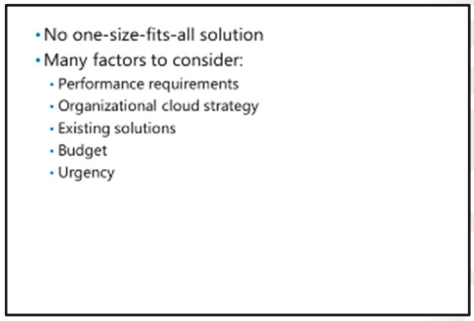
- **File size limits**. Individual files in Azure Blob storage may not be larger than 1 TB.
- **Difficulty in predicting Azure charges**. You may find it difficult to predict your likely charges for Azure Blob storage, especially for new or fast-growing databases.
- **Not all SQL Server features are supported**. Azure Blob storage cannot be used for FILESTREAM data; because of this, in-memory online transaction processing (OLTP) is not available when using Azure Blob storage.

For more information about this topic, see the topic *SQL Server Data Files in Microsoft Azure* on MSDN.

SQL Server Data Files in Microsoft Azure

http://aka.ms/m1jwx4

Selecting a Storage Solution



There is no single storage solution that will be the best fit for every SQL Server installation. To select a storage solution, you will need to evaluate requirements, examine the strengths and weaknesses of the options that are available to you, and assess how closely they match your organization's IT strategy.

When making this evaluation, the following are some factors that you might consider:

- **Application performance requirements**. What are the performance characteristics that the SQL Server instance is required to support? This might be covered by a service level agreement.
- **Organizational cloud strategy**. Does your organization have a policy about moving systems and services to the cloud? Does your organization have legal or contractual requirements for data protection that exclude the use of cloud services?
- **Existing storage solutions**. Has your organization already invested in a large SAN installation? Will the SAN be able to support your I/O performance requirements?
- **Budget**. What budget is available to provision a storage solution?
- **Urgency**. How quickly is the solution required? A large SAN installation might take many months; cloud storage will be available within minutes.

Question: Which storage solution does your organization most commonly use? What are its benefits? What are its limitations?

Lesson 3
I/O Setup and Testing

In this lesson, you will learn about various points to consider when configuring an I/O system for SQL Server. You will also learn how to test disk configuration and benchmark disk performance.

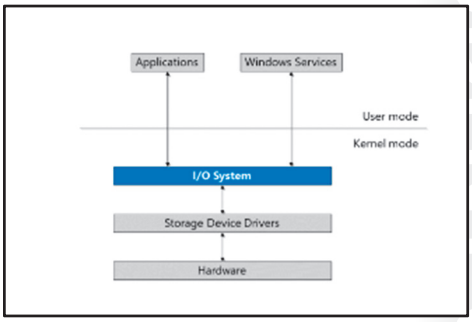
Lesson Objectives

After this lesson, you will be able to:

- Describe the Windows I/O system.
- Describe disk types.
- Describe mount points.
- Explain partition alignment and NTFS file system allocation unit size, and the effect that they can have on system performance.
- Carry out tests to benchmark disk performance.

Windows I/O System

The Windows I/O system is a subsystem of the Windows operating system through which applications can access storage devices. The I/O system runs in kernel mode (meaning that it has unrestricted access to system hardware), and communicates with storage devices through device drivers.



The I/O system consists of several subcomponents, the details of which are beyond the scope of this course. I/O system subcomponents communicate through a packet-driven protocol called I/O request packet, or IRP.

The I/O system defines several operations for storage devices—such as open, close, read, and write—which storage device drivers must implement. This abstraction enables user-mode applications to interact with storage devices through a standard interface.

Disk Types

Before Windows can use a storage device, one or more partitions must be created on the device. After a partition has been formatted with a valid file system, it is ready for use by Windows and is referred to as a volume. When discussing device partitioning, Windows supports two types of disk: basic disks and dynamic disks.

- Storage devices must be partitioned by using an MBR or GPT partition table
- A partition formatted with a file system is referred to as a volume
- Storage devices may be treated either as:
  - Basic disk:
    - Supports logical and extended partitions
    - Backward-compatible with MS-DOS
  - Dynamic disk:
    - Supports many complex configurations

Basic Disks

Basic disks have been supported since the MS-DOS operating system. A basic disk may contain primary partitions and extended partitions. An extended partition may contain one or more logical drives. The partition table for a basic disk may be held in a master boot record (MBR) or GUID partition table (GPT) format. Not all versions of Windows support GPT, but MBR is backward-compatible.

When working with a basic disk, you can:

- Create and delete primary and extended partitions.
- Create and delete logical drives within an extended partition.
- Format a partition and mark it as active.

Dynamic Disks

Dynamic disks support more complex configurations than basic disks, including, but not limited to, simple volumes, spanned volumes, striped volumes, mirrored volumes, and RAID 5 volumes. Dynamic disks support both MBR and GPT partitioning schemes. A dynamic disk MBR is similar to basic disk, except that dynamic disk allows only one primary partition and a hidden partition. The dynamic disk GPT layout is similar to basic disk, except that it contains one Logical Disk Manager (LDM) partition entry instead of a 1-*n* partition type in a basic disk GPT. It also contains a hidden LDM database partition with a corresponding GUID partition entry for it.

Dynamic disks use a database to track information about dynamic volumes and other dynamic disks in a server. Each dynamic disk stores a replica of the dynamic database. The database can be used to repair a dynamic disk from corruption.

Dynamic volumes can have noncontiguous extents on one or more physical disks. Dynamic disks support LDM, Virtual Disk Service (VDS), and other associated features. These features enable you to perform tasks, such as converting basic disks and creating fault-tolerant systems.

The dynamic disks support the following operations:

- Creating and deleting simple, spanned, striped, mirrored, and RAID 5 volumes.
- Extending a simple or spanned volume.
- Removing a mirror from a mirrored volume, or breaking a mirrored volume into two volumes.
- Repairing mirrored or RAID 5 volumes.
- Reactivating a missing or offline disk.

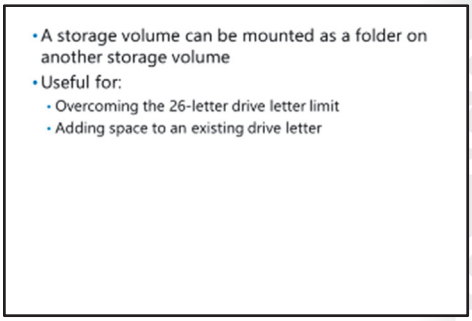
For more information about disk types, see the topic *Basic and Dynamic Disks* on MSDN.

Basic and Dynamic Disks

http://aka.ms/sa118i

Mount Points

In Windows, storage volumes are typically mounted with a drive letter; for example, C. This lettering system imposes a maximum of 26 volumes that can be mounted in this way, one for each letter of the modern Latin alphabet.



Mount points provide a way around this limit. A mount point is a special object in an NTFS file system that enables an empty directory in one storage volume to point to the root of another storage volume. After it is completed, this action is invisible to applications, which can continue to interact with the mount point as if it were a normal folder.

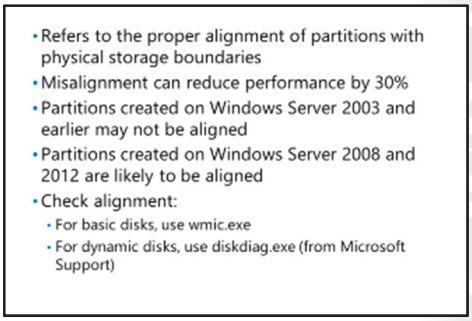
In addition to enabling a system to mount more than 26 volumes, mount points can also be useful to add more storage to a system without adding drive letters (which might require software to be reconfigured).

Mount points can be configured in three ways:

- Using the Microsoft Management Console Disk Management utility.
- Using mountvol.exe from the command-line interface.
- Using Win32 application programming interface (API) calls from an application.

Partition Alignment

Partition alignment refers to proper alignment of partitions to the available disk storage boundaries. Disk arrays reserve the first 63 sectors, each 512 bytes in size, for the master boot records.



Before Windows Server 2008, the operating system did not take account of this offset; this could result in misalignment of the partition with stripe units, disk controller, and cache segment lines with fundamental physical boundaries. Misalignment can reduce performance by up to 30 percent for magnetic disks. Performance effects of misalignment on SSDs are less severe.

In Windows Server 2008 and 2012, partition alignment defaults to 1,024 KB, which correlates well with the available stripe size of 64 KB, 128 KB, 256 KB, 512 KB, and 1,024 KB.

The correct offset can be calculated as partition offset or stripe size.

Note that, although the default value is correct in many circumstances, partition alignment values may need to be customized where vendor-specific settings are required, or storage administrators are setting partition offset manually when creating volumes.

The most accurate method to check partition alignment values for basic disks is by using the wmic.exe command-line utility. The starting offset value is returned in bytes.

Check Partition Alignment by Using wmic.exe

```
wmic partition get BlockSize, StartingOffset, Name, Index
```

For dynamic disks, you must use **diskdiag.exe** to get accurate information. This executable file is available by contacting Microsoft Customer Support.

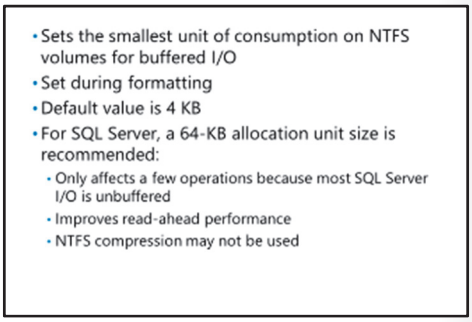
For more information about this topic specifically as it relates to SQL Server, see the Microsoft best-practice paper *Disk Partition Alignment Best Practices for SQL Server* on TechNet. Note that this paper references SQL Server 2008 and Windows Server 2003, so some advice may be out of date. However, the core technical concepts are still correct.

Disk Partition Alignment Best Practices for SQL Server

http://aka.ms/i82zpu

NTFS Allocation Unit Size

NTFS allocation unit size is defined as the smallest unit of consumption on the disk for buffered I/O. By default, the NTFS allocation unit size is 4 KB. This means that if you create a file on the disk that is formatted with default NTFS allocation unit size, the file will consume 4 KB even if the data in the file occupies less than 4 KB. The NTFS allocation unit size is set when the disk is formatted during the initial setup.



The recommended NTFS allocation unit size for SQL Server is 64 KB. The data is stored in SQL Server as 8-KB pages. After the first allocation of pages from mixed extent for any object in a database, the next set of pages are allocated as uniform extents. An extent is eight contiguous 8-KB pages, resulting in 64 KB of space. This means that any read operation on a table gets higher performance with read-ahead reads when the allocation unit size is set to 64 KB.

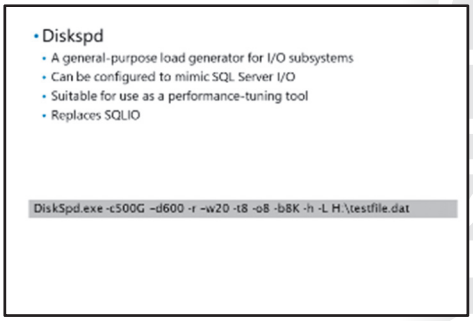
Note: Native NTFS compression may not be used on volumes that have an allocation unit size of more than 4 KB. However, it is not recommended to use NTFS compression on volumes that contain SQL Server database files.

Most SQL Server I/O activity takes the form of unbuffered I/O, which is not affected by the NTFS allocation unit size.

Storage Performance Testing

There are various tools and techniques for benchmarking and performance testing of storage systems. One such utility is Diskspd.

Diskspd



Diskspd is a command-line utility, developed by Microsoft, for load generation and performance testing of storage I/O subsystems. Although Diskspd is not specifically designed for simulating SQL Server I/O activity, it can be configured to do so. Diskspd replaces SQLIO, an older utility that had a similar purpose.

Diskspd is suitable for use when performance-tuning I/O subsystems because, for a given set of parameters, the load that is generated will always be the same.

Diskspd is available as a download from Microsoft.

Diskspd Utility: A Robust Storage Testing Tool (superseding SQLIO)

http://aka.ms/diskspd

Detailed instructions on simulating SQL Server I/O activity, and instructions on how to interpret the results, are included in a document that is packaged with the Diskspd download (**UsingDiskspdforSQLServer.docx**).

Diskspd accepts parameters to determine test file size, balance of read/write activity, read/write block size, and many other options. See the documentation for details.

The following example runs a test for 15 seconds using a single thread to drive 100 percent random 64-KB reads at a depth of 15 overlapped (outstanding) I/Os to a regular file:

The following example runs a test for 15 seconds using a single thread to drive 100 percent random 64-KB reads at a depth of 15 overlapped (outstanding) I/Os to a regular file:

Diskspd Example 1

```
DiskSpd –d300 –F1 –w0 –r –b64k –o15 c:\testfile.dat
```

You will want to customize the parameters that you pass to Diskspd to reflect the I/O characteristics of your SQL Server application. The UsingDiskspdforSQLServer.docx document includes examples of different configurations for different load types.

This example is configured as follows:

- **-c500G** (a 500-GB or 0.5365-TB file). The file size should be greater than the storage solution's write cache
- **-d600** (10 minutes)
- **-r** (random I/O)
- **-w20** (20 percent writes, 80 percent reads)
- **-t8** (eight threads)
- **-o8** (eight outstanding I/O requests)
- **-b8K** (block size is 8 KB)
- **-h** (disable both software caching and hardware write caching)
- **-L** (measure latency statistics)
- **H:\testfile.dat** (file path and name to create for test)

This configuration might be suitable for testing a storage device that is intended to be used for data files on an OLTP system that has a high ratio of reads to writes.

Diskspd Example 2

```
DiskSpd.exe -c500G –d600 -r -w20 -t8 -o8 -b8K -h -L H:\testfile.dat
```

The output of a Diskspd test includes:

- CPU activity during the test, for each CPU.
- Total I/O, read I/O, and write I/O statistics for each thread.
- Total speed, read speed, and write speed by percentile.

Demonstration: Benchmarking I/O

In this demonstration, you will see how to benchmark I/O by using Performance Monitor.

Demonstration Steps

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Navigate to the folder D:\Demofiles\Mod02 in Windows Explorer, and then run **Setup.cmd** as Administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. Double-click the **demo.PerfmonCfg** file. Performance Monitor will start with several counters included, but hidden on the histogram.
5. Return to D:\Demofiles\Mod02 in Windows Explorer, right-click **start_load_1.ps1**, and then click **Run with PowerShell**. If a message is displayed asking you to confirm a change in execution policy, type **Y**. This script starts a workload that ramps up over 60 seconds, and runs for five minutes.
6. While the load is running, return to Performance Monitor and display the performance counters on the histogram by selecting the box next to their names in the list at the bottom of the window. Values relate to benchmark measures as follows:
    - IOPS:
        - Physical Disk: Avg. Disk Bytes/Read
        - Physical Disk: Avg. Disk Bytes/Write
    - Throughput:
        - Physical Disk: Disk Read Bytes/Sec
        - Physical Disk: Disk Write Bytes/Sec
    - Latency factor:
        - Physical Disk: Avg. Disk sec/Read
        - Physical Disk: Avg. Disk sec/Write
7. Close Performance Monitor, Windows Explorer, and PowerShell.

Check Your Knowledge

Question	
Which Diskspd parameter controls the percentage of reads and writes?	
Select the correct answer.	
	-d
	-c
	-r
	-t
	-w

Lab: Testing Storage Performance

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. The owners of the company have decided to start a new direct marketing arm of the company. It has been created as a new company named Proseware Inc. Even though Proseware Inc. has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that can support both the existing workload and the workload from the new company.

Objectives

At the end of this lab, you will be able to configure and run Diskspd to test I/O subsystem performance.

Estimated Time: 30 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Configuring and Executing Diskspd

Scenario

You have reviewed wait statistics for the AdventureWorks database and noticed high wait statistics for I/O, among others. You want to make sure that I/O has been set up correctly and is performing optimally. In this exercise, you will use the Diskspd utility to test storage performance.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Execute Diskspd

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.

2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

▶ Task 2: Execute Diskspd

1. On 10987C-MIA-SQL, you will find a copy of Diskspd.exe in D:\Labfiles\Lab02\Diskspd-v2.0.15\amd64fre.

2. Using this copy of the tool, run a test with the following parameters:

   - Duration: 3 minutes
   - Test file name: D:\Labfiles\Lab02\test.dat
   - Test file size: 2 GB
   - Thread count: 4
   - Percentage of writes: 40%
   - Block size: 64 KB
   - Outstanding I/O requests: 32
   - Read/write method: Random
   - Include Latency Statistics: Yes

Note: You will need to execute the tool as an administrator.

3. Review the output of the test.
4. Delete the test file (**D:\Labfiles\Lab02\test.dat**) when the test is complete.
5. Close Windows PowerShell®.

Results: At the end of this exercise, you will have configured and run Diskspd to test I/O subsystem performance.

Question: What do the results from Diskspd show?

Real-world Issues and Scenarios

In general, you should not run Diskspd on servers that are running production workloads. This is because the additional load that Diskspd places on the I/O subsystem is likely to negatively affect the performance of the production workload. If you need to run Diskspd on production servers, plan to do so during a maintenance window or at a time when the server is out of load.

Module Review and Takeaways

Best Practice: In this module, you have learned how to discuss and analyze storage systems in the context of SQL Server installations. You have gained an understanding of various storage system types, and the ways in which they may be used, and learned about tools and techniques to assess storage system performance.

Categorize Activity

Place each Performance Monitor performance counter into the category of the performance measure to which it relates. Indicate your answer by writing the category number to the right of each item.

Items	
1	Physical Disk: Disk Reads/Sec
2	Physical Disk: Disk Read Bytes/Sec
3	Physical Disk: Avg. Disk Sec/Read
4	Physical Disk: Disk Writes/Sec
5	Physical Disk: Disk Write Bytes/Sec
6	Physical Disk: Avg. Disk Sec/Write
7	Logical Disk: Disk Reads/Sec
8	Logical Disk: Disk Read Bytes/Sec
9	Logical Disk: Avg. Disk Sec/Read
10	Logical Disk: Disk Writes/Sec
11	Logical Disk: Disk Write Bytes/Sec
12	Logical Disk: Avg. Disk Sec/Write

Category 1		Category 2		Category 3
IOPS		Throughput		Latency Factor

Module 3

Database Structures

Contents:

Module Overview

This module covers database structures, data files, and **tempdb** internals. It focuses on the architectural concepts and best practices related to data files, for user databases and **tempdb**, which enable you to configure SQL Server® for maximum performance.

Objectives

After completing this module, you will be able to:

- Describe database structures.
- Understand data file internals and best practices.
- Describe **tempdb** internals and best practices.

Lesson 1
Database Structure Internals

This lesson focuses on physical database structure. You will learn about different components of a database. A thorough understanding of database structure internals helps in designing high performance database solutions.

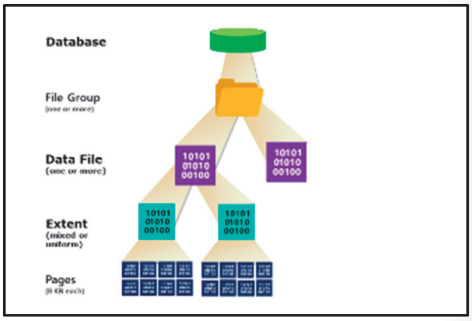
Lesson Objectives

After completing this lesson, you will be able to:

- Understand database components.
- Configure database files and file groups.
- Describe database extents.
- Analyze page structures and page types.
- Explain record structures and record types.
- Understand allocation bitmaps and special pages.
- Analyze page allocation and allocation units.

Database Components

A database is a collection of data, logically stored in tables and views. The data is physically stored in one or more data files and the changes made to the data, or "transactions", are recorded in one or more transaction log files. A SQL Server database consists of the following components:



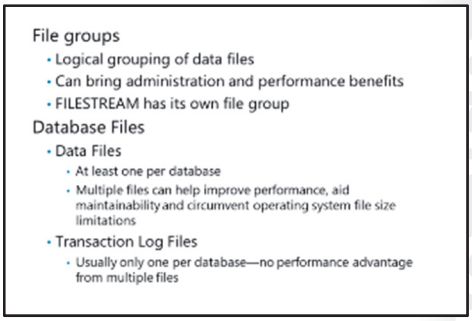
- Database data files
- Transaction log files
- File groups
- Extents
- Pages

A logical file group contains database data files—a database data file is made up of extents, and an extent comprises eight pages, each of which is 8 KB in size. A page is the basic unit of storage used by SQL Server.

Transaction log files store details of all transactions that are carried out against the database.

File Groups and Files

SQL Server stores database data in a collection of operating system files, organized into logical file groups to simplify maintenance and administration.



File Groups

File groups logically group one or more data files. Every database has a primary file group and can also have additional user-defined file groups. The primary file group is the default file group that SQL Server uses, unless another file group has been set as default. The primary file group always contains the first data file. The primary file group can also contain one or more additional data files.

When you create database objects without specifying a file group, they are assigned to the default file group. Systems objects are always assigned to the primary file group, even if it is not the default file group.

Some benefits of file groups are as follows:

- Administrative
  - File group backups help to reduce backup time and speed up the database recovery.
  - Data and nonclustered indexes can be separated across different disks for more efficient index maintenance.
- Performance
  - Two data files can be created on separate disks grouped under one file group. A table can then be assigned to this file group. The queries to the table will be spread across the two disk spindles, improving query performance.
  - A table can be partitioned across multiple file groups. Each file group can have one or more data files on separate disks. By spreading the partitions across multiple disks, maintainability and performance is improved.

When creating a database, you can create user file groups by adding a FILEGROUP clause to the CREATE DATABASE statement. Appending DEFAULT to the FILEGROUP clause sets a user file group as the default file group for the database.

The code example creates a database called “test”, with an additional file group named “user”.

CREATE DATABASE with FILEGROUP clause

```
CREATE DATABASE [test]
 ON  PRIMARY
( NAME = N'test_data1', FILENAME = N'E:\mssql\data\test_data1.mdf' , SIZE = 5120KB ,
FILEGROWTH = 1024KB ),
 FILEGROUP [user] DEFAULT
( NAME = N'test_data2', FILENAME = N'E:\mssql\data\test_data2.ndf' , SIZE = 5120KB ,
FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'test_log', FILENAME = N'E:\mssql\log\test_log.ldf' , SIZE = 2048KB ,
FILEGROWTH = 512KB)
GO
```

The FILESTREAM feature, which enables SQL Server to store unstructured data directly in the file system, uses a special FILESTREAM file group to contain a pointer to the file system folder where data is stored.

Database Files

A database has two types of files—data files and transaction log files.

Data Files

Every database, whether system or user, has at least one data file. The first or primary data file has database startup information and details about the other files used by the database. The recommended file extension for the first data file is **mdf** and for subsequent data files it is **ndf**.

Creating more than one data file for a database has a number of advantages:

- It allows data to be spread across multiple disks to improve I/O performance.
- Separate table and nonclustered indexes can increase performance and maintainability.
- Partial availability and piecemeal restore.
- If a database exceeds maximum file size limit, an additional file can be added, so that the database can continue to grow without failure.

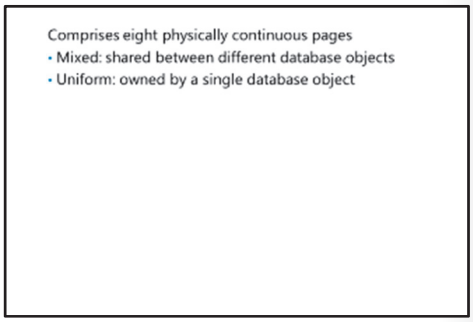
Transaction Log Files

The transaction log file records the details of every transaction that occurs against the database and is primarily used for database recovery. There is usually only one log file for each database. The recommended and default file extension is **ldf**. Unlike data files, creating multiple transaction log files across different disks will not improve performance because transaction logs are written sequentially.

Extents

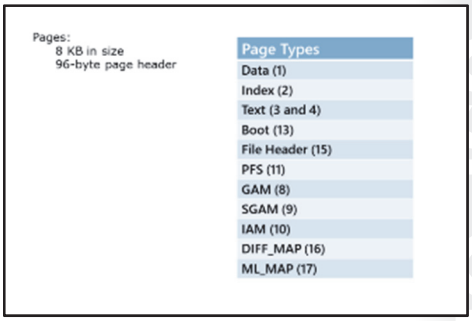
An extent consists of eight physically continuous pages of data, making it 64 KB in size. SQL Server uses two types of extents:

- **Mixed extents**. These are shared extents. Each page in a mixed extent may belong to a different object.
- **Uniform extents**. These are owned by a single object. All eight pages belong to the owning object.



When you create a new object, it is assigned a page from a mixed extent. When the object grows to the point that it uses eight pages, it switches to use a uniform extent for subsequent allocations. If a new index is being created on a table that has enough rows to require more than eight pages, all allocations are made to a uniform extent.

Page Structure and Page Types



A page is the basic unit of storage in SQL Server. Each page is 8 KB in size and is divided into three parts: page header, data storage space, and the row offset array. The page header is 96 bytes in size and stores metadata about the page, including:

- Page ID. Identifies the page within a data file.
- Page type. Identifies the type of the page—for example, data, index, or Global Allocation Map (GAM) page.
- Pointers to previous and next page in a table.
- Number of records in a page.
- Number of ghost records in a page.
- Log sequence number (LSN) of the last log record that modified the page.
- Free space on the page.

The data rows are written to the data storage space serially after the header.

The row offset table starts at the end of the page and contains a two-byte pointer to a row on the page. It contains one entry for each row on the page. The pointer records how far the row is from the start of the page. The row offset array is in a reverse sequence to the rows on the page.

The rows within a page are not always stored in a logical order, whereas the row offset is ordered and modified whenever a row is inserted or deleted from within the page. The row offset maintains the logical order of the rows.

Pages are stored in extents. Read and write operations are all performed at a page level.

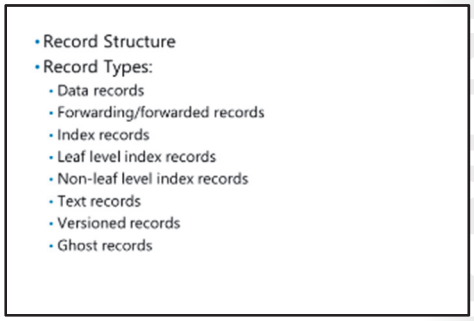
Page Types

The different types of pages in SQL Server and their specific Page ID, with the description, are as follows:

- **Data (1)** – Contains the data rows in a table.
- **Index (2)** – Contains index entries for intermediate levels of clustered index and all levels of nonclustered index.
- **Text pages**
  - **Text mix (3)** – Holds large object (LOB) values less than 8 KB size. It can be shared between LOB values in the same partition of an index or a heap.
  - **Text tree (4)** – Holds LOB values larger than 8 KB.
- **GAM (8)** – Global Allocation Map records whether an extent is allocated or not.
- **SGAM (9)** – Shared Global Allocation Map records whether an extent can be used to allocate mixed pages.
- **IAM (10)** – Index Allocation Map contains information about extents used by a table or index per allocation unit.
- **PFS (11)** – Page Free Space contains information about page allocation and free space available within a PFS interval.

- **Boot (13)** – Contains information about the database. There is only one boot page in a database. Page 9 in file 1 is always a boot page.
- **File header (15)** – Page 0 in every file is the file header page. It contains information about the file.
- **Diff map (16)** – The Differential Change Map page contains information about the extents changed since the last full or differential backup within a GAM interval.
- **ML map (17)** – The Minimally Logged Changed Map contains information about the extents changed in bulk-logged mode since the last backup.

Record Structure and Record Types



A record is the physical storage related to a table or an index row. The record structure is as follows:

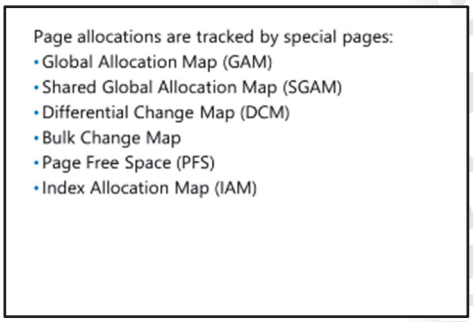
- A four-byte record header: two bytes for record type and two bytes to record the forward pointer to the NULL bitmap.
- A fixed length portion to store fixed length data type columns.
- NULL bitmap that consists of:
  - Two bytes to store the count of columns in the record.
  - Variable number of bytes to store one bit per column in the record for nullable or not nullable columns.
- Variable length column offset array:
  - Two bytes to store the count of variable length columns in the record.
  - Two bytes per variable length column.
- Versioning:
  - 14-byte structure to store timestamp and pointer to the version store in **tempdb**.

The different record types are:

- **Data records**
  - Data records are rows from a heap or leaf level of a clustered index.
  - Stored in data pages.
- **Forwarded/forwarding records**
  - Data records present only in heaps.
  - When a data record is updated so that its size cannot fit in the original page, it is moved to a new page. The record at the new location is the "forwarded" record, and the pointer to it in the old page is the "forwarding" record.
  - Avoids the need for updating nonclustered indexes but it may reduce lookup performance.

- **Index records**
  - Stored in index pages.
  - Come in two types of index record: leaf level index record and non-leaf level index record.
  - Leaf level index records store nonclustered index key columns, a link to the corresponding row in a table, and included columns.
  - Non-leaf level index records occur in all index types in the levels above the leaf level. Non-leaf level index records contain information to assist the storage engine in navigating to the correct point at the leaf level.
- **Text records**
  - Stored in text pages.
  - Text records store "off-row" LOB and all row overflow data.
  - Off-row means that the record stores a pointer to the root of a loose tree structure that holds the LOB data. The pointer is 16/24 bytes and ranges up to 72 bytes. The text tree is different from the index b-tree structure.
- **Ghost records**
  - Records that have been logically deleted but physically exist on a page.
  - Simplifies key-range locking and transaction rollback.
  - A record is marked as deleted by setting a bit. A ghost cleanup process physically deletes the ghost records after the calling transaction commits.
- **Versioned records**
  - Used by features that use the versioning system, such as online index operation and snapshot isolation.
  - Latest version of a record on a page has a 14-byte pointer to the previous version in the version store in **tempdb**.

Allocation Bitmaps and Special Pages



When a new object is created, SQL Server allocates the first eight pages from mixed extents. The subsequent space allocations for the object are then completed with uniform extents. SQL Server uses special pages to track the extent allocation—called allocation bitmaps. The different types of pages which track allocations are as follows:

- **Global Allocation Map**: the GAM pages track whether an extent is allocated or not. A data file is logically divided into a GAM interval of 4 GB, or 64,000 extents. At the start of every GAM interval, there is a GAM extent that contains GAM pages to track the GAM interval. The bits in the GAM bitmap have the following meaning:
  - Bit 1. The extent is available for allocation.
  - Bit 0. The extent is already allocated and is not available for further allocations.

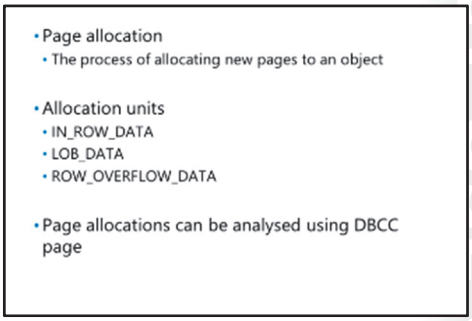
- **Shared Global Allocation Map**: the SGAM is exactly same as the GAM and tracks mixed extents. It is essentially used to track mixed extents with unallocated pages. The bits in the SGAM bitmap have the following meaning:
  - Bit 1. The extent is a mixed extent with a free page available for use.
  - Bit 0. The extent is either uniform or mixed with no free page available for allocation.
- **Differential Change Map**: the DCM pages track the extents modified since the last full backup was taken. The DCM page is structured in the same way as the GAM and covers the same interval. The bits in the DCM page have the following meaning:
  - Bit 1. The extent has been modified since the last full backup.
  - Bit 0. The extent has not been modified.
- **Bulk Change Map**: minimally logged pages track extents modified by the minimally logged operations, such as SELECT INTO and BULK INSERT, since the last transaction log backup in bulk-logged recovery model. The ML bitmaps are similar in structure to GAM bitmaps, but differ in bitmaps semantics.
  - Bit 1. The extent has been modified by the minimally logged operation since the last transaction log backup.
  - Bit 0. The extent has not been modified.
- **Page Free Space**: PFS tracks free space in pages. The data file is logically divided into a PFS interval of 64 MB or 8,088 pages. The PFS page has a byte map instead of bitmap, with one byte for each page in the PFS interval (excluding itself). The bits in each byte have the following meaning:
  - Bits 0-2. The amount of free space in a page.
  - Bit 3. This records the presence of a ghost record in a page.
  - Bit 4. This indicates whether a page is an Index Allocation Map (IAM) page.
  - Bit 5. This indicates whether a page is a mixed page.
  - Bit 6. This indicates whether a page is allocated or not.
  - Bit 7. This is unused.
- **Index Allocation Map**: Index Allocation Map (IAM) pages track all extent allocations for a table/index/partition in a GAM interval of a data file. An IAM page only tracks the space for a single GAM interval in a single file. If a database has multiple data files or a data file larger than 4 GB then multiple IAM pages are required. An IAM page has two records, an IAM header, and the bitmap. The IAM header has the following information:
  - The position of the IAM page in the IAM chain.
  - The GAM interval that the page tracks.
  - The pages allocated from the mixed extents. This information is only used in the first IAM page in an IAM chain.

An IAM page has the same structure as that of a GAM, but the bitmap has different semantics:

- Bit 1. The extent is allocated to the IAM chain or allocation unit.
- Bit 0. The extent is not allocated.

An IAM page from one file can track the allocations of the other file. IAM pages are single page allocations associated with mixed extents. An IAM page isn't tracked anywhere.

Page Allocation and Allocation Units



Page allocation is the process of allocating new pages to an object. A page is either allocated from a mixed extent or from a uniform extent. A brief summary of the steps involved in allocating the first page is as follows:

1. **Find an extent to allocate**: SQL Server looks for an available mixed extent to allocate. If a mixed extent is not available, a new mixed extent is allocated.
2. **Allocate the data page**: the data page is marked as allocated and mixed in the PFS; the extent is marked as available in the SGAM.
3. **Allocate the IAM page**: the page is marked as allocated, mixed, and an IAM page in the PFS. The corresponding extent is marked as available in the SGAM page.
4. **Modify IAM page**: the allocated data page ID is recorded in the IAM single page slot array.
5. **Modify table metadata**: the allocated IAM page ID and the data page ID are recorded in the table metadata.

Allocation Units

An allocation unit is a group of pages within a heap or a b-tree (clustered index) used to manage data based on different page types. There are three types of allocation units in SQL Server:

- **IN_ROW_DATA** is a collection of data/index pages containing all data except LOB data. Every table, view, or indexed view partition has one IN_ROW_DATA allocation unit. This also contains additional pages for each nonclustered and xml index defined on a table or a view. The **sys.allocation_units** dynamic management view (DMV) contains a row for each allocation unit in a database. The internal name of this allocation unit is HoBt—heap or b-tree.
- **ROW_OVERFLOW_DATA** is a collection of text/image pages containing LOB data. Every table, view, or indexed view partition has one ROW_OVERFLOW_DATA allocation unit. The pages to this allocation unit are only allotted when a data row with variable length column (**varchar**, **nvarchar**, **varbinary**, **sql_variant**) in an IN_ROW_DATA allocation unit exceeds the 8 KB row size limit. When the size limit is reached, the column on the data page is moved to a page in ROW_OVERFLOW_DATA and the original page is updated with a 24-bit pointer to the new page. The internal name of this allocation unit is small LOB.
- **LOB_DATA** is a collection of text/image pages containing LOB data of the data types **text**, **ntext**, **image**, **xml**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, or CLR user-defined types. Internally, this is called a LOB.

Analyzing Page Allocations

You can analyze page allocations using the **sys.dm_db_database_page_allocations** DMV and the **DBCC page** command. You can use the **sys.dm_db_database_page_allocations** DMV to obtain page information for a specific database and/or table. With the **DBCC page** command, you can view the contents of a page.

- **sys.dm_db_database_page_allocations** takes five mandatory parameters: database id, table id, index id, partition id, and mode. Table id, index id and partition id can be passed the value NULL to return information on all tables, indexes, or partitions. The mode parameter accepts the values detailed or limited. Limited returns only page metadata, detailed returns metadata, and additional information such as inter-page relationship chains and page types.

- **DBCC page** takes four mandatory parameters: database id or name, file number, page number, and output option. Output option can be any of the following four integers, which give different levels of detail:
  - 0 – Prints the page header only.
  - 1 – Prints the page header, per row hex and page slot array dump.
  - 2 – Prints the page header and the complete page hex dump.
  - 3 – Prints the page header and detailed row information.

The output of **DBCC page** is sent to the SQL Server error log, but can be viewed in SQL Server Management Studio by turning on the trace flag 3604.

Demonstration: Analyzing Database Structures

In this demonstration, you will see how to:

- Analyze SQL Server page structures.
- Analyze record structures.

Demonstration Steps

1. Start the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Demofiles\Mod03** folder, run **Setup.cmd** as Administrator.
3. In the **User Account Control** dialog box, click **Yes**, wait for the script to finish, and then press Enter.
4. Open SQL Server Management Studio, connect to **MIA-SQL** database engine with Windows Authentication.
5. In SQL Server Management Studio, open the **PageStructure.sql** Transact-SQL file in the **D:\Demofiles\Mod03** folder.
6. Highlight the Transact-SQL code under the comment **Create table and insert test data**, and click **Execute** to create a test table and populate it with data.
7. Highlight the Transact-SQL code under the comment **Find Page information**, and click **Execute**, and from the query results note the value in the column **allocated_page_page_id**.
8. Highlight the Transact-SQL code under the comment **Enable trace flag**, and click **Execute** to turn on trace flag 3604.
9. Modify the Transact-SQL code under the comment **View page allocation**, replacing **XXX** with the **allocated_page_page_id** value from step 7.
10. Highlight the Transact-SQL code under the comment **View page allocation**, and click **Execute**, and examine the page information, noting that slot 0, slot 1, and slot 2 are of the type PRIMARY_RECORD because they contain row data.
11. Highlight the Transact-SQL code under the comment **Update data**, and click **Execute** to update the data stored on the page.
12. Highlight the Transact-SQL code under the comment **Find Page information**, and click **Execute**, noting that an extra row is returned because data for the table now consumes more than one page.

13. Modify the Transact-SQL code under the comment **View updated page allocation**, replacing **XXX** with the **allocated_page_page_id** value from step 7.
14. Highlight the Transact-SQL code under the comment **View updated page allocation**, and click **Execute**. Examine the page information, noting that slot 0 is of the type FORWARDING_STUB because some row data is stored in a separate page.
15. Close SQL Server Management Studio without saving changes.

Check Your Knowledge

Question	
Which of the following page types track changes to pages since the last full backup?	
Select the correct answer.	
	Global Allocation Map
	Index Allocation Map
	Differential Change Map
	Shared Global Allocation Map
	Page Free Space

Lesson 2
Data File Internals

This lesson focuses on data file internals. You will learn about configuring storage for data files, auto grow and auto shrink features, how it affects performance and best practices related to them. You will also learn about allocation of data within data files.

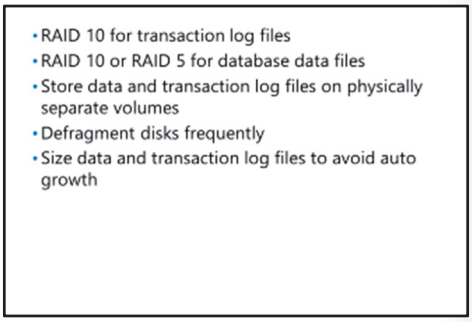
Lesson Objectives

After completing this lesson, you will be able to:

- Understand volume configuration best practices.
- Understand best practices for a number of data files and data file placement.
- Describe physical versus logical I/O.
- Understand allocations within data files.
- Implement instant file initialization.
- Describe auto grow and auto shrink.
- Understand data file shrinking.
- Monitor database files.

Volume Configuration Best Practices

A database consists of two types of files: data files for storing data, and transaction log files for recording database transactions. The configuration of the volumes on which these files are stored affects overall database performance, so consideration must be given to the configuration of these volumes.



When using RAID sets, RAID level 10 is the preferred option for both data and transaction log files. To make cost savings over a complete RAID 10 solution, you can place data files on RAID 5, leaving log files on RAID 10.

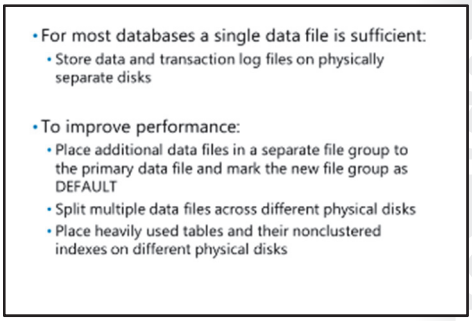
Because the performance of a database is directly proportional to the speed at which SQL Server can write to the transaction log files, RAID 5 should not be used for transaction log files—RAID 5 writes are slower, due to the calculation of parity bits.

Always store database data files and transaction log files on different physical disks and, where possible, have the disks attached to different controllers. The aim is to isolate the I/O of transaction log files from the I/O of data files as much as possible.

NTFS volumes become fragmented over time and this fragmentation degrades performance. To maintain optimal performance, it is important to regularly defragment volumes. Volumes should only be defragmented when the SQL Server service is not running. You can minimize disk fragmentation by having dedicated disks for SQL Server data files and pre-allocating data file sizes to avoid auto growth.

Number of Data Files and Placement

The majority of user databases will perform adequately with a correctly sized single data file; however, you can improve performance by using additional data files. If you choose to use more than one data file, create an additional file group for the data files and mark the new file group as default. This will result in the primary data file only storing system objects, which reduces the possibility of database corruption and makes recovery simpler.

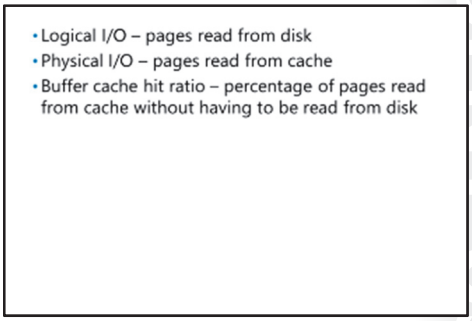


When using multiple data files, they should be split across different physical disks and always stored on physically separate disks to the transaction log files and **tempdb**.

You can place heavily used tables and their nonclustered indexes on different disks to improve performance—consider doing so with tables that appear at either side of a join query. The increase in performance is due to parallel I/O and the improvement can be very worthwhile.

Physical I/O vs. Logical I/O

Logical I/O in SQL Server represents the pages that are read from cache to answer a query.



Physical I/O in SQL Server represents pages that are read from disk to answer a query. To answer a query, SQL Server fetches any pages from disk that are not currently stored in the cache, and places them in the cache. The query is then answered from the cache.

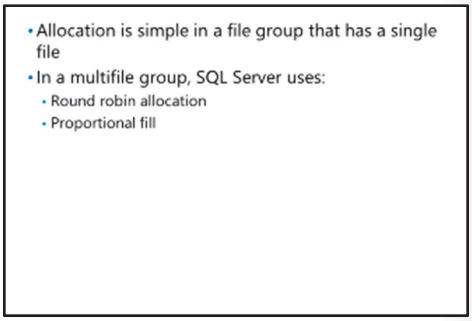
Ultimately, all queries are answered entirely from cache and therefore, when performance tuning, logical reads is a good performance indicator, because it represents the total number of pages SQL Server has to read to answer the query. Logical reads can normally be reduced by effective indexing or rewriting of the query.

Buffer cache hit ratio is an important metric for measuring SQL Server performance. It is calculated by the formula ((logical reads–physical reads)/logical reads) * 100. A SQL Server instance that is performing well will have a buffer cache hit ratio close to 100. The buffer cache hit ratio can be improved by making more memory available to SQL Server.

You can view physical and logical I/O stats using the DMV **sys.dm_exec_query_stats** or by using the Transact-SQL command SET STATISTICS IO ON before issuing a query. SET STATISTICS IO ON will only return statistics for queries in the current session.

Allocations Within Data Files

If a file group contains only a single data file, then extent allocation is straightforward. For file groups with more than one data file, SQL Server uses a round robin proportional fill algorithm to allocate extents. The round robin algorithm means allocations are made in each data file, proportional to the amount of free space in each file. For example, if in a file group, file A has 80 MB free and file B has 20 MB free, and SQL Server needs to write 10 MB, 8 MB will be written to file A and 2 MB to file B, in an attempt to ensure both files will reach fill-up at the same time.



If auto growth is enabled, files are automatically expanded one at a time, again in a round robin way. For example, if the space in file f1 and file f2 is exhausted, file f1 is expanded first. Then, when file f1 is exhausted again, file f2 is expanded.

Data is striped evenly (proportional to available space in each file) across all the files until the free space in all files has been exhausted. If enabled, auto grow then starts and allocations are made to one file at a time. If the rate of auto growth is large, the allocations are made to one file until it fills up completely, resulting in uneven stripping/allocation and ultimate performance degradation.

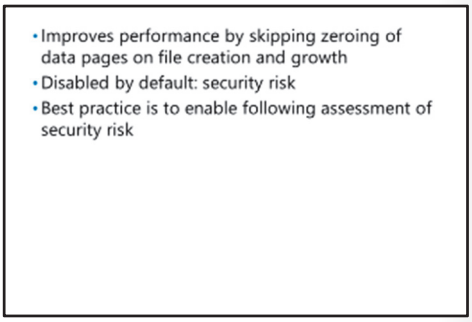
Best practices to even out allocation within file groups include:

- Spreading the files within file groups across different disks.
- Initializing all files to be of the same size, to even out the data distribution among files.
- Enabling auto growth and setting it equal for all the files within a file group, so files auto grow equally, making for a more even distribution of data.

Instant File Initialization

Whenever space is allocated to a data or log file, SQL Server overwrites the existing data on disk by filling the allocated space with zeros. This zeroing out of allocated spaces occurs under the following operations:

- Database creation.
- Adding new data or log file to existing database.
- Increasing the size of existing database files, including auto grow operations.
- Restoring a database or file group.



The zeroing is a time-consuming process that causes a delay in space allocation, resulting in reduced performance.

Instant file initialization is a feature of SQL Server that allows file allocation requests to skip the zeroing process on initialization. Skipping the zeroing process speeds up the data file space allocation by calling

the **SetFileValidData** windows function. The **SetFileValidData** function skips zeroing when writing non-sequentially to a file and marks the data in the file as valid. Data files are therefore initialized instantly, resulting in increased performance and reduced I/O. The **SetFileValidData** function only benefits data file creation; it has no impact on the performance of transaction log file creation.

Instant file initialization is disabled by default. SQL Server utilizes this feature if the SQL Server service account has been granted the Windows permission SE_MANAGE_VOLUME_NAME. You can grant this permission by adding the appropriate account to the Perform Volume Maintenance Tasks security policy.

SQL Server instant file initialization will not work when Transparent Data Encryption is enabled.

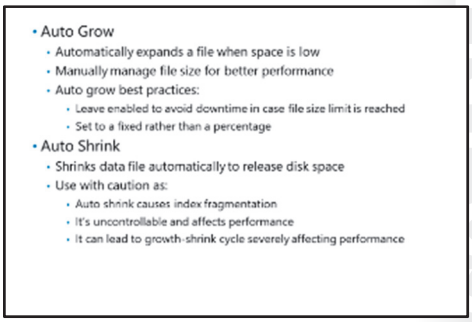
The performance gain from instant file initialization carries with it an inherent security risk. When the SQL Server service account is granted the Perform Volume Maintenance Tasks permission, it may be able to read the encrypted contents of a recently deleted file by using the undocumented DBCC PAGE command.

You can use the trace flag 3004 to verify whether the instant file initialization feature is running; do this by enabling trace flags 3004 and 3605, auto growing a data file, and then examining the SQL Server error log. If instant file initialization is enabled, the error log will contain messages stating the data file has been instantly initialized.

You can disable instant file initialization by removing the SQL Server service account from the Perform Volume Maintenance Tasks security policy and restarting the service. Best practice is to assess the security risk and enable instant file initialization if the security risk is acceptable to your organization.

Auto Grow and Auto Shrink

Auto grow and auto shrink are per file database options in SQL Server that can assist with the management of data file and transaction log file sizes.



Auto Grow

The auto grow file property automatically increases the file size by a specified limit when free space in the file becomes low. File growth can be specified as a percentage of current file size or a fixed amount. The percentage based file growth means that the auto growth gets bigger as the file size increases. This can result in performance issues if instant file initialization is disabled. Because instant file initialization is not supported for transaction log files, it is recommended that auto growth is set to a fixed amount for log files.

Some best practices related to auto growth are as follows:

- Auto growth should always be used as a contingency plan for unexpected growth. It should not be used to manage file growth on a day-to-day basis.
- Use alerts to monitor file size and increase file size proactively to avoid fragmentation.
- The auto growth size should be large enough to avoid performance issues resulting from file initialization. The exact value to auto growth value depends on many factors; however, a general rule of thumb is to set auto growth to one eighth of the file size.
- Keeping the transaction sizes small will prevent unplanned file growth.

You can use the **sys.database_files** system view to return the max file size and auto growth values. Configure the auto growth option by using any one of the following methods:

- ALTER DATABASE Transact-SQL statement.
- SQL Server Management Studio.
- **sp_dboption** stored procedure.

Best Practice: You should consider creating data and log files with sufficient space for the expected volume of data in the database. This will avoid the frequent auto growth of files.

Auto Shrink

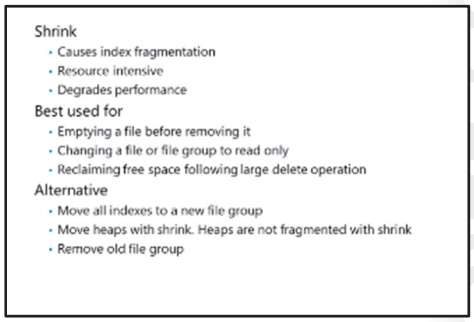
Auto shrink is disabled by default and helps to control excessive file growth by automatically shrinking database files; however, there are a number of disadvantages, including:

- Shrinking files can result in index fragmentation.
- The shrink operation is resource intensive.
- The shrink operation shrinks a database every 30 minutes and you cannot control the start time.
- A database requires free space for normal functioning. The database will grow as required, depending on the auto growth setting. This may result in a shrink-grow cycle causing severe performance issues.

The auto shrink option is set to ON if the **is_auto_shrink** column in the **sys.databases** system view contains a value of **1**. It is set to OFF if the value of **is_auto_shrink** is **0**. Configure the auto shrink option by using any of the following methods:

- ALTER DATABASE Transact-SQL statement.
- SQL Server Management Studio.
- **sp_dboption** stored procedure.

Data File Shrinking



The data file shrink operation manually reduces data file size and releases space back to the operating system. Perform data file shrink operations in any of the following ways:

- DBCC SHRINKFILE command.
- SQL Server Management Studio.

There are three shrink actions possible on a data file:

- **Release unused space** – equivalent to running DBCC SHRINKFILE with TRUNCATE_ONLY option. This releases the unused space in a data file to the operating system. This option does not result in index fragmentation.
- **Shrink file to a specified size** – reorganizes the data pages before releasing the unused space. The shrink algorithm picks up allocated pages at the end of the file and moves them to the front of the data file. This shuffling of index pages in a clustered/nonclustered index results in index fragmentation.

- **Empty file** – equivalent to running DBCC SHRINKFILE statement with EMPTY_FILE option. This empties a file by moving all data from the data file to the other files within the same file group. Normally used when deleting a data file, because data files can only be removed when empty.

Shrink is a resource intensive process. It moves a lot of data through buffer pool, which can force hot pages out of the buffer, resulting in performance issues. In an already busy I/O subsystem, shrink can result in a long disk queue length and I/O timeouts.

Shrinking is not recommended—an alternative solution to release space to the operating system is:

1. Create a new file group and add a new data file with the size initially set to X.
2. Move indexes to the new file group using CREATE INDEX... WITH DROP_EXISTING.
3. Move heaps with shrink command. Heaps are not fragmented with shrink operation.
4. Drop the original file group.

Demonstration: Shrinking Databases

In this demonstration, you will see:

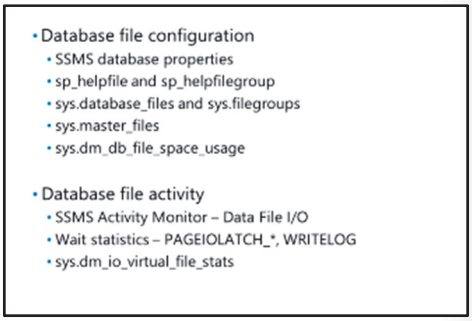
- How to check free space in database files.
- How to shrink a data file.
- The impact of shrinking data files on index fragmentation.

Demonstration Steps

1. Open SQL Server Management Studio, connect to **MIA-SQL** database engine with Windows Authentication.
2. In SQL Server Management Studio, open the **shrinkdb.sql** Transact-SQL file in the **D:\Demofiles\Mod03** folder.
3. Highlight the Transact-SQL code under the comment **Check database free space**, and click **Execute**, note that the database has more than 350 MB of unallocated space.
4. Highlight the Transact-SQL code under the comment **Check index fragmentation**, and click **Execute**. Note the values in the **avg_fragmentation_in_percent** column.
5. Highlight the Transact-SQL code under the comment **Shrink database data file**, and click **Execute**.
6. Highlight the Transact-SQL code under the comment **Check database free space**, and click **Execute**. Note that the unallocated space has reduced significantly.
7. Highlight the Transact-SQL code under the comment **Check index fragmentation**, and click **Execute**. Note that some values in the **avg_fragmentation_in_percent** column are much higher than they were before shrinking the database file.
8. Close SQL Server Management Studio without saving changes.

Monitoring Database Files

There are several different methods to gather information about SQL Server database files and file activity.



Database File Configuration

You can view details of database file configuration using the following methods:

- **SQL Server Management Studio**. Information about database files and file groups is available on the **Files** and **Filegroups** pages of the Database Properties window, accessed through SQL Server Management Studio (SSMS) Object Explorer.
- **sp_helpfile**. The **sp_helpfile** system stored procedure returns basic information about database files for the current database.
- **sp_helpfilegroup**. The **sp_helpfilegroup** system stored procedure returns basic information about database file groups for the current database.
- **sys.database_files**. The **sys.database_files** system view returns detailed information about database files for the current database.
- **sys.filegroups**. The **sys.filegroups** system view returns detailed information about database file groups for the current database.
- **sys.master_files**. The **sys.master_files** system view returns detailed information about the database files for all databases in the current instance of SQL Server.
- **sys.dm_db_file_space_usage**. The **sys.dm_db_file_space_usage** system DMV returns detailed information about occupied and free space in the data files of the current database.

For more information about the system views that provide information about database files, see MSDN:

Databases and Files Catalog Views (Transact-SQL)

http://aka.ms/e8xxy1

For more information about **sys.dm_db_file_space_usage**, see Microsoft Docs:

sys.dm_db_file_space_usage (Transact-SQL)

http://aka.ms/kc2jhz

Database File Activity

Database file activity can be monitored from within SQL Server.

- **SSMS**. A GUI view of current file I/O activity by database file is available in the **Data File I/O** section of SSMS Activity Monitor. Information about FILESTREAM I/O is not available in this view.
- **Wait statistics**. Several wait statistics types measure when worker processes are waiting for file system activity, including PAGEIOLATCH_* and WRITELOG. These wait statistics can indicate if I/O performance is affecting the responsiveness of SQL Server.

- **sys.dm_io_virtual_file_stats**. The **sys.dm_io_virtual_file_stats** system dynamic management function (DMF) provides information about data file I/O activity since the SQL Server instance was most recently started.

For more information about **sys.dm_io_virtual_file_stats**, see MSDN:

sys.dm_io_virtual_file_stats (Transact-SQL)

http://aka.ms/o22st7

Question: When configuring auto grow for SQL Server data files, should you specify a percentage growth or a fixed size growth—and why?

Lesson 3
tempdb Internals

This lesson focuses on **tempdb** internals. **tempdb** is a system database used by SQL Server to hold temporary tables, internal objects and versioning details. Smooth functioning of **tempdb** is essential for overall SQL Server performance.

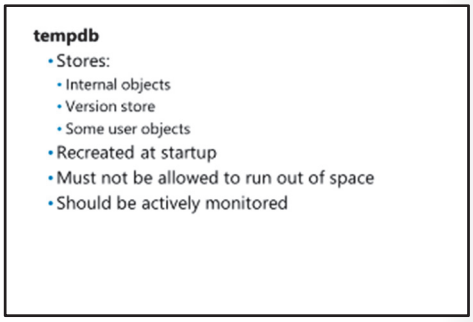
Lesson Objectives

After completing this lesson, you will be able to:

- Describe **tempdb** usage.
- Understand version store internals.
- Configure **tempdb** for optimal performance.

tempdb Usage

tempdb is a system database used by SQL Server to store internal objects, version store, and certain user objects.



- Internal objects are created by SQL Server so it can process Transact-SQL statements. They are created or dropped within a scope of statement. Internal objects can be one of the following:
  - Intermediate sort results from GROUP BY, ORDER BY, UNION queries, or from index rebuild with SORT_IN_TEMPDB set to ON.
  - Work tables for cursor, spool operations, and LOB storage.
  - Work files for hash joins and hash aggregate operations.
- Version store. **tempdb** holds versions created during the following operations:
  - Online index operations.
  - Transactions with Read Committed Snapshot and Snapshot isolation level.
  - Triggers.
- User objects are explicitly created by a user. Some examples of user objects that might be stored in **tempdb** are:
  - Global and local temporary tables.
  - Table variables, when spilled to disk.
  - Temporary indexes.
  - Tables from table-valued functions.

There is only one **tempdb** system database on each SQL Server instance and it is recreated whenever the SQL Server instance is restarted. Creating user objects, such as tables, in **tempdb** is not recommended as they are lost in the event of a server restart. You cannot back up or restore **tempdb**; however, you can roll back **tempdb** transactions because they are minimally logged.

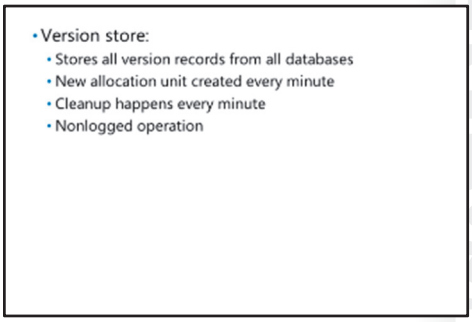
On most SQL Server instances, **tempdb** is a very active database storing a lot of data. Should **tempdb** run out of space, errors will occur and the SQL Server service can become unstable. It is recommended that you monitor **tempdb** to ensure good system performance.

You can monitor the disk space consumed by using the **sys.dm_db_file_space_usage** DMV. You can also use the **sys.dm_db_session_space_usage** DMV and **sys.dm_db_task_space_usage** DMV to find objects using a lot of space in **tempdb**. The following performance monitor counters can be used to track **tempdb** space usage:

- Database: Log file(s) Size (KB).
- Database: Log File(s) Used (KB).
- Free Space in Tempdb (KB).
- Version Store Size (KB).
- Version Generation Rate (KB/s).
- Version Cleanup Rate.

Version Store Internals

A version store consists of data pages that hold the data rows required to support features within SQL Server that use row versioning. There are two version stores within SQL Server; a common version store and an online index build version store. The version stores contain the following:



- Row versions from Read Committed Snapshot and Snapshot isolation level.
- Row versions from online index operations, multiple active result set (MARS), and AFTER triggers.

Row versioning based isolation levels eliminate the need for shared locks on read operations, so reducing the number of locks acquired by a transaction. This results in increased system performance, primarily due to reducing blocking and deadlocks, and the resources needed to manage them.

Row versioning increases the **tempdb** usage. Enabling a row versioning based isolation level causes data modifications to be versioned. One new allocation unit is created every minute and a cleanup process runs every minute to remove the unwanted versions from the version store. Version store operations are the only nonlogged operations in SQL Server. A copy of the data before data modification is stored in **tempdb** even when there are no active transactions using a row versioning based isolation level. The modified data stores a pointer to the versioned data stored in **tempdb**.

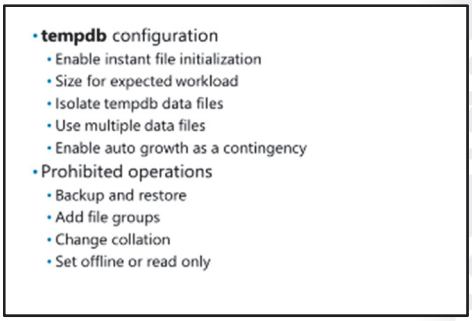
Use the following DMVs to monitor version store:

- **sys.dm_tran_top_version_generators**. Returns a virtual table for the objects producing the most versions in the version store.
- **sys.dm_tran_active_snapshot_database_transactions**. Returns a virtual table for all active transactions in all databases within the SQL Server instance that use row versioning.
- **sys.dm_tran_transactions_snapshot**. Returns a virtual table that displays snapshots taken by each transaction.
- **sys.dm_tran_current_transaction**. Returns a single row that displays row versioning related state information of the transaction in the current session.
- **sys.dm_tran_current_snapshot**. Returns a virtual table that displays all active transactions at the time the current snapshot isolation transaction starts.

Additionally, you can use the following performance counters to monitor version store in **tempdb**:

- **Version store size (KB)**. This monitors the size in KB of all version stores. This information is helpful to calculate an estimate of additional space needed for **tempdb**.
- **Version generation rate (KB/s)**. This returns the version generation rate in KB per second in all version stores.
- **Version cleanup rate (KB/s)**. This returns the version cleanup rate in KB per second in all version stores.
- **Version store unit count**. This returns the count of version store units.
- **Version store unit creation**. This returns the total number of version store units created to store row versions since the instance was started.
- **Version store unit truncation**. This returns the total number of version store units truncated since the instance was started.
- **Update conflict ratio**. This returns the ratio of update snapshot transactions that have update conflicts to the total number of update snapshot transactions.
- **Longest transaction running time**. This returns the longest running time in seconds of any transaction using row versioning. This can be used to determine long running transactions.
- **Transactions**. This returns the total number of active transactions, excluding system transactions.
- **Snapshot transactions**. This returns the total number of active snapshot transactions.
- **Update snapshot transactions**. This returns the total number of active snapshot transactions that perform update operations.
- **Non-snapshot version transactions**. This returns the total number of active non-snapshot transactions that generate version records.

tempdb Configuration



tempdb needs to be managed effectively to avoid performance bottlenecks. During installation, **tempdb** is created with as many data files as there are CPU cores or, if there are more than eight CPU cores, eight data files. Some recommendations for an optimal **tempdb** configuration are as follows:

- **Instant file initialization**. Enable instant file initialization for the benefit of data file space allocations. This will speed up space allocations in **tempdb** data files, increasing query performance.
- **Sizing tempdb.** To avoid file growth as much as possible, **tempdb** should be sized depending on the application workload. It can be difficult to assess the workload. Running a general production workload, including index rebuild operations whilst monitoring **tempdb** size, should give a good baseline figure for **tempdb** sizing.
- **Isolate tempdb storage**. Isolate **tempdb** storage I/O as much as possible. Using fast disks, separating controllers, and appropriate RAID levels, can significantly improve **tempdb** performance. Using SSD disks can offer performance far above that of conventional disks.
- **Multiple tempdb data files**. During setup, **tempdb** is created with multiple data files. It is not recommended that this be changed, except for a non-production low-use system. Multiple files help to reduce space allocation contention in **tempdb**. Distributing multiple data files across different disks can offer performance benefits.
- **Auto growth**. Enable auto growth in **tempdb** as a contingency plan to avoid issues resulting from unexpected growth. If **tempdb** has multiple data files, set the auto growth for each file to the same fixed value to maintain even distribution of data when a file auto grows.

Note: At the time of installing SQL Server, you can configure tempdb to have multiple data files of up to 256GB each.

There are a number of operations which cannot be performed on the **tempdb** database. These include:

- Backup and restore – **tempdb** is recreated at server startup. It is not possible to back up or restore it.
- Add file groups – it is not possible to add file groups to **tempdb**.
- Change collation – **tempdb** inherits its collation from the server default and this cannot be changed.
- Set offline/read only – **tempdb** cannot be set offline or read only.

tempdb size can be monitored using the **sys.dm_db_file_space_usage** DMV.

For more information on the **tempdb** database, see MSDN:

tempdb Database

http://aka.ms/G7t3ny

Demonstration: Monitoring tempdb Usage

In this demonstration, you will see how to monitor **tempdb** usage.

Demonstration Steps

1. In SQL Server Management Studio, connect to the **MIA-SQL** database engine with Windows Authentication.
2. In SQL Server Management Studio, open the **monitortempdb.sql** Transact-SQL file in the **D:\Demofiles\Mod03** folder.
3. Highlight the Transact-SQL code under the comment **Amount of space in each tempdb file (Free and Used)**, and click **Execute**. Note the amount of space in each **tempdb** file.
4. Highlight the Transact-SQL code under the comment **Amount of free space in each tempdb file**, and click **Execute**. The amount of free space in each **tempdb** file is shown.
5. Highlight the Transact-SQL code under the comment **Amount of space used by the version Store**, and click **Execute**. Note the amount of **tempdb** space used by the version store.
6. Highlight the Transact-SQL code under the comment **Number of pages and the amount of space in MB used by internal objects**, and click **Execute**. The amount of **tempdb** space used by internal objects is returned.
7. Highlight the Transact-SQL code under the comment **Amount of space used by user objects in tempdb**, and click **Execute**. The amount of space used by user objects in **tempdb** is shown.
8. Close SQL Server Management Studio without saving changes.

Question: tempdb has run out of space causing your SQL Server instance to crash. How might you recover from this scenario?

Lab: Database Structures

Scenario

You have reviewed the AdventureWorks database and noticed high wait statistics for CPU, memory, I/O, blocking, and latching. In this lab, you will explore database structures and internals for a user database. You will enable instant file initialization and note the performance improvement. Finally, you will reduce **tempdb** latch contention by adding more data files to **tempdb**.

Objectives

After completing this lab, you will be able to:

- Explore database structures and data file internals.
- Improve performance by enabling instant file initialization.
- Reduce **tempdb** latch contention.

Estimated Time: 30 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Exploring Page Allocation Structure

Scenario

You have reviewed the AdventureWorks database and, amongst other things, noticed high wait statistics for I/O. Before investigating database data files and **tempdb** data files, you want to explore page and allocation structure.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Explore Page Structure
3. Explore Record Structure

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab03\Starter** folder as Administrator.

▶ Task 2: Explore Page Structure

1. Start Microsoft SQL Server Management Studio if it is not already running and connect to the **MIA-SQL** database instance using Windows Authentication.
2. Analyze page structure for the **Person.ContactType** table in the **AdventureWorks** database.

▶ Task 3: Explore Record Structure

- Turn on trace flag 3604 and use DBCC PAGE to analyze record structure for the **Person.Contact** table in the **AdventureWorks** database.

Results: After completing this exercise, you will have explored data page and record structure.

Exercise 2: Configuring Instant File Initialization

Scenario

You have reviewed the AdventureWorks database and, amongst other things, noticed high wait statistics for I/O. One potential cause that you have identified is that instant file initialization is not being used. In this exercise, you will enable instant file initialization and record performance improvement.

The main tasks for this exercise are as follows:

1. Reset Security Policy
2. Record Workload Execution Time
3. Enable Instant File Initialization and Compare Run Time

▶ Task 1: Reset Security Policy

1. Use the **Local Security Policy** tool to identify which users have the **Perform volume maintenance** right.
2. Remove this right from the **Administrators** group.

▶ Task 2: Record Workload Execution Time

1. In SQL Server Management Studio, restart the SQL Server Services.
2. Open the file **InstantFileInit.sql** in the folder D:\Labfiles\Lab03\Starter and execute the code.
3. Note how long the script takes to complete.

▶ Task 3: Enable Instant File Initialization and Compare Run Time

1. Use the **Local Security Policy** tool to identify which users have the **Perform volume maintenance** right.
2. Add the right for the **Administrators** group.
3. Restart the SQL Server services, open the file **InstantFileInit.sql** in the folder D:\LabFiles\Lab03\starter if it is not already open, and then execute the code.
4. Note how long the script takes to complete.
5. Compare the run time of the script with and without instant file initialization enabled.

Results: At the end of this exercise, you will have enabled instant file initialization.

Exercise 3: Reconfiguring tempdb Data Files

Scenario

You have reviewed the AdventureWorks database and noticed, among other things, high wait statistics for latches. You have identified latch contention in **tempdb**. In this exercise, you will add more data files to **tempdb** to reduce latch contention.

The main tasks for this exercise are as follows:

1. Execute Workload and Record Latch Contention Metrics
2. Add Additional Data Files to tempdb
3. Measure Performance Improvement

▶ Task 1: Execute Workload and Record Latch Contention Metrics

1. Execute the script **tempdbLoad.cmd D:\Labfiles\Lab03\Starter** as an administrator.
2. When all the command windows have closed, use the **sys.dm_os_wait_stats dmv** to record the LATCH waits for the **MIA-SQL** server instance.

▶ Task 2: Add Additional Data Files to tempdb

- Open the file **addTempdbFiles.sql** in SQL Server Management Studio and execute the code to create seven additional **tempdb** data files.

▶ Task 3: Measure Performance Improvement

1. Compare the wait stats figures before and after the additional **tempdb** files were added, noting the reduced waits with more **tempdb** files.
2. Close SQL Server Management Studio without saving changes.

Results: After completing this lab, **tempdb** will be using multiple data files.

Module Review and Takeaways

This module covered database structures, data files, and **tempdb** internals. It focused on the architectural concepts and best practices related to data files, for user databases and **tempd**b, which enable you to configure SQL Server for maximum performance.

Module 4

SQL Server Memory

Contents:

Module Overview

This module covers Windows® and SQL Server® memory internals, in addition to In-Memory OLTP. It focuses on the architectural concepts and best practices related to SQL Server memory configuration that means you can tune a SQL Server instance for maximum performance.

Objectives

After completing this module, you will be able to:

- Describe Windows memory.
- Describe SQL Server memory.
- Describe In-Memory OLTP.

Lesson 1
Windows Memory

This lesson focuses on Windows memory concepts at a high level. It will serve as a foundation for the next lesson in understanding SQL Server memory concepts.

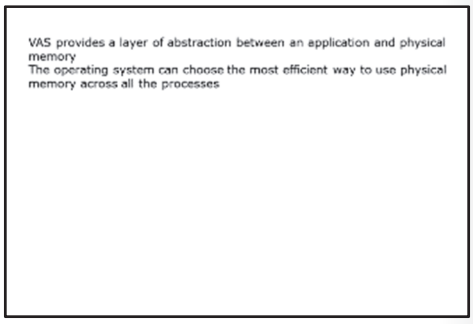
Lesson Objectives

After completing this lesson, you will be able to:

- Describe virtual address space (VAS).
- Describe physical versus virtual memory.
- Describe 32-bit versus 64-bit memory.
- Describe non-uniform memory access (NUMA).

Virtual Address Space

You can monitor memory on a Windows operating system by using various methods. The most common and easy way is to use Task Manager. To do this, on the **Performance** tab, view the memory resources as described in the following list:



- **Memory**: this is the total amount of RAM provisioned on the system.
- **Cached**: this is the amount of physical memory that is most recently used for the system resources.
- **Available**: this is the amount of memory that is readily available to be used by the processes, operating system, or the drivers.

Virtual Address Space

VAS for any process is the set of virtual addresses that process can use. This is independent of the physical memory. VAS is private to the process and can only be accessed by that specific process unless it is shared.

On 32-bit systems, the VAS is limited to 2 GB for user processes. This can be extended to 3 GB by using the 4-gigabyte tuning (4 GT) method. The memory range in a 32-bit system is described in the following table:

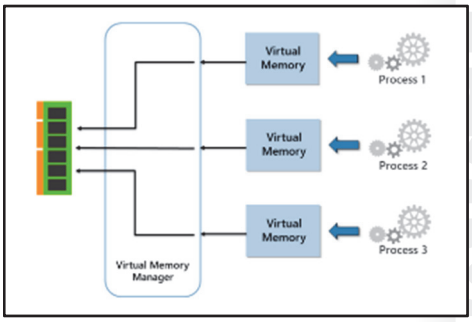
Option	Memory Usage for Process	Memory Usage for System
Without 4 GT	2 GB	2 GB
With 4 GT	3 GB	1 GB
Custom	2 GB to 3 GB	1 GB to 2 GB

A custom option is one where the windows boot configuration data command-line tool, BCDEdit, has been used to set a custom user VAS.

On a 64-bit system, the user VAS is limited to 8 TB for user processes. Also on a 64-bit system, the accessible address space is 16 Exabytes (2^64). This limit is very high for current available hardware; therefore, address bus is limited to 44 bits. This allows access of 16 TB of address space: 8 TB for user mode usage and 8 TB for kernel mode usage.

A key point to remember with VAS is that two different processes can both use the memory address 0xFFF because it is a virtual address and each process has its own VAS with the same address ranges.

Physical vs. Virtual Memory



Physical memory refers to the volatile storage space most commonly named random access memory (RAM). This is sometimes also referred to as primary storage or main memory. The physical memory is mostly interpreted as RAM, but it also includes the page file. RAM is the fastest accessible storage. The speed of a RAM is measured in gigabytes per second (GB/s); with nanosecond response time when compared to megabytes per second (MB/s); with millisecond response time for hard disks; and with microsecond response time for solid state disks. Compared to non-volatile storage, RAM is costlier.

With larger amounts of physical memory becoming very common recently, allowing all applications direct access to physical memory leads to performance problems. Therefore, Windows introduces an abstract layer above the physical memory, referred to as virtual memory.

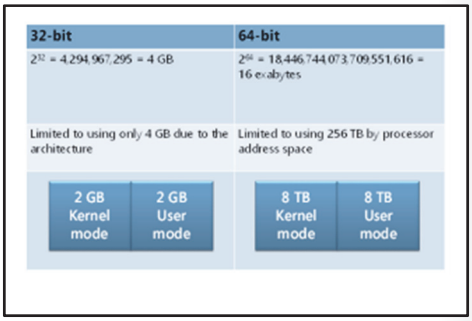
Windows provides a virtual address space to each application, commonly referred to as virtual memory. This allows the applications to refer to the same address space. This also means the operating system can use the underlying physical memory more efficiently among all the processes. The mapping of virtual memory to the underlying physical memory is managed by the Virtual Memory Manager (VMM).

When an application makes a request for memory, it accesses its virtual memory. The VMM maps the address in virtual memory to physical memory. When under memory pressure, the physical page is moved to page file and the virtual memory reference is marked as invalid. When the application tries to reuse the memory space, the page gets loaded from page file into main memory. This is known as page faults. The application remains unaware of these background processes, which are completely managed by the operating system.

The size of virtual memory allocated to each application depends on the processor architecture. With 32-bit architecture, up to 4 GB of memory can be accessed. With 64-bit architecture, up to 16 exabytes of memory can, in theory, be accessed.

32-bit vs. 64-bit

32- and 64-bit versions of Windows have different limits on memory and how it is used.



32-bit

32-bit processor architecture has a 32-bit address size limiting the maximum addressable memory to 4 GB.

Windows divides the physical memory into system memory (kernel mode) and user memory (user mode). The high memory address range is used by the system and the low memory address range is used by the user processes or applications. So, in a 32-bit system, the user applications can access only 2 GB of memory without any special configurations.

The methods that you can implement to use more than 2 GB of memory for user applications are described in the following list:

- **/3GB**

  The /3GB switch is also known as 4 GT (4 gigabit tuning). Use this switch to allow applications to use up to 3 GB of RAM, leaving the kernel to use memory between 1 and 2 GB. This only extends the maximum usage limit of user applications to 3 GB. Under memory pressure, the system can use up to 2 GB, reducing the user applications memory to less than 3 GB. You can set the /GB switch in boot.ini in Windows 2003 or earlier operating systems. In Windows 2008 and later versions, use the following command to set this configuration:

  ```
  BCDEdit /set increaseUserVA 3072
  ```

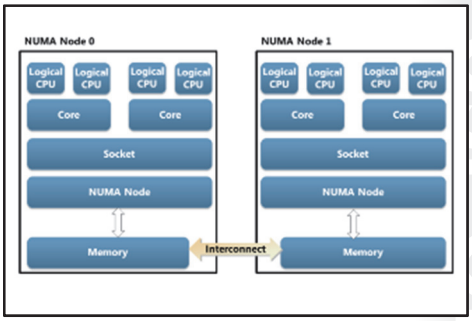
- **/PAE**

  The Physical Address Extension (/PAE) switch is an Intel-specific technology. Setting this switch will increase the address bus to use 36 bits. By enabling this option, memory address space increases to 2^36 or 64 GB. This allows user applications to extend the memory usage to 64 GB. This option is only possible when Address Windowing Extension (AWE) is enabled in SQL Server. AWE has been deprecated from SQL Server 2012 and 32-bit SQL Server is now limited to 4 GB. If SQL Server needs to use more than 4 GB of memory, it should be upgraded to a 64-bit version.

64-bit

64-bit processor architecture has a 64-bit address size, giving access to considerably more memory than 32-bit architecture. The theoretical maximum memory that can be addressed with 64-bit architecture is 16 exabytes (16,777,216 terabytes). However, processor manufacturers limit the address size on 64-bit processors to 48 bits for AMD64 processors and 42 bits for Intel processors, giving a maximum addressable memory of 256 TB.

NUMA



Traditional symmetric multiprocessing (SMP) systems have a single main memory that can only be accessed by a single processor at any one time. In multiple processor systems, this design creates a bottleneck because each processor must wait until memory is free. The performance bottleneck increases with the number of processor cores and becomes significant with eight or more cores. Memory NUMA is a hardware solution that makes separate memory available to each processor, to address the memory bottleneck issue.

NUMA systems have multiple cores that are divided into multiple nodes. Each node has memory on the local node which is local to the CPUs on that node. The CPUs on one NUMA node can access the memory from other NUMA nodes, but access will be slower. This is referred to as foreign access or remote access. All the nodes are connected through high-speed interconnect that allows the access of foreign memory. Even though foreign memory access is slower than local memory access, it is significantly faster than accessing the data from the disk subsystem.

With NUMA, CPUs can be made scalable. Traditional systems hit performance limitations and bottlenecks when the CPUs count increases beyond eight CPUs. With NUMA, a system can scale to 100s of CPUs.

Software NUMA is the implementation of NUMA on non-NUMA systems. Often referred to as soft-NUMA, it allows the grouping of CPUs into smaller sets. While soft-NUMA gives the benefit of more IO threads, it does not give CPU the memory affinity that NUMA hardware provides.

Question: What are some advantages of the Windows Virtual Address Space method of memory management over having applications access memory directly?

Lesson 2
SQL Server Memory

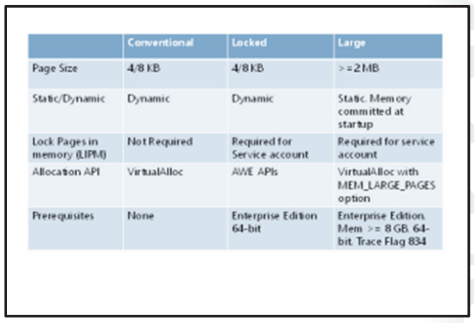
This lesson focuses on SQL Server Memory. You will learn about the internals of SQL Server Memory, how memory is managed in SQL Server, and how to configure SQL Server to use memory efficiently.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe SQL Server memory models.
- Describe SQL Server memory architecture.
- Describe SQL Server memory allocation.
- Understand how SQL Server uses NUMA.
- Describe SQL Server memory configuration.

SQL Server Memory Models



	Conventional	Locked	Large
Page Size	4/8 KB	4/8 KB	>=2 MB
Static/Dynamic	Dynamic	Dynamic	Static. Memory committed at startup
Lock Pages in memory (LPIM)	Not Required	Required for Service account	Required for service account
Allocation API	VirtualAlloc	AWE APIs	VirtualAlloc with MEM_LARGE_PAGES option
Prerequisites	None	Enterprise Edition 64-bit	Enterprise Edition. Mem >= 8 GB. 64-bit. Trace Flag 834

There are three types of memory models in SQL Server:

- Conventional memory model
- Locked memory model
- Large memory model

Conventional Memory Model

The conventional memory model is the common model for allocating memory in SQL Server. The physical page size is 4 or 8 KB. This is the only model used in 32-bit SQL Server and is available in all editions of SQL Server. With 64-bit SQL Server, additional models are provided to handle different types of memory allocations. In the conventional memory model, pages are allocated dynamically and can be paged from physical memory. The conventional memory model does not require any additional settings or configuration.

Locked Memory Model

The locked memory model is also used for 4 or 8 KB page allocations. The memory allocations in this model are also dynamic; however, they prevent the paging-out to disk of pages allocated to the SQL Server process. To enable the locked memory model, you need to assign the Lock Pages in Memory (LPIM) permission to the SQL Server service account memory.

Use the following steps to assign this permission:

1. Right-click **Start**, type **gpedit.msc**, and then click **gpedit.msc**.
2. In the **Local Group Policy Editor** console, expand **Computer Configuration**, expand **Windows Settings**, expand **Security Settings**, expand **Local Policies**, and then click **User Rights Assignment**. The policies will be displayed in the details pane.
3. In the details pane, double-click **Lock pages in memory**.
4. In the **Lock pages in memory Properties** dialog box, click **Add User or Group**.
5. In the **Select Users, Service Accounts, or Groups** dialog box, add the SQL Server service account.
6. Restart the computer for the permissions to take effect.

The locked memory model is only available in the 64-bit Enterprise Edition of SQL Server.

You can confirm whether an instance of SQL Server is using locked memory model by checking the error log for the following message at startup:

```
Using locked pages in the memory manager.
```

You can check the size of locked pages by using the following query:

Check size of locked pages

```
SELECT locked_page_allocations_kb FROM sys.dm_os_process_memory;
```

Large Memory Model

The large memory model is used for allocating large pages of 2 MB and above in SQL Server. This model is supported in the 64-bit Enterprise Edition of SQL Server only on a computer with more than 8 GB of memory. The large memory model needs the LPIM privileges assigned to the SQL Server service account—trace flag 834 should also be enabled. In the large memory model, SQL Server allocates all of the buffer pool memory at startup. Trace flag 834 improves performance by increasing the efficiency of the translation look-aside buffer (TLB) in the CPU.

The large pages memory allocation is loaded at SQL startup and is never released or increased—the allocation is static. When the instance starts up, the following message is recorded in the Error log:

```
Using large pages for buffer pool.
```

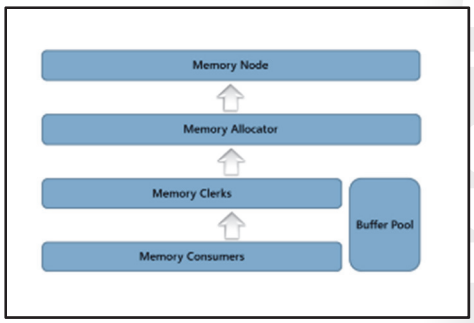
The Large Page Granularity is the minimum size of the large page on the given Windows platform. SQL Server engine calls the Windows API GetLargePageMinimum() to get this information. On a 64-bit platform, this size is 2 MB. For large pages, there is an allocation of 32 MB for each memory node.

The total amount of space allocated for large pages can be checked using the following query:

Check large page allocations

```
SELECT large_page_allocations_kb FROM sys.dm_os_process_memory
```

SQL Server Memory Architecture



Memory in SQL Server is split into several components.

- **Memory Nodes**: memory nodes are logical chunks of memory. For NUMA machines, there will be a memory node for each NUMA node. For non-NUMA machines, there is a single memory node.
- **Memory Allocator**: the memory allocator allocates all memory on the memory nodes. Memory It handles all requests, forwarding them to the appropriate API.
- **Memory Clerks**: a memory clerk exists for each of the major memory consumers in SQL Server. Memory It manages memory for the consumer and allows easy monitoring. There are four different categories of memory clerk:
    - **Cachestore**: these clerks have multiple hash tables and flush out entries using an algorithm, based on a Least Recently Used policy. The procedure cache is an example of a cachestore memory clerk.
    - **Generic**: generic clerks sit outside of the SQL OS framework but retain the ability to respond to memory pressure. The buffer pool is an example of a generic memory clerk, although the buffer pool is a special case acting as a memory clerk and a memory consumer.
    - **Object Store**: the object store clerks, sometimes referred to as the memory pool, hold collections of similar objects. Locks are an example of an object store clerk.
    - **User Store**: user store clerks are clerks used by developers for implementing their own caching mechanisms and retention profiles.

You can find details about memory clerks in use on a SQL Server instance from the **sys.dm_os_memory_clerks** Dynamic Management View (DMV).

When a memory clerk requires memory, it must request it from the memory allocator. A memory clerk cannot go directly to a memory node.

For small memory requests (bytes rather than megabytes) the memory allocator will use the Heap Allocation API. For larger allocations, the Virtual Alloc API is used, providing the flexibility for independent memory management. Where the SQL Server service account has the LPIM privilege, the memory allocator may also use the AWE APIs for memory allocation.

Memory Consumers

Memory consumers are the end users of memory. Some of the most important memory consumers are:

- **Buffer Pool**: the buffer pool acts as both a memory clerk and a memory consumer. The buffer pool is optimized for handling 8 KB pages and other consumers that require memory in 8 KB chunks, such as the Database Page Cache—this will request memory from the buffer pool to improve performance.
- **Backup Buffer**: requests for memory from backup buffer consumers are handled by the SQLUtilities memory clerk that calls Virtual Alloc to allocate memory.
- **Plan Cache**: where the plan cache consumer requires a single page of memory, it requests this from the buffer pool, from the SQLQUERYPLAN memory clerk. For multipage requests, the SQLQUERYPLAN memory clerk will invoke the multipage allocator.
- **Optimizer**: the optimizer consumer requests memory from the SQLOPTIMIZER memory clerk. The optimizer releases memory fairly quickly and therefore the SQLOPTIMIZER clerk acts primarily as a monitor.

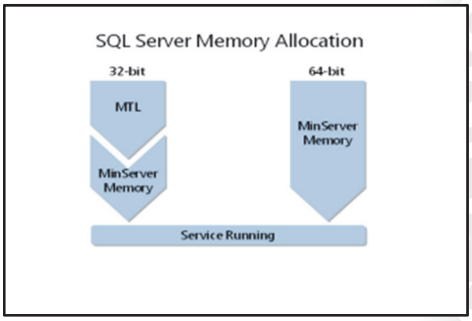
Memory Pools

SQL Server primarily uses memory pools to hold different types of data structures, including system level data and caches. Memory clerks use memory pools to allocate memory to different consumers.

Details of memory pool usage can be obtained by querying the **sys.dm_os_memory_pools** DMV.

SQL Server Memory Allocation

Due to differences in version architecture, the SQL Server memory allocation process is different in 32-bit and 64-bit versions of SQL Server.



32-bit

At SQL Server startup, memory to leave (MTL) is the first memory that is reserved by SQL Server. MTL is separate from the buffer pool and is contiguous. It is reserved to use for external memory consumers of SQL Server such as common language runtime (CLR), thread stack, and extended procedures. Before SQL Server 2012, MTL was also used for multipage allocations; that is, pages larger than 8 KB.

SQL Server calculates how much memory to allocate for MTL using the following formula:

(MaxWorkerThreads * StackSize) + DefaultReservationSize

The default StackSize value is 512 KB for 32-bit versions of SQL Server and the MaxWorkerThreads value in the formula is calculated dynamically using the following formula:

((NumberOfSchedulers – 4) * 8) + BaseThreadCount

The default BaseThreadCount value is 256 for 32-bit versions of SQL Server.

The DefaultReservationSize value has a default value of 256 MB. You can manage the value by using the -g parameter in the SQL Server startup parameters. However, it is recommended that you only change the default value if the following messages appear in the error log:

"Failed Virtual Allocate Bytes: FAIL_VIRTUAL_RESERVE <size>"

"Failed Virtual Allocate Bytes: FAIL_VIRTUAL_COMMIT <size>"

In versions of SQL Server before SQL Server 2012, buffer pool memory was allocated after MTL, but the buffer pool is now included in Min Server Memory allocation.

When MTL reservation is complete, SQL Server reserves the Min Server Memory. SQL Server may not allocate the entire Min Server Memory if it is not needed.

64-bit

In 64-bit SQL Server, MTL is not reserved but the MaxWorkerThreads value is still calculated based on the following formula and is reserved at SQL Server startup:

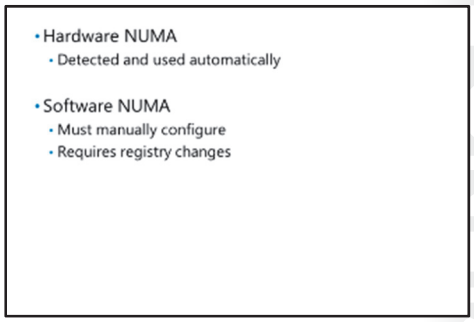
((NumberOfSchedulers – 4) * 16) + BaseThreadCount

The default BaseThreadCount value is 512 for 64-bit versions of SQL Server.

In versions of SQL Server before SQL Server 2012, buffer pool memory was allocated next, but the buffer pool is now included in Min Server Memory allocation.

When MTL reservation is complete, SQL Server reserves the Min Server Memory. SQL Server may not allocate the entire Min Server Memory if it is not needed.

SQL Server NUMA



SQL Server is NUMA-aware, detecting and using NUMA hardware automatically. You can check NUMA node configuration using the performance monitor counter *NUMA Node Memory*.

You can also check NUMA node configuration from SQL Server by running the following query:

You can also check NUMA node configuration from SQL Server by running the following query:

NUMA node configuration.

```
SELECT DISTINCT memory_node_id FROM sys.dm_os_memory_nodes
```

You can use soft-NUMA with SQL Server to subdivide machines and give better locality. Utilizing soft-NUMA is a two-step process:

1. Set the processor affinity mask in SQL Server using the ALTER SERVER CONFIGURATION command.
2. Modify the registry to add the soft-NUMA mappings.

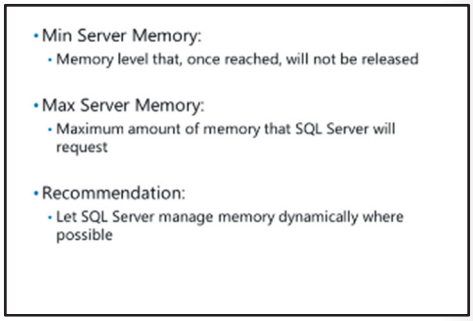
You should back up the registry before making any modifications because incorrectly modifying the registry can cause serious damage to a system.

For more details on soft-NUMA configuration, see:

Configure SQL Server to Use Soft-NUMA (SQL Server)

http://aka.ms/Ic6rei

SQL Server Memory Configuration



It is recommended that you allow SQL Server to manage memory dynamically; however, there are situations where it is necessary to set limits on the amount of memory available to SQL Server. Do this by using the Min Server Memory and Max Server Memory options.

SQL Server reserves the minimum memory required at the startup of an instance. It uses more memory as required and only releases memory back to Windows if there is memory pressure on the operating system.

After reserving the minimum required memory, SQL Server starts asking for more memory from Windows as required and, from then on, grows beyond the Min Server Memory setting. After the Min Server Memory value is reached, SQL Server will not release memory below Min Server Memory back to the operating system, even if the operating system suffers memory pressure. This prevents SQL Server crashing under memory pressure.

The Max Server Memory setting is the maximum amount of memory that SQL Server is permitted to use and can prevent SQL Server from consuming all the available system memory. Memory allocations for external objects such as VAS allocator, CLR, and extended procedures, happen outside the Max Server Memory. This means that setting the Max Server Memory to a value that leaves sufficient free memory for the operating system, external objects, and any other applications running on the server, can help to avoid out-of-memory errors in SQL Server.

You can set Min Server Memory and Max Server Memory in the SQL Server Management Studio GUI or by using Transact-SQL commands.

The following Transact-SQL sets the Min Server Memory to 512 MB and the Max Server Memory to 2,048 MB:

Set Min Server Memory and Max Server Memory

```
EXEC  sp_configure N'Max Server memory (MB)',N'512'
EXEC sp_configure N'Max Server memory (MB)',N'2048'
```

After setting the Min Server Memory and Max Server Memory values, you must issue the RECONFIGURE command as shown in the following code or restart the service:

```
Reconfigure with override
```

You can establish whether SQL Server is under memory pressure by looking at the metrics:

- **Buffer cache hit ratio**. The number of pages SQL Server was able to fetch from memory.
- **Page life expectancy**. The length of time a page stays in the buffer pool.
- **MEMORY_ALLOCATION_EXT** wait. Waits associated with allocating memory from the operating system or the internal SQL Server pool.
- **RESOURCE_SEMAPHORE** wait. An internal SQL Server worker process executing a query is waiting for a memory request to be granted. This wait occurs when many simultaneous queries exhaust the allocated memory. This will correspond to an increase in the value of the **SQLServer:Memory Manger\Memory Grants Pending** performance counter.
- **Working set trimmed** log message. Under external memory pressure, the working set memory of SQL Server is trimmed and paged out to the page file. An error message containing the text "**A significant part of SQL Server process memory has been paged out**" will be recorded in the SQL Server error log.

A SQL Server instance that is not experiencing memory pressure will generally have a buffer cache hit ratio above 90 percent and a high page life expectancy. If either value is low or they change rapidly, then the SQL Server instance is experiencing memory issues that need investigating.

Buffer cache hit ratio and page life expectancy can be checked by querying the **sys.dm_os_performance_counters** DMV.

Query for obtaining buffer cache hit ratio for a SQL Server instance.

Obtain Buffer cache hit ratio.

```
SELECT (pc.cntr_value * 1.0 / pcb.cntr_value) * 100.0 as BufferCacheHitRatio
FROM sys.dm_os_performance_counters  pc
JOIN  (SELECT cntr_value, OBJECT_NAME
    FROM sys.dm_os_performance_counters
    WHERE counter_name = 'Buffer cache hit ratio base'
        AND OBJECT_NAME = 'SQLServer:Buffer Manager') pcb ON  pc.OBJECT_NAME =
pcbb.OBJECT_NAME
WHERE pc.counter_name = 'Buffer cache hit ratio'
AND pc.OBJECT_NAME = 'SQLServer:Buffer Manager'
```

 **Note:** To obtain a useable metric, the counter buffer cache hit ratio must be divided by the buffer cache hit ratio base.

MEMORY_ALLOCATION_EXT waits can be queried using the **sys.dm_os_wait_stats** DMV.

Example code for obtaining memory allocation wait information:

sys.dm_os_wait_stats

```
SELECT * FROM sys.dm_os_wait_stats WHERE
wait_type = 'MEMORY_ALLOCATION_EXT'
```

For more information on the **sys.dm_os_wait_stats** DMV, see:

 **sys.dm_os_wait_stats (Transact-SQL)**

http://aka.ms/Cu73dm

Demonstration: Analyzing Memory Allocations

In this demonstration, you will see how to analyze SQL Server memory allocation.

Demonstration Steps

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Navigate to the folder D:\Demofiles\Mod04\Demo01 and run the file **Setup.cmd** as Administrator.
3. In the **User Account Control** dialog box, click **Yes**, and then wait until the script completes.
4. Click **Start** menu, type **SQL Server 2017 Configuration Manager**, and the press Enter.
5. In the **User Account Control** dialog, click **Yes**.
6. In SQL Server Configuration Manager, click **SQL Server Services** node, and restart the **SQL Server (MSSQLSERVER)** service.
7. In SQL Server Management Studio, connect to the **MIA-SQL** database instance.
8. Open a new query window and click **File**, **Open**, **File**. Navigate to the D:\DemoFiles\Mod04\Demo01 folder and double-click **MonitorMemory.sql** to open it. Execute the script.
9. Click the Start menu, open **Resource Monitor**, and on the memory tab note the values for **Working Set** and **Commit**.

10. In the Results pane, in SQL Server Management Studio, compare the values under the columns **physical_memory_in_use_kb** and **virual_address_space_committed_kb** with the values under the columns **Working Set(KB)** and **Commit(KB)** for the process **sqlservr.exe** in the Resource Monitor.
11. Close Resource Monitor.

Check Your Knowledge

Question	
Which of the following is an advantage of using the SQL Server locked memory model over the conventional memory model?	
Select the correct answer.	
	The locked memory model uses less RAM.
	The locked memory model will not allow pages allocated to SQL Server to be paged to disk.
	The locked memory model writes least used pages to disk, freeing up system memory.
	The locked memory model can run on a system with lower available memory.

Lesson 3
In-Memory OLTP

In this lesson, you will learn about In-Memory OLTP, memory-optimized tables and natively compiled stored procedures. These technologies can significantly improve the performance of queries that use them.

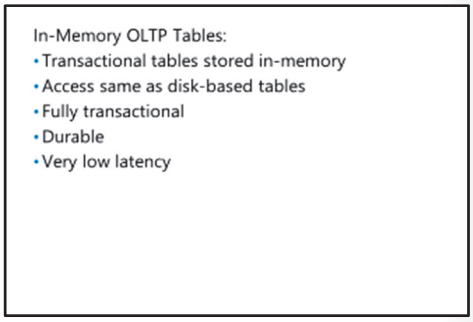
Lesson Objectives

After completing this lesson, you will be able to:

- Describe In-Memory OLTP.
- Describe memory-optimized tables.
- Describe natively compiled stored procedures.

What Is In-Memory OLTP?

In-Memory OLTP provides an ideal solution for improving the performance of OLTP business applications where there are a large number of short insert and update transactions, or there is business logic built into stored procedures. In-Memory OLTP takes advantage of the large amounts of memory and CPU cores found in modern servers to deliver significant improvement in OLTP database performance by using in-memory tables.



With In-Memory OLTP, performance gains of between five and 20 times can be achieved by using optimized algorithms, lock elimination, and compiled stored procedures.

In-memory tables are accessed using the same Transact-SQL commands as traditional disk-based tables; they are fully ACID (Atomic, Consistent, Isolated, and Durable) compliant, and have extremely low latency.

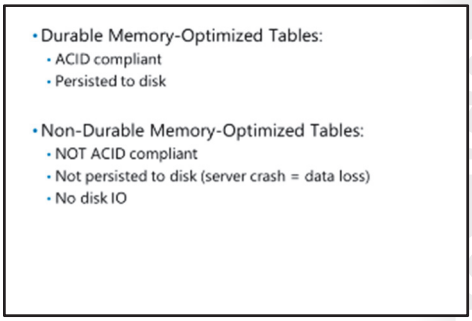
For more detailed information about In-Memory OLTP, see:

In-Memory OLTP (In-Memory Optimization)

http://aka.ms/S548m5

Memory-Optimized Tables

Memory-optimized tables reside in memory. You create them using the standard Transact-SQL **CREATE TABLE** statement with the **WITH (MEMORY_OPTIMIZATION=ON)** clause. You must create memory-optimized tables in a memory-optimized filegroup which you can add to an existing database using the standard **ALTER TABLE** statement with the **CONTAINS MEMORY OPTIMIZED DATA** clause.



There are two types of memory-optimized tables that can be created: durable and non-durable.

Durable Memory-Optimized Tables

Durable memory-optimized tables are the default type. They are fully ACID compliant and keep a second copy of data on disk for durability purposes. All read and write activity takes place in memory with data only being read from disk at startup.

Increased performance on durable memory-optimized tables can be obtained with delayed durable transactions; however, when you enable delayed durable transactions, the table is no longer ACID compliant and some committed transactions might be lost in the event of a server crash or failover.

Non-Durable Memory-Optimized Tables

Non-durable memory-optimized tables do not persist data to disk. They have no associated disk IO and therefore provide increased performance. Non-durable memory-optimized tables are not ACID compliant and data will be lost when the server is restarted or in the event of a failover or server crash. SQL Server stores the schema definition for non-durable memory-optimized tables and recreates the tables at server startup, making them ideal for storing temporary data, such as session information for a website.

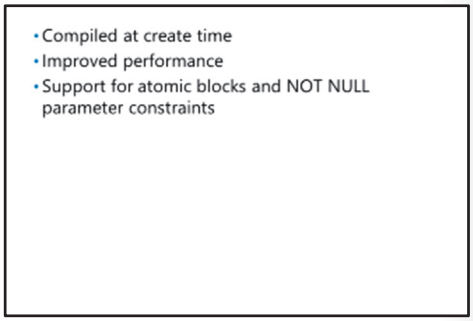
You can access data stored in memory-optimized tables using standard TRANSACT-SQL commands or natively compiled stored procedures.

For more details on memory-optimized tables, see:

Introduction to Memory-Optimized Tables

http://aka.ms/m6dx8h

Natively Compiled Stored Procedures



Natively compiled stored procedures are fully compiled when they are executed giving more efficient execution. You can create them by adding a **WITH native_compilation** clause to a standard **CREATE PROCEDURE** Transact-SQL statement.

Because natively compiled stored procedures are compiled when they are executed, SQL Server can detect and correct errors at compile time before the stored procedure is ever executed. Traditional stored procedures, which are only compiled when they are first executed, will show up many error conditions only during first execution.

Natively compiled stored procedures offer a runtime performance improvement over traditional stored procedures because they are pre-compiled into native code, and do not have to be interpreted or compiled at run time. Natively compiled stored procedures include support for some features not available in traditional stored procedures, including:

- **Atomic blocks**: blocks of code that succeed or are rolled back as a single unit.
- **Parameter NOT NULL constraints**: constraints on parameters and variables that prevent them being set to NULL.

There are some limitations on Transact-SQL inside natively compiled stored procedures. For more information, see:

Supported Features for Natively Compiled TRANSACT-SQL Modules

http://aka.ms/Dw0vt5

Demonstration: Creating Memory-Optimized Tables

In this demonstration, you will see how to:

- Create a Memory-Optimized Table.

Demonstration Steps

1. Ensure that the 10987C-MIA-DC and 10987C-MIA-SQL virtual machines are both running, and then log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Start SQL Server Management Studio and connect to the **MIA-SQL** instance database engine using Windows authentication.
3. Open the file **CreateMemoryOptTables.sql** located in the D:\Demofiles\Mod04\Demo02 folder.
4. Review and execute the Transact-SQL code, noting that it creates and populates two memory-optimized tables, one durable and one non-durable.
5. Open the file **SelectFromMemoryOptTables.sql** located in D:\Demofiles\Mod04\Demo02.
6. Execute the code preceding the comment **Restart SQL Server Services** and note both tables return results.

7. In Object Explorer, right-click **MIA-SQL** and then click **Restart**. In the **User Account Control** dialog box, click **Yes** to allow the restart to proceed.
8. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes** to restart the service, and then in the second **Microsoft SQL Server Management Studio** dialog box, click **Yes** to stop the SQL Server Agent service.
9. When the SQL Server service restarts, Execute the two SELECT statements again, noting that both tables still exist, but the non-durable table no longer contains data.
10. Close SQL Server Management Studio without saving changes.

Check Your Knowledge

Question	
One of your organization's key transactional systems is suffering from poor performance due to the volume of transactions on an important table. You decide to convert this table to a memory-optimized table. Which type of memory-optimized table mode would be most suitable?	
Select the correct answer.	
	Non-durable memory-optimized table.
	Durable memory-optimized table with delayed durable transactions.
	Memory-optimized tables are not suitable for transactional systems.
	Durable memory-optimized table.

Lab: SQL Server Memory

Scenario

You have reviewed statistics for the AdventureWorks database and noticed high wait stats for CPU, Memory, IO, Blocking, and Latching. In this lab, you will address the memory wait stats. You will set minimum and maximum memory configuration to appropriate values to reduce memory waits.

Objectives

After completing this lab, you will be able to:

- Configure SQL Server memory appropriately.

Estimated Time: 30 minutes.

Virtual machine: **10987C-MIA-SQL**

Username: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Reconfigure SQL Server Memory

Scenario

You have reviewed statistics for the AdventureWorks database and noticed high wait stats for memory, amongst others. One potential cause you have identified is that Min and Max memory configuration is not set correctly. In this exercise, you will set appropriate values for Min and Max memory for the SQL Server instance.

The main tasks for this exercise are as follows:

1. Execute Workload and Record Memory Wait

2. Set Min and Max Memory Appropriately

3. Execute Workload, Record Memory Wait and Measure Performance Improvement

▶ Task 1: Execute Workload and Record Memory Wait

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Navigate to the folder D:\Labfiles\Lab04\Starter, right-click the file **Setup.cmd**, and then click **Run as administrator**. In the **User Account Control** dialog box, click **Yes**.
3. Start SQL Server Management Studio and connect to the **MIA-SQL** SQL Server instance using Windows Authentication.
4. Execute the PowerShell™ script **loadscript.ps1** located in D:\Labfiles\Lab04\Starter to apply an artificial load to the system.
5. When the script completes, record stats for the **MEMORY_ALLOCATION_EXT** wait type.
6. Leave SQL Server Management Studio open for the next task.

▶ Task 2: Set Min and Max Memory Appropriately

1. Set Min Server Memory to 512 MB and Max Server Memory to 4,096 MB.
2. Restart the **MIA-SQL** SQL Server instance.

▶ Task 3: Execute Workload, Record Memory Wait and Measure Performance Improvement

1. Execute the PowerShell script **loadscript.ps1** located in D:\Labfiles\Lab04\Starter to apply an artificial load to the system.
2. When the script has completed, record stats for the **MEMORY_ALLOCATION_EXT** wait type again.
3. Compare the results you obtained before and after you changed Min Server Memory and Max Server Memory.

Results: After this lab, the SQL Server memory settings will be reconfigured.

Module Review and Takeaways

 **Best Practice:** Avoid setting the Min Server Memory and Max Server Memory in SQL Server to identical values and effectively fixing the amount of memory SQL Server uses. Allowing SQL Server to manage memory dynamically will give the best overall system performance.

Review Question(s)

Question: The Min Server Memory and Max Server Memory setting for your SQL Server have been left at the default values. Increasingly, you are noticing another application running on the same server is performing badly and the operating system is intermittently unresponsive. Having looked at the perfmon counters you have established the cause of the issues to be memory pressure. How might you address this?

Module 5

SQL Server Concurrency

Contents:

Module Overview

Concurrency control is a critical feature of multiuser database systems; it allows data to remain consistent when many users are modifying data at the same time. This module covers the implementation of concurrency in Microsoft® SQL Server®. You will learn about how SQL Server implements concurrency controls, and the different ways you can configure and work with concurrency settings.

Note: Transactions and locking are closely interrelated; it is difficult to explain either topic without reference to the other. This module covers transactions and concurrency before covering locking, but you will find some references to locking in the description of transactions.

Objectives

At the end of this module, you will be able to:

- Describe concurrency and transactions in SQL Server.
- Describe SQL Server locking.

Lesson 1
Concurrency and Transactions

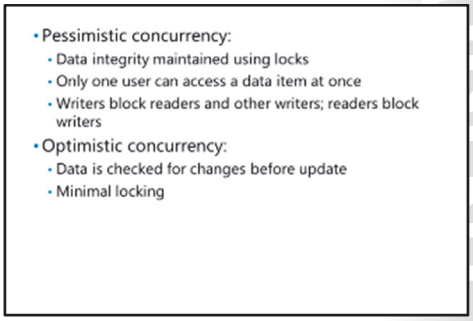
This lesson focuses on how SQL Server implements concurrency and transactions. You will learn about different concurrency models, and the strengths and weaknesses of each model. You will then learn about different types of isolation levels and transaction internals.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe different models of concurrency.
- Identify concurrency problems.
- Implement isolation levels.
- Work with row versioning isolation levels.
- Describe how SQL Server implements transactions.
- Explain best practices when working with transactions.

Concurrency Models



Concurrency can be defined as a system's ability to allow multiple users to access or change shared data simultaneously. The greater the number of active users able to work on shared data, the greater the level of concurrency. As the level of concurrency increases, the likelihood of conflicting data operations (where two or more users attempt to access or amend the same data at the same time) also increases.

There are two different approaches to resolving data conflicts during concurrent operation; these are pessimistic and optimistic concurrency.

Pessimistic Concurrency

The pessimistic concurrency model assumes that conflicting data operations will occur frequently. In this model, locks are used to ensure that only one user can access one data item at a time. While a data item is locked to one user, other users cannot access it. A pessimistic concurrency model exhibits the following properties:

- Data being read is locked, so that no other user can modify the data.
- Data being modified is locked, so that no other user can read or modify the data.
- The number of locks acquired is high because every data access operation (read/write) acquires a lock.
- Writers block readers and other writers. Readers block writers.

The pessimistic concurrency model is suitable for a system where:

- Data contention is high.
- Locks are held for a short period of time.
- The cost of preventing conflicts with locks is lower than the cost of rolling back the change, in the case of a concurrency conflict.

Optimistic Concurrency

The optimistic concurrency model assumes that conflicting data operations will occur infrequently. In this model, locks are not used; instead, the state of the affected data is recorded at the start of a data operation. This state is checked again at the end of the operation, before any changes are written. If the state has not changed, new changes are written. If the state has changed, the new changes are discarded and the operation fails. An optimistic concurrency model exhibits the following properties:

- Data being read is not locked; other users may read or modify the data.
- Data being modified is not locked; other users may read or modify the data.
- Before modified data is written, it is checked to confirm that it has not changed since being read; only if it has not changed will the changes be written.
- The number of locks acquired is low.

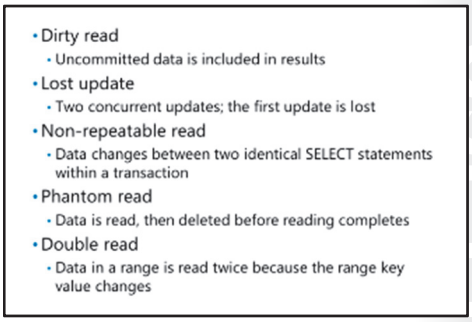
The optimistic concurrency model is suitable for a system where:

- Data contention is low.
- Data modifications may take long periods of time.
- The cost of rolling back and then retrying a change is lower than the cost of holding locks.
- Readers should not block writers.

SQL Server supports implementations of both optimistic concurrency and pessimistic concurrency. Pessimistic concurrency is the default concurrency model for the database engine. The In-Memory OLTP Engine uses a type of optimistic concurrency called row versioning; it does not implement pessimistic concurrency.

Concurrency Problems

There are several categories of problems that may occur when concurrency control is lacking or insufficient, and multiple sessions attempt to access or change the same data item.



Dirty Read

A dirty read occurs when one transaction reads a row that is in the process of being modified by another transaction. The reading transaction reads uncommitted data that may be changed later by a transaction modifying the data.

For example, user A changes a value from *x* to *y*, but does not finalize the change by committing the transaction. A second user, user B, reads the updated value *y* and performs processing based on this value. User A later changes the value again, from *y* to *z*, and commits the transaction. User B reads the uncommitted (dirty) value.

Lost Update

A lost update occurs when one or more transactions simultaneously updates the same row, based on the same initial value. When this happens, the last transaction to update the row overwrites the changes made by other transaction(s), resulting in lost data.

For example, user C and user D select value *x* to update. User C first updates the value from *x* to *y*, and then user D updates the value *x* to *z*. The modifications made by user A are overwritten by user B, resulting in data loss.

Non-Repeatable Read

A non-repeatable read occurs when a transaction reads different values for the same row each time the row is accessed. This happens when data is changed by another transaction in between two SELECT statements.

For example, user E begins a transaction that contains two similar SELECT statements, *s1* and *s2*. The SELECT statement *s1* reads value *x*, and then does some processing. Another user, user F, modifies value *x* to *y* while user E is executing other queries. When user E subsequently executes *s2*, the value *y* is returned instead of the initial *x*.

Phantom Read

A phantom read is a variation of a non-repeatable read. Phantom reads occur when one transaction carries out a DELETE operation or an INSERT operation against a row that belongs to the range of rows being read by another transaction.

For example, user G has two similar SELECT statements, *s1* and *s2*, within a transaction; the SELECT statement *s1* reads the count of rows as *n*, and then does other processing. Meanwhile, another user, user H, deletes a row from the range of rows being read by select statement *s1* and *s2*. When user G returns to execute *s2*, the row count is *n-1*. The SELECT statement *s1* returns a phantom read for a row that does not exist at the end of user G's transaction.

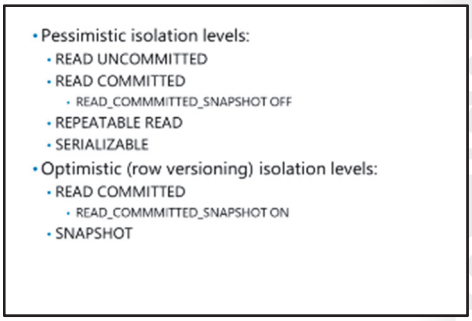
Double Read

A double read occurs when a transaction reads the same row value twice when reading a range of rows. This happens when the row value that defines the range is updated by another transaction while the range is being read.

For example, user I executes a SELECT statement that returns rows with values in a range *a* to *z*, that is implemented as an index scan. After the scan has read rows with value *a*, but before the scan completes, another user, user J, updates a row with value *a* to value *z*. The updated row is read again when the scan reaches rows with value *z*.

Note: It is also possible for this issue to miss a row, but this is still referred to as a double read problem. In the example, a row could be updated from value *z* to value *a* while the scan was running.

Transaction Isolation Levels



You can use transaction isolation levels to control the extent to which one transaction is isolated from another, and to switch between pessimistic and optimistic concurrency models. Transaction isolation levels can be defined in terms of which of the concurrency problems (that you learned about earlier in this lesson) are permitted. A transaction isolation level controls:

- Whether locks should be acquired when data is being read and the type of locks to be acquired.
- The duration that the locks are held.
- Whether a read operation accessing rows being modified by another transaction:
  - Is blocked until the exclusive lock on the row is released.
  - Fetches the committed data present at the time the transaction started.
  - Reads the uncommitted data modification.

The transaction isolation level controls only whether locks are to be acquired or not for read operations; write operations will always acquire an exclusive lock on the data they modify and hold the lock until the transaction finishes, whatever the isolation level of transaction.

Isolation levels represent a trade-off between concurrency and consistency of data reads. At lower isolation levels, more concurrent data access is possible, but you experience more concurrency problems. At higher isolation levels, concurrency is reduced, but you experience fewer concurrency problems.

Five isolation levels are available in SQL Server:

READ UNCOMMITTED

READ UNCOMMITTED is the lowest level of isolation available in SQL Server. The READ UNCOMMITTED isolation level has the following properties:

- No locks are taken for data being read.
- During read operations, a lock is taken to protect the underlying database schema from being modified.
- Readers do not block writers, and writers do not block readers; however, writers do block writers.
- All of the concurrency problems (dirty reads, non-repeatable reads, double reads, and phantom reads) can occur.
- Data consistency is not guaranteed.
- Not supported on FILESTREAM enabled databases.

READ COMMITTED

READ COMMITTED is the SQL Server default isolation level. The READ COMMITTED isolation level has the following properties when the READ_COMMITTED_SNAPSHOT database option is OFF (the default for SQL Server installations):

- Read locks are acquired and held until the end of the statement.
- Dirty reads are eliminated by allowing access to committed data only.
- Because read locks are held until the end of the statement, data can be changed by other transactions between individual statements within the current transaction, resulting in non-repeatable reads, double reads, and phantom reads.

When the READ_COMMITTED_SNAPSHOT database option is ON (the default for Azure SQL Database), the READ COMMITTED isolation level has the following properties:

- Row versioning is used to provide statement-level read consistency. Because each statement in a transaction executes, a snapshot of old data is taken and stored in version store. The snapshot is consistent until the statement finishes execution.
- Read locks are not held because the data is read from the version store, and not from the underlying object.
- Dirty reads do not occur because a transaction reads only committed data, but non-repeatable reads and phantom reads can occur during a transaction.

READ COMMITTED isolation is supported on FILESTREAM enabled databases.

REPEATABLE READ

REPEATABLE READ has the following properties:

- Read locks are acquired and held until the end of the transaction. Therefore, a transaction cannot read uncommitted data and cannot modify the data being read by other transactions until that transaction completes.
- Eliminates non-repeatable reads. Phantom reads and double reads still occur. Other transactions can insert or delete rows in the range of data being read.
- Not supported on FILESTREAM enabled databases.

SERIALIZABLE

SERIALIZABLE is the highest level of isolation available in SQL Server. It has the following properties:

- Range locks are acquired on the range of values being read and are held until the end of the transaction.
- Transactions cannot read uncommitted data, and cannot modify the data being read by other transactions until the transaction completes; another transaction cannot insert or delete the rows in the range of rows being read.
- Provides lowest level of concurrency.
- Not supported on FILESTREAM enabled databases.

SNAPSHOT

SNAPSHOT isolation is based on an optimistic concurrency model. SNAPSHOT isolation has the following properties:

- Uses row versioning to provide transaction-level read consistency. A data snapshot is taken at the start of the transaction and remains consistent until the end of the transaction.
- Transaction-level read consistency eliminates dirty reads, non-repeatable reads, and phantom reads.
- If update conflicts are detected, a participating transaction will roll back.
- Supported on FILESTREAM enabled databases.
- The ALLOW_SNAPSHOT_ISOLATION database option must be ON before you can use the SNAPSHOT isolation level (OFF by default in SQL Server installations, ON by default in Azure SQL Database).

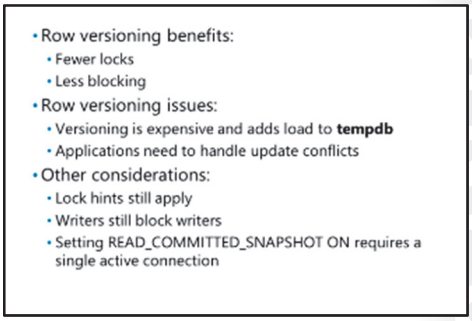
For more information on transaction isolation levels, see the topic *SET TRANSACTION ISOLATION LEVEL (Transact-SQL)* in Microsoft Docs:

SET TRANSACTION ISOLATION LEVEL (Transact-SQL)

http://aka.ms/faim9a

Working with Row Versioning Isolation Levels

Row versioning isolation levels (SNAPSHOT isolation, and READ COMMITTED isolation with READ_COMMITTED_SNAPSHOT ON) have costs as well as benefits. In particular, row versioning makes use of **tempdb** to hold versioning data; you should ensure your storage subsystem can accommodate the additional load on **tempdb** before enabling a row versioning isolation level.



In general, row versioning based isolation levels have the following benefits:

- Readers do not block writers.
- Fewer locks overall—SELECT statements do not acquire locks:
  - Reduced blocking and deadlocks.
  - Fewer lock escalations.

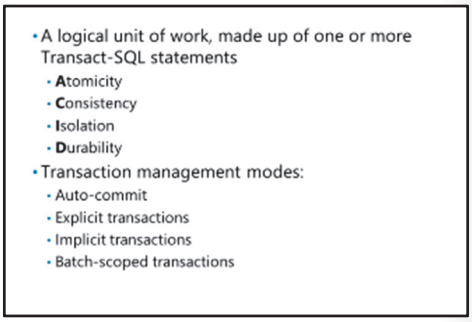
Row versioning-based isolation levels can cause the following issues:

- Read performance may degrade because the set of versioned rows ages and large version chains must be scanned.
- Increased resource utilization in maintaining row versions in **tempdb**.
- Versions are maintained even when there is no active transaction using a row versioning isolation level.
- SNAPSHOT isolation may result in transaction rollback because of update conflict. Applications may need to be modified to handle update conflicts.

Other points you should consider include:

- SNAPSHOT isolation does not affect queries with lock hints. The lock hints still apply.
- Writers still block writers.
- Setting READ_COMMITTED_SNAPSHOT ON requires that only one connection is open (the connection issuing the command). This can be challenging in a production database.
- When READ_COMMITTED_SNAPSHOT is ON, row versioning may be bypassed by using the READCOMMITTEDLOCK table hint, in which case the table will not be row versioned for the purposes of the statement using the hint.

Transactions



A transaction is considered to be one or more Transact-SQL statements that are logically grouped into a single unit of work. A transaction might be made up of multiple statements; changes made to data by these statements are applied to the database only when the transaction completes successfully. A transaction must adhere to the ACID principles:

- **Atomicity**. A transaction must be atomic in nature; that is, either all of the changes are applied or none.
- **Consistency**. After completion, a transaction must leave all data and related structures in a consistent state.
- **Isolation**. A transaction must have a view of the data independent of any other concurrent transaction; a transaction should not see data in an intermediate state.
- **Durability**. Data changes must be persisted in case of a system failure.

SQL Server Transaction Management Modes

Auto-commit mode. Auto-commit is the default transaction management mode in SQL Server. A transaction is either committed or rolled back after completion. If a statement completes successfully without any error, it is committed. If it encounters errors, it is rolled back. Auto-commit mode is overridden when a user initiates an explicit transaction or when implicit transaction mode is enabled.

Explicit transaction mode. In explicit transaction mode, you explicitly define the start and end of a transaction.

- BEGIN TRANSACTION. Marks the start of a transaction.
- COMMIT TRANSACTION. Marks the successful completion of a transaction. The modifications made to a database are made permanent; the resources held by a transaction are released.
- ROLLBACK. Marks the unsuccessful termination of a transaction; the modifications made by a transaction are discarded; the resources held by a transaction are released.
- SAVE TRANSACTION. You can set a save point part-way through the transaction; you can use a ROLLBACK statement to revert to a save point, which means you can retry without having to start the entire transaction again.

When an explicit transaction completes, the connection returns to the transaction mode it was using before the start of the explicit transaction.

For more information on transaction control statements in explicit transaction mode, see the topic *Transaction Statements (Transact-SQL)* in Microsoft Docs:

Transaction Statements (Transact-SQL)

http://aka.ms/krt2iy

Implicit transaction mode. In implicit transaction mode, SQL Server automatically manages the start of a transaction. You can commit or roll back an implicit transaction but you cannot control the start of the transaction. SQL Server automatically starts a new implicit transaction after the current implicit transaction finishes, generating a continuous chain of transactions.

- Implicit transaction is a session level setting and can be changed by setting the IMPLICIT_TRANSACTION option to ON/OFF.
- SQL Server automatically starts an implicit transaction when any of the following statements are executed: ALTER TABLE, CREATE, DELETE, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE TABLE, UPDATE.

For more information about implicit transaction mode, see the topic *SET IMPLICIT_TRANSACTIONS (Transact-SQL)* in the SQL Server 2016 Technical Documentation:

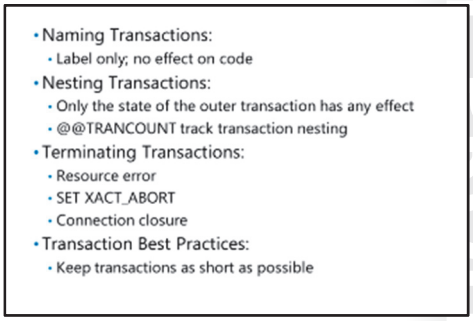
SET IMPLICIT_TRANSACTIONS (Transact-SQL)

http://aka.ms/vima93

Batch-scoped transaction mode. The batch-scoped transaction is applicable only to multiple active result sets (MARS). A transaction (whether implicit or explicit) that starts under MARS is converted to a batch-scoped transaction. A batch-scoped transaction that is neither committed nor rolled back on batch completion is automatically rolled back by SQL Server.

The transaction mode is set at the connection level. If a connection changes from one transaction mode to another, other active connections are not affected.

Working with Transactions



You should be aware of some features of transactions in SQL Server when you start to use them.

Naming Transactions

Transaction names are optional, and have no effect on the behavior of SQL Server; they act purely as labels to assist developers and DBAs in understanding Transact-SQL code.

Explicit transactions may be named.

Transaction Naming Example

```
BEGIN TRANSACTION my_tran_name;

COMMIT TRANSACTION my_tran_name;
```

Nesting Transactions

Explicit transactions can be nested; you can issue a BEGIN TRANSACTION command inside an open transaction. Only the outermost transaction has any effect; if the outermost transaction is committed, all the nested transactions are committed. If the outermost transaction is rolled back, all the nested transactions are rolled back. The level of transaction nesting is tracked by the @@TRANCOUNT function, which is maintained at connection level:

- Each BEGIN TRANSACTION statement increments @@TRANCOUNT by one.
- Each COMMIT statement decrements @@TRANCOUNT by one. The COMMIT that reduces @@TRANCOUNT to zero commits the outermost transaction.
- A ROLLBACK statement rolls back the outer transaction and reduces @@TRANCOUNT to zero.

Terminating Transactions

As well as an explicit COMMIT or ROLLBACK, a transaction can be terminated for the following reasons:

- **Resource error**. If a transaction fails because of a resource error (for example, lack of disk space), SQL Server automatically rolls back the transaction to maintain data integrity.
- **SET XACT_ABORT**. When the connection-level SET XACT_ABORT setting is ON, an open transaction is automatically rolled back in the event of a runtime error. When XACT_ABORT is OFF, a statement that causes an error is normally rolled back, but any open transaction will remain open.
- **Connection closure**. If a connection is closed, all open transactions are rolled back.

For more information on SET XACT_ABORT, see the topic *SET XACT_ABORT (Transact-SQL)* in Microsoft Docs:

SET XACT_ABORT (Transact-SQL)

http://aka.ms/nrph4c

Transaction Best Practices

- **Short transactions**. Keep transactions as short as possible. The shorter the transaction, the sooner the locks will be released. This will help reduce unnecessary blocking.
- **Avoid user input**. Avoid user interaction during a transaction. This might add unnecessary delay, because a user might open a transaction and go out for a break. The transaction will hold locks until the user returns to complete the transaction or the transaction is killed. Other transactions requiring locks on the same resource will be blocked during this time.
- **Open a transaction only when necessary**. If possible, avoid opening a transaction when browsing through data. Do the preliminary data analysis and then open a transaction to perform any necessary data modification.
- **Access only relevant data**. Access only the relevant data within a transaction. This reduces the number of locks, so reducing the blocking and deadlocks.
- **Use the appropriate transaction isolation level**. Not all applications require high level isolation level, such as repeatable read and serializable. Many applications work well with the default repeatable read isolation.
- **Beware of triggers containing transactions**. Triggers containing transactions should be written carefully. Issuing a ROLLBACK command within a trigger will roll back the whole transaction, of which the trigger is a part.

Demonstration: Analyzing Concurrency Problems

In this demonstration, you will see:

- Examples of concurrency problems.
- How changes to transaction isolation levels address concurrency problems.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Demofiles\Mod05** folder as Administrator. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.
3. Start SQL Server Management Studio and connect to your Azure instance of the **AdventureWorksLT** database engine instance using SQL Server authentication. If the **Microsoft SQL Server Management Studio** dialog box appears, click **OK**.
4. Open the **Demo1.ssmssln** solution in the **D:\Demofiles\Mod05\Demo1** folder.
5. Open the **Demo 1a - concurrency 1.sql** script file and the **Demo 1b - concurrency 2.sql** script file; open these files in different query windows, because you will be switching between them.
6. In both script files, click **ADVENTUREWORKSLT** in the **Available databases** list.
7. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 3**, select the code, and then click **Execute** to check the current settings for SNAPSHOT isolation.
8. Under the comment that begins **Step 4**, select the code, and then click **Execute** to view the current state of the row used in this demonstration.
9. In the **Demo 1b - concurrency 2.sql** script file, under the comment that begins **Query 1**, select the code, and then click **Execute**.
10. In the **Demo 1a - concurrency 1.sql** file, under the comment that begins **Step 5**, select the code, and then click **Execute** to demonstrate READ UNCOMMITTED isolation.
11. Under the comment that begins **Step 6**, select the code, and then click **Execute** to demonstrate READ COMMITTED isolation. The query will wait until you complete the next step.
12. In the **Demo 1b - concurrency 2.sql** file, under the comment that begins **Query 2**, select the code, and then click **Execute**.
13. In the **Demo 1a - concurrency 1.sql** script file, note that the query under **Step 6** (which was already running) has now returned a result.
14. Under the comment that begins **Step 7**, select the first five lines of code, and then click **Execute**.
15. In the **Demo 1b - concurrency 2.sql** file, under the comment that begins **Query 3**, select the code, and then click **Execute**.
16. In the **Demo 1a - concurrency 1.sql** file, under the comment that begins **Step 7**, select the final four lines of code, and then click **Execute** to demonstrate a non-repeatable read.
17. Under the comment that begins **Step 8**, select the first six lines of code, and then click **Execute**.
18. In the **Demo 1b - concurrency 2.sql** file, under the comment that begins **Query 4**, select the code, and then click **Execute**. Note that this query will not return until you complete the next step, because REPEATABLE READ holds a lock on the affected row.

19. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 8**, select the final four lines of code, and then click **Execute** to demonstrate that REPEATABLE READ isolation prevents a non-repeatable read.
20. Under the comment that begins **Step 9**, select the first six lines of code, and then click **Execute**.
21. In the **Demo 1b - concurrency 2.sql** script file, under the comment that begins **Query 5**, select the code, and then click **Execute**.
22. In the **Demo 1a - concurrency 1.sql** file, under the comment that begins **Step 9**, select the final four lines of code, and then click **Execute** to demonstrate that READ COMMITTED isolation allows a phantom read.
23. Under the comment that begins **Step 10**, select the first six lines of code, and then click **Execute**.
24. In the **Demo 1b - concurrency 2.sql** script file under the comment that begins **Query 5**, select the code, and then click **Execute**. Note that this query will not return until you complete the next step, since SERIALIZABLE holds a lock on the affected table.
25. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 10**, select the final four lines of code, and then click **Execute** to demonstrate that SERIALIZABLE isolation prevents a phantom read.
26. Under the comment that begins **Step 11**, select the first two lines of code, and then click **Execute** to reset the value of the target data row.
27. In the **Demo 1b - concurrency 2.sql** file, under the comment that begins **Query 6**, select the code, and then click **Execute**.
28. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 11**, select the six lines of code under the comment that begins **run the following statements**, and then click **Execute**. Note that the committed value of the row is returned.
29. In the **Demo 1b - concurrency 2.sql** file, under the comment that begins **Query 7**, select the code, and then click **Execute**.
30. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 11**, select the final four lines of code, and then click **Execute** to demonstrate the behavior of READ_COMMITTED_SNAPSHOT ON.
31. Under the comment that begins **Step 12**, select the first two lines of code, and then click **Execute** to reset the value of the target data row.
32. In the **Demo 1b - concurrency 2.sql** script file, under the comment that begins **Query 6**, select the code, and then click **Execute**.
33. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 12**, select the six lines of code under the comment that begins **run the following statements**, and then click **Execute**. Note that the committed value of the row is returned.
34. In the **Demo 1b - concurrency 2.sql** script file, under the comment that begins **Query 7**, select the code, and then click **Execute**.
35. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 12**, select the final four lines of code, and then click **Execute** to demonstrate the behavior of SNAPSHOT isolation.
36. Under the comment that begins **Step 13**, select the first two lines of code, and then click **Execute** to reset the value of the target data row.
37. In the **Demo 1b - concurrency 2.sql** script file, under the comment that begins **Query 6**, select the code, and then click **Execute**.

38. In the **Demo 1a - concurrency 1.sql** script file, under the comment that begins **Step 13**, select the six lines of code under the comment that begins **run the following statements**, and then click **Execute**. Note that this query will not return until you complete the next step.
39. In the **Demo 1b - concurrency 2.sql** script file, under the comment that begins **Query 7**, select the code, and then click **Execute**.
40. In the **Demo 1a - concurrency 1.sql** query window, notice that an update conflict in SNAPSHOT isolation mode has been reported.
41. Under the comment that begins **Step 13**, select the final line of code, and then click **Execute** to demonstrate that the open transaction was rolled back when the error was raised.
42. Close SQL Server Management Studio without saving any changes.

Check Your Knowledge

Question	
User A starts to update a customer record, and while the transaction is still in progress, User B tries to update the same record. User A's update completes successfully, but User B's update fails with an error message: "This customer's record has been updated by another user". Which concurrency model is the system using?	
Select the correct answer.	
	Pessimistic concurrency
	Optimistic concurrency

Lesson 2
Locking Internals

SQL Server uses locks to ensure the consistency of data during a transaction. This lesson discusses the details of the locking architecture used by SQL Server, how locks are used during the life of a transaction, and the various methods available to you to influence the default locking behavior.

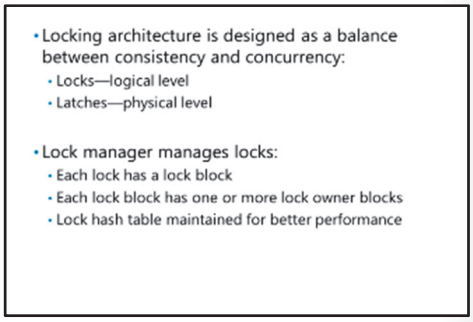
Lesson Objectives

At the end of this lesson, you will be able to:

- Describe the SQL Server locking architecture.
- Describe lock hierarchy and lock granularity.
- Explain lock escalation.
- Understand lock modes.
- Explain lock compatibility.
- Explain the data modification process.
- Use locking hints.
- Understand deadlocks.
- Explain latches and spinlocks.

Locking Architecture

In a hypothetical database system, the least sophisticated locking architecture possible is to allow locks only at database level. Every user reading or writing data would lock the entire database, preventing access by any other user until the change was complete. While this approach ensures data is consistent, it prohibits concurrent database activity.



SQL Server's locking system is designed to ensure data consistency while still allowing concurrent activity. Locks are acquired at an appropriate level of granularity to protect the data that is modified by a transaction; locks are held until the transaction commits or rolls back. Different objects affected by a transaction can be locked with different types of lock.

Locks and Latches

SQL Server implements two locking systems. The first system manages locks for database objects (tables, indexes, and so on) that are accessible directly to users; these locks act at a logical level to ensure data consistency. This locking system is managed by the lock manager. The second system is used to ensure the physical consistency of data in memory; for this process, a lightweight locking mechanism, known as a latch, is employed. This system is managed by the storage engine.

Lock Manager

Internally, locks are automatically managed by the lock manager, a component of the database engine. When the database engine processes a Transact-SQL statement, the Query Processor subcomponent determines the resources that will be accessed. The Query Processor also determines the type of locks to acquire, based on the type of data access (read or write) and the transaction isolation level setting. The Query Processor then requests these locks from the lock manager. The lock manager grants the locks if no conflicting locks are being held by other transactions.

Locks are in-memory structures; the lock manager maintains a memory structure for each lock, called a *lock block*, which records the lock type and the resource that is locked. Each lock block will be linked to one or more *lock owner blocks*; the lock owner block links a lock to the process requesting the lock. The lock manager also maintains a lock hash table, to track locked resources more efficiently.

When a SQL Server instance starts, lock manager acquires sufficient memory to support 2,500 locks. If the total number of locks exceeds 2,500, more memory is allocated dynamically to lock manager.

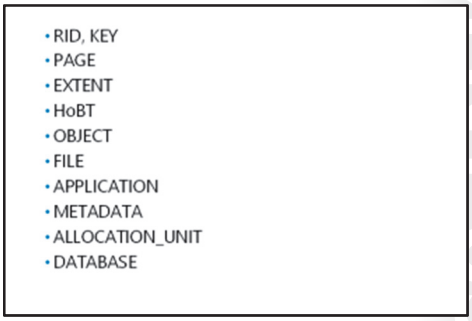
For more information on locking in SQL Server, see the topic *SQL Server Transaction Locking and Row Versioning Guide* on MSDN:

SQL Server Transaction Locking and Row Versioning Guide

http://aka.ms/mc5pmh

Lock Granularity and Hierarchy

Database objects and resources can be locked at different levels of granularity; to allow more concurrent activity, SQL Server will attempt to lock as few resources as possible to efficiently process a Transact-SQL statement. The efficiency of the locking strategy is determined by comparing the overhead of maintaining many locks at a fine grain against the increase in concurrency from lower-grained locking. Locking at higher granularity levels—such as at table level—decreases concurrency because the entire table is inaccessible to other transactions. However, the overhead is less, as fewer locks are to be maintained.



SQL Server acquires locks at any of the following levels, ordered here from lowest grain to highest grain. The first two items (RID and KEY) are of equivalent grain:

- **RID**. RID stands for row id. A row id is a lock on a single row in a heap (table without clustered index).
- **KEY**. A key lock applies to a single row in a clustered or nonclustered index.
- **PAGE**. A lock on an 8 KB page in a database, such as a data or index page. If a page is locked, all of the data rows contained in the page are locked.
- **EXTENT**. A lock on a 64 KB extent (a block of eight pages). If an extent is locked, all of the pages in the extent are locked.
- **HoBT**. HoBT stands for heap or b-tree. A lock protecting a b-tree (index), or the heap data pages in a table that does not have a clustered index. All the extents that make up the heap or b-tree are locked.

- **OBJECT**. Typically, a lock on a table. If a table is locked, all of the associated data and index pages are also locked.
- **FILE**. A lock on a database file. If a file is locked, all of the objects it contains are locked.
- **APPLICATION**. An application-specified lock, created using **sp_getapplock**.
- **METADATA**. A lock on catalog views.
- **ALLOCATION_UNIT**. A lock on an allocation unit such as IN_ROW_DATA.
- **DATABASE**. A lock on an entire database. All the objects in the database are also locked.

The objects in this list make up a hierarchy; databases are composed of files, files contain tables, and tables are in turn made up of extents, pages, and rows. To fully protect a resource during the processing of a command, a process might acquire locks at multiple levels of the resource hierarchy. For example, when processing a command that affects a single row in a table, locks might be acquired on the affected row, the page in which the row is stored, the page's extent, and the table to which the row belongs. This both fully protects the table and simplifies the detection of locking conflicts with other concurrent processes that may hold locks on different rows in the same table.

Lock Escalation

Lock escalation occurs when many fine-grained locks held by a transaction on a single resource are converted to a single coarser-grained lock on the same resource. Lock escalation is used to limit the total number of locks the lock manager must manage; the cost being that lock escalation might reduce concurrency.

- Reduces lock manager memory overhead by converting many fine-grained locks to a single coarser-grained lock
  - Row and page locks escalate to table locks
- Control at table level with ALTER TABLE SET LOCK_ESCALATION
- Control at session or instance level with trace flags 1224 and 1211

When lock escalation from row locks occurs, the lock is always escalated to table level; lock escalation does not take place from row level to page level.

In previous versions of SQL Server, the default conditions for lock escalation were hard-coded; when a transaction held more than a fixed number of row level or page level locks on a resource, the lock would be escalated. This is no longer true. Lock escalation decisions are based on multiple factors; there is no fixed threshold for lock escalation.

Lock escalation can also occur when the memory structures maintained by the lock manager consume more than 40 percent of the available memory.

You can control lock escalation behavior for individual tables by using the ALTER TABLE SET LOCK_ESCALATION command. LOCK_ESCALATION can be set to one of three values:

- TABLE. The default value. When lock escalation occurs, locks are always escalated to table level whether or not the table is partitioned.
- AUTO. If the table is partitioned when lock escalation occurs, locks can be escalated to partition level. If the table is not partitioned, locks are escalated to table level.
- DISABLE. Prevents lock escalation occurring in most cases. Table locks might still occur, but will be less frequent.

For more information on controlling lock escalation behavior, see the topic *ALTER TABLE (Transact-SQL)* in Microsoft Docs:

ALTER TABLE (Transact-SQL)

http://aka.ms/hb1ub7

Lock escalation behavior can also be controlled at session level or instance level by use of trace flags:

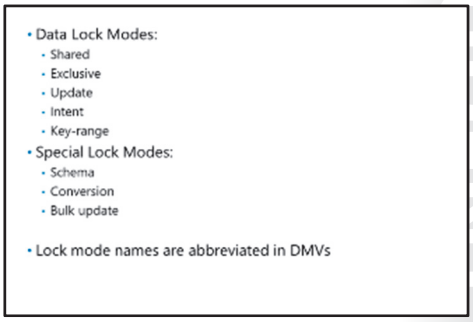
- Trace flag 1224 disables lock escalation, based on the number of locks held on a resource. Lock escalation due to memory pressure can still occur.
- Trace flag 1211 disables lock escalation completely, whether due to the number of locks held on a resource or due to memory pressure. Disabling lock escalation can have a severe effect on performance and is not recommended.

For more information on trace flags 1224 and 1211, see the topic *Trace Flags (Transact-SQL)* in Microsoft Docs:

Trace Flags (Transact-SQL)

http://aka.ms/hvmsq7

Lock Modes



SQL Server locks resources using different lock modes. The lock modes determine how accessible a resource is to other concurrent transactions.

Data Lock Modes

The following lock modes are used to lock resources:

- **Shared lock**. Shared locks are acquired when reading data. The duration for which a shared lock is held depends on transaction isolation level or locking hints. Many concurrent transactions may hold shared locks on the same data. No other transaction can modify the data until the shared lock is released.
- **Exclusive lock**. Exclusive locks are acquired when data is modified (by an INSERT, UPDATE, or DELETE statement). Exclusive locks are always held until the end of the transaction. Only one transaction may acquire an exclusive lock on a data item at a time; while an exclusive lock is held on a data item, no other type of lock may be acquired on that data item.
- **Update lock**. Update locks are acquired when modifying data and are a combination of shared and exclusive locks. Update locks are held during the searching phase of the update, where the rows to be modified are identified; they are converted to exclusive locks when actual modification takes place. Only one transaction may acquire an update lock on a data item at one time; other transactions might hold or acquire shared locks on the same data item while an update lock is in place.
- **Intent lock**. An intent lock is not a locking mode in its own right—it acts as a qualifier to other lock modes. Intent locks are used on a data item to indicate that a subcomponent of the data item is locked; for instance, if a row in a table is locked with a shared lock, the table to which the row belongs would be locked with an intent shared lock. Intent locks are discussed in more detail in the next topic.
- **Key-range locks.** Key-range locks are used by transactions using the SERIALIZABLE isolation level to lock ranges of rows that are implicitly read by the transaction; they protect against phantom reads.

Special Lock Modes

Special lock modes are used to control stability of the database schema, when locks are converted between modes, and during bulk update operations:

- **Schema lock**. Schema locks are used when an operation dependent on the table schema is executed. There are two types of schema lock:
  - **Schema modification lock**. Schema modification locks are acquired when a data definition language (DDL) operation is being performed against a table, such as adding or dropping a column.
  - **Schema stability lock**. Schema stability locks are used during query compilation to prevent transactions that modify the underlying database schema. Schema stability locks are compatible with all other lock types (including exclusive locks).
- **Conversion lock**. A conversion lock is a specialized type of intent lock used to manage the transition between data lock modes. Conversion locks appear in three types:
  - **Shared with intent exclusive**. Used when a transaction holds a mixture of shared locks and exclusive locks on subobjects of the locked object.
  - **Shared with intent update**. Used when a transaction holds a mixture of shared locks and update locks on subobjects of the locked object.
  - **Update with intent exclusive**. Used when a transaction holds a mixture of exclusive locks and update locks on subobjects of the locked object.
- **Bulk update lock**. Bulk update locks can optionally be acquired when data is bulk inserted into a table using a bulk command such as BULK INSERT. A bulk update lock can only be acquired if no other incompatible lock types are held on the table.

Locks held by active transactions can be viewed by using the **sys.dm_tran_locks** dynamic management view (DMV). DMVs use abbreviations for lock mode names, summarized in the following table:

Abbreviation	Lock Mode
S	Shared
U	Update
X	Exclusive
IS	Intent Shared
IU	Intent Update
IX	Intent Exclusive
RangeS_S	Shared Key-Range and Shared Resource lock
RangeS_U	Shared Key-Range and Update Resource lock
RangeI_N	Insert Key-Range and Null Resource lock
RangeI_S	Key-Range Conversion lock
RangeI_U	Key-Range Conversion lock
RangeI_X	Key-Range Conversion lock
RangeX_S	Key-Range Conversion lock
RangeX_U	Key-Range Conversion lock
RangeX_X	Exclusive Key-Range and Exclusive Resource lock
Sch-S	Schema stability
Sch-M	Schema modification
SIU	Shared Intent Update
SIX	Shared Intent Exclusive
UIX	Update Intent Exclusive
BU	Bulk Update

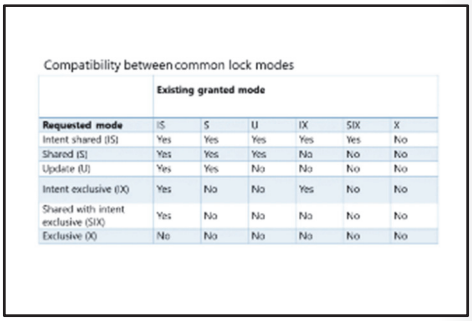
For more information on the **sys.dm_tran_locks** DMV, see the topic *sys.dm_tran_locks (Transact-SQL)* in the SQL Server 2016 Technical Documentation:

sys.dm_tran_locks (Transact-SQL)

http://aka.ms/jf98cd

Lock Mode Compatibility

Processes may acquire locks of different modes. Two lock modes are said to be compatible if a process can acquire a lock mode on a resource when another concurrent process already has a lock on the same resource. If a process attempts to acquire a lock mode that is incompatible with the mode of an existing lock, the process must wait for the existing lock to be released before acquiring the new lock.


Compatibility between common lock modes

	Existing granted mode					
Requested mode	IS	S	U	IX	SIX	X
Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No	No
Update (U)	Yes	Yes	No	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No	No
Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
Exclusive (X)	No	No	No	No	No	No

SQL Server uses lock compatibility to ensure transactional consistency and isolation, while still permitting concurrent activity; it allows processes that read data to run concurrently, while ensuring that modification of a resource can only be carried out by one process at a time.

Note: Lock compatibility gives an insight into the differences in behavior between the different isolation levels you learned about in the previous topic; the more pessimistic isolation levels acquire and hold locks that are less compatible with other lock types.

When processes wait for incompatible lock types to be released, they wait in a first-in, first-out queue. If there are already processes queuing for a resource, a process seeking to acquire a lock on the same resource must join the end of the queue, even if the mode of the lock it seeks to acquire is compatible with the current lock on the resource. On busy resources, this prevents processes seeking less compatible lock modes from waiting indefinitely when other, more compatible lock modes are in use.

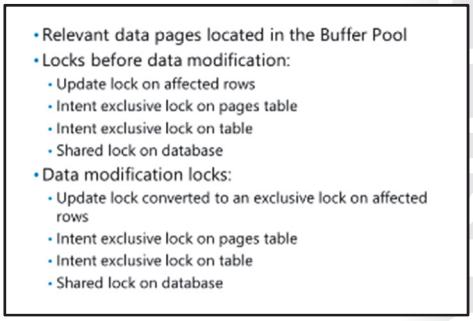
Note: When a process is waiting for an incompatible lock on a resource to be released, it is said to be blocked. Because of the way processes queue when waiting for locks, chains of blocked processes can develop, slowing—or potentially stopping—system activity.

For a full lock compatibility matrix, see the topic *Lock Compatibility (Database Engine)* on Microsoft TechNet (note that this page comes from the SQL Server 2008 R2 documentation; Microsoft has not published a version of this matrix for SQL Server 2016):

Lock Compatibility (Database Engine)

http://aka.ms/t0ia22

The Data Modification Process



To understand how locks are used as Transact-SQL statements are processed, consider the example of a statement that modifies data in an UPDATE statement.

The following UPDATE query could be run in the AdventureWorks database:

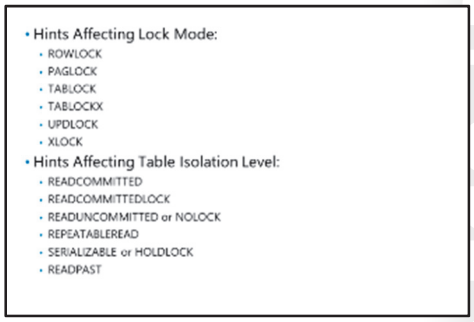
Example UPDATE Statement Changing Two Rows

```
UPDATE HumanResources.Employee
SET MaritalStatus = 'S'
WHERE BusinessEntityId IN (3,289);
```

The following steps are involved when modifying data in SQL Server:

- A user or an application sends the UPDATE query to SQL Server.
- The database engine receives the update request and locates the data pages to be updated in the cache—or reads the data pages from the storage subsystem into cache.
- The database engine tries to grant the lock on the necessary data to the user's session:
  - If any transaction already has an incompatible lock on the affected data, the UPDATE query waits for the existing lock to be released.
  - Because this UPDATE statement is highly selective (affecting only two rows) the database engine uses row level locking to acquire an update lock on each of the two rows being modified.
- The following additional locks are acquired to secure the pages and the table in question:
  - Two intent-exclusive page level locks (one for each page, since the rows are in different pages).
  - One intent-exclusive table level lock.
  - A shared database level lock.
  - Additional locks may be required if the data being modified makes up an index. In this example, no indexes are affected.
- SQL Server starts the data modification. The steps are as follows:
  - The data modification is made (in the cache). At the same time, the update lock is converted to an exclusive lock.
  - The changes are logged in transaction log pages (in cache).
  - The locks are released.
  - The transaction is committed.
- Acknowledgement is sent to the user or application.

Locking Hints



There might be instances where you need to influence locking behavior; several table hints are available that help you adjust the locking of individual tables during a single Transact-SQL statement. You can use hints to influence:

- The mode of any locks acquired on a table.
- The transaction isolation level applied to a table.

Table hints are applied by including a WITH command after the name of the table for which you want to influence locking in the FROM clause of a Transact-SQL statement.

Table Hint Example

```
…
FROM <table name> WITH (<table hint> [,<table hint>])
```

You can specify multiple hints for a table—the hints should be comma-separated in the brackets of the WITH command.

Best Practice: In general, it is best to avoid locking hints and allow the SQL Server Query Optimizer to select an appropriate locking strategy. Be sure to regularly review any locking hints you use; confirm that they are still appropriate.

Hints Affecting Lock Mode

The following hints affect the lock mode acquired by a Transact-SQL statement:

- ROWLOCK. Row locks should be acquired where page or table locks would normally be used.
- PAGLOCK. Page locks should be acquired where row or table locks would normally be used.
- TABLOCK. A table lock should be acquired where row or page locks would normally be used.
- TABLOCKX. An exclusive table lock should be acquired.
- UPDLOCK. An update lock should be acquired.
- XLOCK. An exclusive lock should be acquired.

Hints Affecting Table Isolation Level

The following hints affect the isolation level used by a Transact-SQL statement:

- READCOMMITTED. Use the READ COMMITTED isolation level. Locks or row versioning are used, depending on the value of the READ_COMMITTED_SNAPSHOT database setting.
- READCOMMITTEDLOCK. Use the READ COMMITTED isolation level, acquiring locks. The value of the READ_COMMITTED_SNAPSHOT database setting is ignored.
- READUNCOMMITTED or NOLOCK. Use the READ UNCOMMITTED isolation level. Both READUNCOMMITTED and NOLOCK hints have the same effect.
- REPEATABLEREAD. Use the REPEATABLE READ isolation level.
- SERIALIZABLE or HOLDLOCK. Use the SERIALIZABLE isolation level. Both SERIALIZABLE and HOLDLOCK hints have the same effect.
- READPAST. Rows that are locked by other transactions will be ignored, instead of waiting for incompatible locks to be released,

For more information on table hints—including those that control locking—see the topic *Table Hints (Transact-SQL)* in the SQL Server 2016 Technical Documentation:

Table Hints (Transact-SQL)

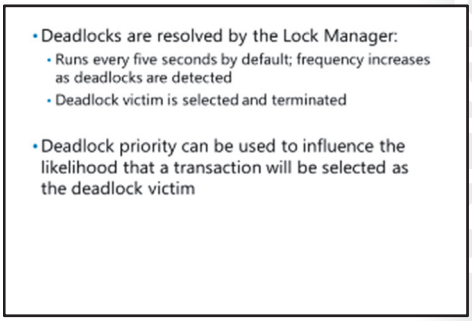
http://aka.ms/fkaztl

Some best practices when using locking hints are:

- Use the TABLOCK hint to speed up bulk insert operations. TABLOCK is only compatible with itself. This allows multiple bulk insert to be made in parallel into a single table, while preventing other processes to update or modify the records. This considerably improves bulk insert performance.
- Avoid using the NOLOCK or READUNCOMMITTED hint to resolve reader-writer blocking; consider setting READ_COMMITTED_SNAPSHOT to ON or using the SNAPSHOT isolation level. The NOLOCK and READUNCOMMITTED hints are only suitable in environments where the effects of the READ UNCOMMITTED isolation level (documented in the previous lesson) are acceptable.
- Use ROWLOCK or UPDLOCK hints to reduce deadlocks in the REPEATABLE READ isolation level.

Deadlock Internals

A deadlock occurs when two or more transactions block one another by attempting to acquire a lock on a resource that is already locked by the other transaction(s) with an incompatible lock mode. For example:



- Transaction A acquires a shared lock on table T1.
- Transaction B acquires a shared lock on table T2.
- Transaction A requests an exclusive lock on table T2. It waits on transaction B to release the shared lock it holds on table T2.
- Transaction B requests an exclusive lock on table T1. It waits on transaction A to release the shared lock it holds on Table T1. A deadlock has occurred.

Without intervention, a deadlock will continue indefinitely.

Deadlock Resolution

The Lock Monitor process is responsible for detecting deadlocks. It periodically searches for the tasks involved in a deadlock. The search process has the following properties:

- The default interval between deadlock detection searches for five seconds.
- As soon as a deadlock is found, the deadlock detection search will run again immediately.
- When deadlocks are detected, the deadlock detection search interval is reduced to as little as 100 milliseconds, depending on the deadlock frequency.

When a deadlock is detected, the Lock Monitor ends the deadlock by choosing one of the thread as the deadlock victim. The deadlock victim command is forcefully terminated; the transaction is rolled back, and the error 1205 is returned to the application. This releases the locks held by the deadlock victim, allowing the other transactions to continue with their work.

The deadlock victim is selected based on the following rules:

- If all the deadlocked transactions have the same deadlock priority, the transaction that is estimated to be the least expensive to roll back is chosen as the deadlock victim.
- If the deadlocked transactions have a different deadlock priority, the transaction with the lowest deadlock priority is chosen as the deadlock victim.

Deadlock Priority

You can specify the deadlock priority of a transaction by using the SET DEADLOCK_PRIORITY command. You can set deadlock priority to an integer value between -10 (the lowest priority) and 10 (the highest priority)—or you can use a text value:

- **LOW**. Equivalent to the integer value -5.
- **NORMAL**. Equivalent to the integer value 0.
- **HIGH**. Equivalent to the integer value 5.

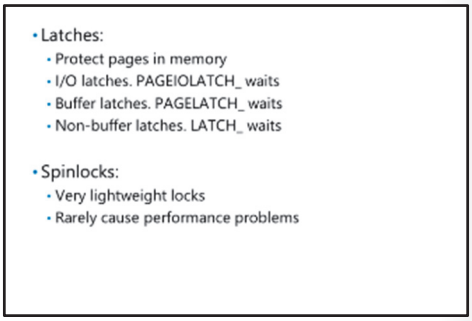
For more information on setting deadlock priority, see the topic *SET DEADLOCK_PRIORITY (Transact-SQL)* in the SQL Server 2016 Technical Documentation:

SET DEADLOCK_PRIORITY (Transact-SQL)

http://aka.ms/vaffc7

Latches and Spinlocks

Some database engine operations avoid the cost of managing locks by using lighter-weight locking mechanisms, latches, and spinlocks.



Latches

Latches are a lightweight locking mechanism used by the storage engine to ensure the consistency of in-memory data structures, such as data pages and non-leaf pages in a b-tree. Latches are managed internally by SQL Server and cannot be controlled by users. Latches are broadly divided into three types:

- **I/O latches**. Used to manage outstanding I/O operations against pages in the Buffer Pool, I/O latches ensure that pages are read only once from I/O into the Buffer Pool.
- **Buffer latches**. Used to prevent concurrent processes from making conflicting changes to pages in the Buffer Pool.
- **Non-buffer latches**. Used to protect shared data structures held outside the Buffer Pool.

When a process waits for a latch, the duration of the wait is recorded in the **sys.dm_os_wait_stats** DMV:

- I/O latches appear as wait types with names starting PAGEIOLATCH_.
- Buffer latches appear as wait types with names starting PAGELATCH_.
- Non-buffer latches are summarized as wait types with names starting LATCH_. A complete list of all non-buffer latch types can be found in the **sys.dm_os_latch_stats** DMV.

For more information on the **sys.dm_os_wait_stats** DMV, see the topic *sys.dm_os_wait_stats (Transact-SQL)* in the SQL Server 2016 Technical Documentation:

sys.dm_os_wait_stats (Transact-SQL)

http://aka.ms/kvkoru

For more information on the **sys.dm_os_latch_stats** DMV, see the topic *sys.dm_os_latch_stats (Transact-SQL)* in the SQL Server 2016 Technical Documentation:

sys.dm_os_latch_stats (Transact-SQL)

http://aka.ms/im1px3

Spinlocks

Spinlocks are very lightweight locking structures used when a process needs to lock an object in memory for a very short time. A process waiting to acquire a spinlock will go into a loop for a period, checking repeatedly whether the lock is available—as opposed to moving onto a waiter list and yielding the CPU immediately. SQL Server uses spinlocks to protect objects such as hash buckets in the lock manager's lock hash table.

Some contention for spinlocks is expected on busy SQL Server instances; spinlock contention should only be considered a problem when it causes significant CPU overhead. Performance problems can be caused by contention for spinlocks, but this is a relatively rare occurrence.

For more information on diagnosing and resolving performance problems caused by spinlock contention, see the Microsoft paper *Diagnosing and Resolving Spinlock Contention on SQL Server*. Note that this paper was written in reference to SQL Server 2008 R2:

Diagnosing and Resolving Spinlock Contention on SQL Server

http://aka.ms/uvpmoe

Demonstration: Applying Locking Hints

In this demonstration, you will see the effects of several locking hints.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd.**
2. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
3. Open the **Demo2.ssmssln** solution in the D:\Demofiles\Mod05\Demo2 folder.

4. Open the **Demo 2a - lock hints 1.sql** script file and the **Demo 2b - lock hints 2.sql** script file. Ensure that both scripts use the **AdventureWorks** database.
5. In the **Demo 2a - lock hints 1.sql** script file, under the comment that begins **Step 3**, select the code, and then click **Execute** to show the current isolation level.
6. Under the comment that begins **Step 4**, select the code, and then click **Execute** to demonstrate the locks held by a transaction using READ UNCOMMITTED isolation.
7. Under the comment that begins **Step 5**, select the first three lines of code, and then click **Execute** to demonstrate the locks held by a transaction using REPEATABLE READ isolation.
8. Under the comment that begins **Step 5**, select the remaining five lines of code, and then click **Execute**.
9. Under the comment that begins **Step 6**, select the first three lines of code, and then click **Execute** to demonstrate the locks held by a transaction using REPEATABLE READ isolation and a READCOMMITTED locking hint.
10. Under the comment that begins **Step 6**, select the remaining five lines of code, and then click **Execute**.
11. Under the comment that begins **Step 7**, select the first three lines of code, and then click **Execute** to demonstrate the locks held by a transaction using READ COMMITTED isolation and a TABLOCKX locking hint.
12. Under the comment that begins **Step 7**, select the remaining five lines of code, and then click **Execute**.
13. Under the comment that begins **Step 8**, select the first three lines of code, and then click **Execute** to demonstrate the locks held by a transaction using REPEATABLE READ isolation and a TABLOCKX locking hint.
14. Under the comment that begins **Step 8**, select the remaining five lines of code, and then click **Execute**.
15. In the **Demo 2b - concurrency 2.sql** script file, under the comment that begins **Query 1**, select the code, and then click **Execute**.
16. In the **Demo 2a - lock hints 1.sql** script file, under the comment that begins **Step 9**, select the code, and then click **Execute** to demonstrate that the statement waits.
17. Allow the query to wait for a few seconds, and then on the **Query** menu, click **Cancel Executing Query**.
18. Under the comment that begins **Step 10**, select the code, and then click **Execute** to demonstrate the behavior of the READPAST hint.
19. In the **Demo 1b - concurrency 2.sql** script file, under the comment that begins **Query 2**, select the code, and then click **Execute** to close the open transaction.
20. Close SSMS without saving any changes.

Check Your Knowledge

Question	
If a process is attempting to acquire an exclusive row lock, what lock mode will it attempt to acquire on the data page and table that contain the row?	
Select the correct answer.	
	Exclusive (X)
	Shared (S)
	Intent shared (IS)
	Intent exclusive (IX)
	Intent update (IU)

Lab: Concurrency and Transactions

Scenario

You have reviewed statistics for the **AdventureWorks** database and noticed high wait stats for CPU, memory, IO, blocking, and latching. In this lab, you will address blocking wait stats. You will explore workloads that can benefit from snapshot isolation and partition level locking. You will then implement snapshot isolation and partition level locking to reduce overall blocking.

Objectives

After completing this lab, you will be able to:

- Implement the SNAPSHOT isolation level.
- Implement partition level locking.

Estimated Time: 45 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Implement Snapshot Isolation

Scenario

You have reviewed wait statistics for the **AdventureWorks** database and noticed high wait stats for locking, amongst others. In this exercise, you will implement SNAPSHOT Isolation to reduce blocking scenarios.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Clear Wait Statistics
3. Run the Workload
4. Capture Lock Wait Statistics
5. Enable SNAPSHOT Isolation
6. Implement Snapshot Isolation
7. Rerun the Workload
8. Capture New Lock Wait Statistics
9. Compare Overall Lock Wait Time

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab05\Starter** folder as Administrator.

▶ Task 2: Clear Wait Statistics

1. Start **SQL Server Management Studio** and connect to the MIA-SQL instance using Windows authentication; then open the project file **D:\Labfiles\Lab05\Starter\Project\Project.ssmssln** and the script file **Lab Exercise 01 - snapshot isolation.sql**.
2. Execute the query under the comment that begins **Task 1** to clear wait statistics.

▶ Task 3: Run the Workload

- In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell™. Wait for the workload to finish before continuing. If a message is displayed asking you to confirm a change in execution policy, type **Y**.

▶ Task 4: Capture Lock Wait Statistics

- In SSMS, amend the query under the comment that begins **Task 3** to capture only lock wait statistics into a temporary table. Hint: lock wait statistics have a **wait_type** that begins "LCK".

▶ Task 5: Enable SNAPSHOT Isolation

- Amend the properties of the **AdventureWorks** database to allow SNAPSHOT isolation.

▶ Task 6: Implement Snapshot Isolation

1. In SSMS Solution Explorer, open the script file **Lab Exercise 01 – stored procedure.sql**.
2. Use the script to modify the stored procedure definition to run under SNAPSHOT isolation.

▶ Task 7: Rerun the Workload

1. In the SSMS query window for **Lab Exercise 01 - snapshot isolation.sql,** rerun the query under the comment that begins **Task 1**.
2. In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell. Wait for the workload to finish before continuing.

▶ Task 8: Capture New Lock Wait Statistics

- In SSMS, under the comment that begins **Task 8**, amend the query to capture lock wait statistics into a temporary table called **#task8**.

▶ Task 9: Compare Overall Lock Wait Time

- In the SSMS query window for **Lab Exercise 01 - snapshot isolation.sql**, execute the query under the comment that begins **Task 9**, to compare the total **wait_time_ms** you have captured between the **#task3** and **#task8** temporary tables.

Results: After this exercise, the **AdventureWorks** database will be configured to use the SNAPSHOT isolation level.

Exercise 2: Implement Partition Level Locking

Scenario

You have reviewed statistics for the **AdventureWorks** database and noticed high wait stats for locking, amongst others. In this exercise, you will implement partition level locking to reduce blocking.

The main tasks for this exercise are as follows:

1. Open Activity Monitor
2. Clear Wait Statistics
3. View Lock Waits in Activity Monitor
4. Enable Partition Level Locking
5. Rerun the Workload

▶ Task 1: Open Activity Monitor

1. In SSMS Object Explorer, open Activity Monitor for the **MIA-SQL** instance.
2. In Activity Monitor, expand the **Resource Waits** section.

▶ Task 2: Clear Wait Statistics

1. If it is not already open, open the project file **D:\Labfiles\Lab05\Starter\Project\Project.ssmssln**, then open the query file **Lab Exercise 02 - partition isolation.sql**.
2. Execute the code under **Task 2** to clear wait statistics.

▶ Task 3: View Lock Waits in Activity Monitor

1. In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait for the workload to finish before continuing (it will take a few minutes to complete).
2. Switch to SSMS and to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
3. Close the PowerShell window where the workload was executed.

▶ Task 4: Enable Partition Level Locking

1. Return to the query window where **Lab Exercise 02 - partition isolation.sql** is open.
2. Under the comment that begins **Task 5**, write a query to alter the **Proseware.CampaignResponsePartitioned** table in the **AdventureWorks** database to enable partition level locking.
3. Rerun the query under the comment that begins **Task 2** to clear wait statistics.

▶ Task 5: Rerun the Workload

1. In the **D:\Labfiles\Lab05\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait for the workload to finish before continuing (it will take a few minutes to complete).
2. Return to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec**) for the **Lock** wait type.
3. Compare this value to the value you noted earlier in the exercise.
4. Close the PowerShell window where the workload was executed.

Results: After this exercise, the **AdventureWorks** database will use partition level locking.

Check Your Knowledge

Question	
When partition level locking is enabled, what combination of locks will be held by an UPDATE statement that updates all the rows in a single partition? Assume that the partition contains more than 1 million rows.	
Select the correct answer.	
	Database: Shared (S) Table: Exclusive (X)
	Database: Shared (S) Table: Intent Exclusive (IX) Partition: Exclusive (X)
	Database: Shared (S) Table: Exclusive (X) Partition: Exclusive (X)

Module Review and Takeaways

In this module, you have learned about SQL Server's implementation of transactions and concurrency. You have learned how to use transaction isolation levels to control data consistency within a transaction, and the concurrency issues you may expect at each isolation level. You have also learned about how locking is used to implement transaction isolation levels, and how to use lock hints to modify locking behavior.

Review Question(s)

Check Your Knowledge

Question	
A transaction is running with the SERIALIZABLE transaction isolation level. The transaction includes a SELECT statement with a single table in the FROM clause; the table is referenced with the READCOMMITTED table hint. Which transaction isolation level applies to the SELECT statement?	
Select the correct answer.	
	SERIALIZABLE
	READ UNCOMMITTED
	REPEATABLE READ
	READ COMMITTED

Module 6

Statistics and Index Internals

Contents:

Module Overview

This module covers the internals of statistics and indexes in Microsoft® SQL Server®. Creating and maintaining statistics and indexes is a core element in enhancing the performance of the SQL Server Database Engine. Both statistics and indexes are used to select suitable query execution plans; indexes also speed up the retrieval of data. To understand and influence query performance, you should have a good understanding of queries and indexes.

Objectives

After completing this module, you will be able to:

- Analyze statistics internals.
- Analyze index internals.
- Describe columnstore indexes.

Lesson 1
Statistics Internals and Cardinality Estimation

This lesson describes statistics internals and cardinality estimation. SQL Server uses statistics to produce optimal execution plans. Stale or missing statistics can lead to poor cardinality estimation, which can then lead to poor query performance. A thorough understanding of selectivity and statistics internals is critical to optimize queries by creating and maintaining the required statistics.

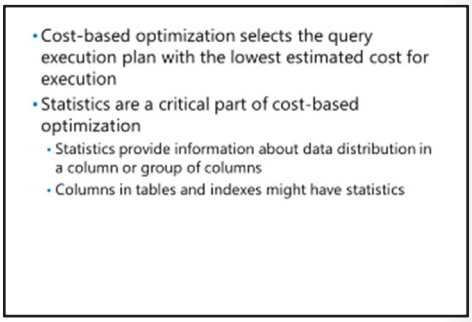
Lesson Objectives

After completing this lesson, you will be able to:

- Describe cost-based optimization.
- Explain predicate selectivity.
- Inspect table and index statistics.
- Understand cardinality estimation.
- Create statistics.
- Update statistics.
- Create filtered statistics.

Cost-Based Optimization

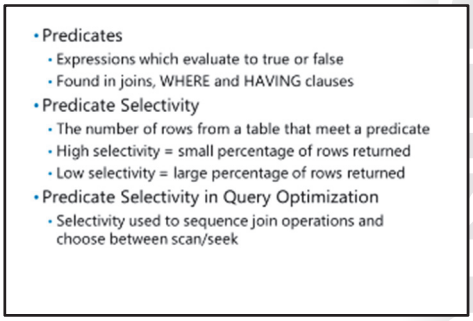
The SQL Server query optimizer is a cost-based optimizer—it analyzes different execution plans for a given query, estimates the cost of each plan, and selects the plan with the lowest cost for execution. The goal of query optimizer is not to find the best plan out of every possible plan; instead, the target is to find an optimal plan quickly. Therefore, it has to strike a balance between the quality of the plan and the time it takes to evaluate plans.



Cost-based estimation carried out by the query optimizer is based on multiple factors, including the availability of indexes and statistics. Statistics provide information on the data distribution in one or more columns of a table, or of an index. The query optimizer uses statistics to estimate the number of rows that could be returned by a query. It then assigns appropriate CPU, I/O, and memory costs to the operators used in a plan. A plan's estimated cost is the sum of the costs for all the operators used. If statistics are stale or missing, the query optimizer may choose a suboptimal plan by either overestimating or underestimating the cost of query execution plans.

Additional Reading: For more information on the internals of the query optimizer, see Module 7 of this course: *Query Execution and Query Plan Analysis*.

Predicate Selectivity



A predicate is an expression which evaluates to true or false; in Transact-SQL statements they are found in joins, WHERE clauses, and HAVING clauses. In queries, predicates act as filters.

The following example has one predicate in the INNER JOIN clause (ON od.SalesOrderID = oh.SalesOrderID):

INNER JOIN predicate

```
SELECT oh.CustomerID, od.ProductID
FROM Sales.SalesOrderHeader AS oh
INNER JOIN Sales.SalesOrderDetail AS od
ON od.SalesOrderID = oh.SalesOrderID;
```

The following example has two predicates in the WHERE clause:

- ModifiedDate > '2011-02-01'
- StoreID = 10

WHERE predicates

```
SELECT CustomerID, AccountNumber
FROM Sales.Customer
WHERE ModifiedDate > '2011-02-01'
AND StoreID = 10;
```

In a Transact-SQL query, the selectivity of a predicate is the percentage of all the rows that might be returned by the query that the predicate will return. A predicate with high selectivity will filter a large percentage of the total possible rows; few rows will be returned. A predicate with low selectivity will filter a small percentage of the total possible rows; many rows will be returned.

The selectivity of a predicate can be calculated against a table using the following formula:

```
predicate selectivity = (number of rows meeting the predicate) / (number of rows
in the table)
```

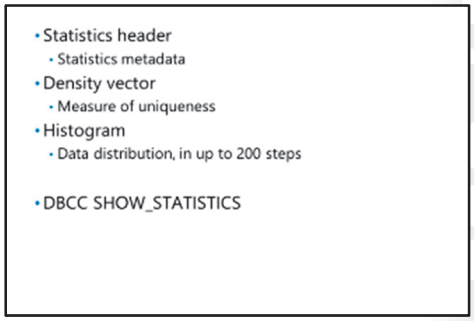
If all the rows in a table will meet the predicate, the predicate's selectivity is 1.0. If none of the rows in a table will meet the predicate, the predicate's selectivity is 0. For example, if a table has 1,000 rows, 10 of which meet a predicate, then the predicate's selectivity is 10/1000, or 0.01.

Note: The more highly selective a predicate, the lower the selectivity value. A predicate with a selectivity of 0.003 has high selectivity (0.3 percent of table rows will be returned). A predicate with a selectivity of 0.9 has low selectivity (90 percent of table rows will be returned).

Predicate Selectivity in Query Optimization

Predicate selectivity matters in query optimization because it is an important measure that the query optimizer can use to select between query execution plans. If a selective predicate is applied early in a query execution plan, fewer rows must be processed in the remainder of the plan. For instance, the query optimizer will use selectivity to choose between index scan and index seek operations, and to order tables in a join operation.

Inspecting Statistics



SQL Server summarizes information about the distribution of data values in tables and indexes in statistics objects. An individual table statistics object may include statistics about more than one column.

There are three aspects of a statistics object that you might normally access:

- A statistics header
- A density vector
- A histogram

Statistics Header

The statistics header includes metadata about the statistics object, including:

- The name of the statistics object; the name of a statistics object covering table columns will vary, depending on how it is created.
- The name of a statistics object relating to an index.
- The date statistics were last updated.
- The number of rows sampled for the last update.
- The number of steps in the histogram.

Density Vector

The density vector is a measure of the uniqueness of data values for each column or combination of columns that the statistics object covers; it is expressed as a decimal value in the range from 0 to 1 that is calculated as follows:

```
density vector = 1 / [number of distinct values in the column(s)]
```

For example, consider a column C that contains city names; it has 10 rows, each with different values. The density of column C will be 1/10=0.1.

When a statistics object covers more than one column, the density vector will contain multiple rows. The first row will include a density vector for the first column in the statistics object, the second row will contain a density vector for the first and second columns in the statistics object, and so on. These vectors provide information about the correlation of values between columns in the statistics object.

Histogram

The histogram contains information about the distribution of data values for the first column covered by the statistics object. The values in the column are summarized into steps—also referred to as buckets or ranges. A histogram can contain up to 200 steps. Each step holds the following information—the value in brackets at the end of each item is the name given to this column in the output of DBCC SHOW_STATISTICS:

- The upper bound data value in the data range covered by the step (**RANGE_HI_KEY**).
- The number of rows in the step, excluding the upper bound (**RANGE_ROWS**).
- The number of rows with data values equal to the upper bound (**EQ_ROWS**).
- The number of distinct values in the step, excluding the upper bound (**DISTINCT_RANGE_ROWS**).
- The average number of rows with duplicate column values in the step, excluding the upper bound (**AVG_RANGE_ROWS**).

A histogram may be generated, based on a full scan of the data values covered by the statistics object, or from a representative sample of data values.

Viewing Statistics

You can use the **sys.stats** and **sys.stats_colums** system views to get information about the statistics objects that exist in a database, and the tables and columns to which they relate.

For more information about **sys.stats**, see the topic *sys.stats (Transact-SQL)* in Microsoft Docs:

sys.stats (Transact-SQL)

http://aka.ms/sj492v

The STATS_DATE function can be used to return the date when a statistics object was last updated; this is the same last updated date information that is returned on the statistics header.

For more information on the STATS_DATE function, see the topic *STATS_DATE (Transact-SQL)* in Microsoft Docs:

STATS_DATE (Transact-SQL)

http://aka.ms/qsccit

Detailed statistics data can be viewed using the DBCC command DBCC SHOW_STATISTICS. DBCC SHOW_STATISTICS can be used to return the statistics header, density vector, histogram, or statistics stream for a statistics object.

For more information on DBCC SHOW_STATISTICS, see the topic *DBCC SHOW_STATISTICS (Transact-SQL)* in Microsoft Docs:

DBCC SHOW_STATISTICS (Transact-SQL)

http://aka.ms/snajce

For more information on statistics in SQL Server, see the topic *Statistics* in MSDN:

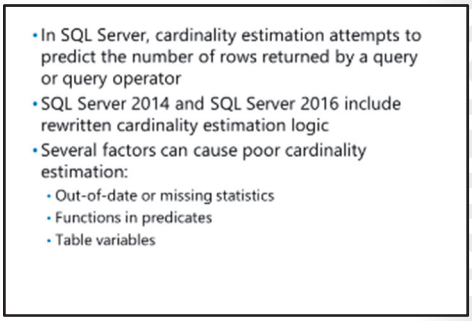
Statistics

http://aka.ms/pml4qg

Cardinality Estimation

In the context of databases, cardinality normally refers to the number of unique data values in a column.

In the case of cardinality estimation in SQL Server, cardinality refers to the number of rows either in a query result or in an interim result set output by a query operator—for example, an index seek or index scan. The query optimizer uses cardinality as a method to select between alternative query execution plans for a statement. The cardinality of a statement or operator is closely related to the selectivity of any predicates.



Cardinality estimation is driven by table and index statistics. The query optimizer uses a number of methods to estimate cardinality, including:

- For simple predicates, if the search value is equal to the histogram upper bound value (**RANGE_HI_KEY**), then **EQ_ROWS** will give a fairly accurate estimate.
- If the search value falls within a histogram step, then the average density of values in that histogram step provides an estimate of cardinality.
- If the search value is unknown at compile time, then the optimizer uses the average column density to calculate the number of rows that would match an average value in the column.
- If no other information is available, the query optimizer uses default cardinality values.

SQL Server's cardinality estimation logic—the cardinality estimator—was rewritten in SQL Server 2014, and further enhancements were added in SQL Server 2016. In both SQL Server 2014 and SQL Server 2016, you can select which version of the cardinality estimator to use; by default, this is controlled at database level by the database compatibility level setting, although it can also be controlled using trace flags.

For more information about cardinality estimation in SQL Server, and how to select different versions of the cardinality estimator, see the topic *Cardinality Estimation (SQL Server)* in Microsoft Docs:

Cardinality Estimation (SQL Server)

http://aka.ms/yc3eqb

In some circumstances, the cardinality estimator can produce poor estimates, resulting in the selection of a suboptimal query execution plan. Examples include:

- **Missing or bad statistics**. This results in inaccurate cardinality estimation. The resolution is to create or update the relevant statistics or indexes.
- **Functions in predicates**. Statistics are not used for queries that join or filter columns using arithmetic or string functions. This can be resolved by precomputing the output of the function, either in a temporary table or a computed column.
- **Table variables**. SQL Server does not maintain a density vector or histogram for table variables; by default, the estimated row count for table variables is 1. This can have negative effects on performance as the actual row count of the table variable increases. This can be resolved in a number of ways:
  - Use a temporary table in place of a table variable; temporary tables have a full set of statistics.
  - Mark statements that use table variables for recompilation with the OPTION (RECOMPILE) hint. This forces the actual row count of the table variable to be used in the plan at the cost of additional recompilations.
  - Use trace flag 2453. When this trace flag is enabled, changes in table variable row count can mark a statement for recompilation.

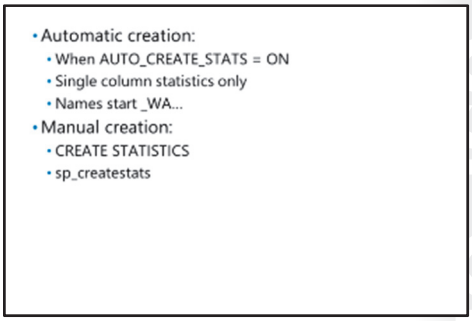
For more information on trace flag 2453, see the topic *FIX: Poor performance when you use table variables in SQL Server 2012 or SQL Server 2014* on the Microsoft Support website. Note that the article references SQL Server 2012 and 2014, however later versions also supports this trace flag:

FIX: Poor performance when you use table variables in SQL Server 2012 or SQL Server 2014

http://aka.ms/nx0q4k

Creating Statistics

Statistics can either be created automatically by SQL Server, or created manually as required.



Automatic Creation

When the AUTO_CREATE_STATISTICS database option is set to ON—the default setting—SQL Server will automatically create statistics that do not already exist for single table columns when the column is referenced by a query predicate. Automatically created statistics have the following properties:

- Missing column statistics are created on individual columns.
- Only single column statistics are created; multicolumn statistics are never created automatically.
- Filtered statistics are not created automatically.
- The name of automatically created statistics will start with _WA.
- The **auto_created** column for the statistics object in the **sys.stats** catalog view will have the value 1.

Note: Statistics are always created for an index when it is created, using a full scan of the index key columns. The value of the AUTO_CREATE_STATISTICS setting does not affect the generation of statistics for indexes.

Manual Creation

Two methods are available to create statistics manually; the CREATE STATISTICS command and the system **sp_createstats** stored procedure:

- **CREATE STATISTICS**. You can use this command to create single column, multicolumn, and filtered statistics on one or more columns of a table or indexed view. You can specify many options, including:
  - The name of the statistics object.
  - The table or indexed view to which the statistics refer.
  - The columns included in the statistics.
  - The sample size on which the statistics are based. This may be a scan of the whole table, a percentage of rows, or a count of rows.
  - A filter for the statistics; filtered statistics are covered later in this lesson.
  - Whether statistics should be created per-partition or for the whole table.
  - Whether statistics should be excluded from automatic update.
- **sp_createstats**. The **sp_createstats** stored procedure is a wrapper procedure for a call to CREATE STATISTICS for creating single column statistics on all columns in a database not already covered by statistics. It accepts a limited selection of the options and parameters supported by CREATE STATISTICS.

For more information on CREATE STATISTICS, see the topic *CREATE STATISTICS (Transact-SQL)* in Microsoft Docs:

CREATE STATISTICS (Transact-SQL)

http://aka.ms/gdg0hw

For more information on **sp_createstats**, see the topic *sp_createstats (Transact-SQL)* in Microsoft Docs:

sp_createstats (Transact-SQL)

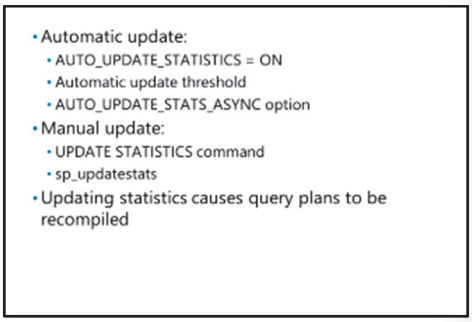
http://aka.ms/ak1mcd

Some of the scenarios where manual statistics can be helpful include:

- When the query predicate has multiple correlated columns that do not exist in any of the indexes.
- When the query has missing statistics.

Updating Statistics

Statistics objects are not automatically updated after every data change; when data is added, removed, and changed, statistics can become inaccurate. Because statistics become more inaccurate over time, they can lead to poor cardinality estimates and the selection of suboptimal query execution plans. Therefore, statistics should be regularly updated to take account of data changes. Statistics can be updated manually or automatically.



Automatic Update

When the AUTO_UPDATE_STATISTICS database option is ON (the default value), SQL Server can detect and update stale statistics. SQL Server detects stale statistics at query compilation time using the following rules:

- Data is added to an empty table.
- The table had 500 or fewer rows at the time of statistics creation, and the column modification counter of the first column of statistics has changed by more than 500 rows.
- The table had 500 or more rows at the time of statistics creation, and the column modification counter of the first column of statistics has changed by more than 500 plus 20 percent of the number of rows in the table when statistics were collected.

A statistics object manually created or updated with the NORECOMPUTE option is excluded from automatic update, regardless of the value of the AUTO_UPDATE_STATISTICS database option.

Statistics can be updated synchronously (the query will wait for the statistics update to complete before executing), or asynchronously (the query executes immediately and a statistics update is triggered in the background). When the AUTO_UPDATE_STATISTICS_ASYNC database option is ON, statistics are updated asynchronously. The default AUTO_UPDATE_STATISTICS_ASYNC value is OFF.

Manual Update

Two methods are available to create statistics manually; the UPDATE STATISTICS command and the **sp_updatestats** system stored procedure:

- UPDATE STATISTICS. This command can be used to update a specific statistics object, or all statistics on a table or indexed view. Options comparable to CREATE STATISTICS are available.
- **sp_updatestats**. The **sp_updatestats** system stored procedure is a wrapper procedure used to call UPDATE STATISTICS for all statistics objects in a database.

For more information on UPDATE STATISTICS, see the topic *UPDATE STATISTICS (Transact-SQL)* in Microsoft Docs:

UPDATE STATISTICS (Transact-SQL)

http://aka.ms/fwgduo

For more information on **sp_updatestats**, see the topic *sp_updatestats (Transact-SQL)* in Microsoft Docs:

sp_updatestats (Transact-SQL)

http://aka.ms/s63xaz

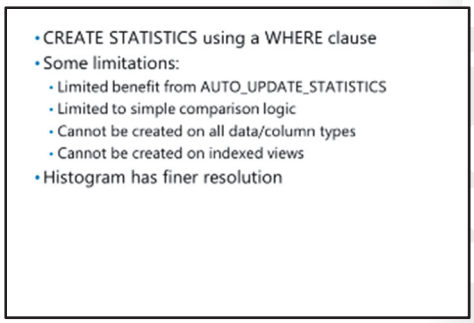
A manual statistics update might be required, even when AUTO_UPDATE_STATISTICS is ON, if the stale statistics detection method does not update statistics frequently enough. The stale statistics detection method requires that approximately 20 percent of the rows in a table must change before statistics are updated. As a table grows, this threshold will be reached less and less frequently. You may therefore decide to manually update statistics on large tables.

You might also decide to update statistics manually as part of a maintenance operation that significantly changes the number of rows in a table—for example, bulk insert or truncation. Doing this can avoid delays for queries that would otherwise have to wait for an automatic statistics update to complete.

 **Note:** Index maintenance operations do not alter the distribution of data, so you do not need to update statistics after running ALTER INDEX REBUILD, DBCC REINDEX, DBCC INDEXDEFRAG, or ALTER INDEX REORGANIZE.
Statistics are automatically updated when you run ALTER INDEX REBUILD or DBCC DBREINDEX.

 **Note:** When statistics are updated, any cached query execution plans based on the statistics are marked for recompilation. You should avoid updating statistics too frequently, because doing so will increase the amount of CPU time spent on recompiling query execution plans.

Filtered Statistics



SQL Server supports the creation of statistics on a subset of rows in a table, referred to as filtered statistics. Filtered statistics are created by specifying a WHERE clause as part of a CREATE STATISTICS statement.

The following example creates filtered statistics for the **Sales**.**SalesOrderHeader**.**PurchaseOrderNumber** column where **TerritoryID** = 1:

Filtered Statistics Example

```
CREATE STATISTICS st_SalesOrderHeader_Territory1
ON Sales.SalesOrderHeader (PurchaseOrderNumber)
WHERE TerritoryID = 1;
```

Filtered statistics have the following limitations:

- Filtered statistics might not benefit from the automatic update statistics process, if it is enabled. A job or process to manually update filtered statistics is recommended to keep them up to date.
- The filter predicate is limited to simple comparison logic, such as <, >, <=, >=, =, !=, IS NULL, and IS NOT NULL.
- Filtered statistics cannot be created on computed columns, user defined data types, spatial data types, or the **hierarchyid** data type.
- Filtered statistics cannot be created on indexed views.

Filtered statistics can be useful on large tables, to get statistics at a finer grain than would otherwise be available. Because a statistics histogram can have a maximum of 200 steps, each step may contain thousands or millions of rows for a large table; this lack of resolution can hinder accurate cardinality estimation. Using several filtered statistics objects to cover the same column will reduce the number of rows in each step, improving cardinality estimation.

Demonstration: Analyzing Cardinality Estimation

In this demonstration, you will see:

- Worked examples of cardinality estimation.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In file explorer, navigate to the **D:\Demofiles\Mod06** folder, right-click **Setup.cmd** and click **Run as Administrator**.
3. Click **Yes** when prompted at the **User Control** dialog box.
4. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine instance using Windows authentication.
5. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod06\Demo** folder.
6. Open the **Demo 1 - cardinality.sql** script file.

7. Execute the code under the comment that begins **Step 1** to use the **AdventureWorks** database.
8. Execute the code under the comment that begins **Step 2** to display the statistics objects linked to **Person.Person**.
9. Execute the code under the comment that begins **Step 3** to examine the **PK_Person_BusinessEntityID** statistics object.
10. Execute the code under the comment that begins **Step 4** to examine the **IX_Person_LastName_FirstName_MiddleName** statistics object.
11. Highlight the code under the comment that begins **Step 5** and press Ctrl+L to examine the estimated execution plan for a simple query matching a ROW_HI_KEY value.
12. Execute the code under the comment that begins **Step 6** to examine the **IX_Person_LastName_FirstName_MiddleName** statistics object again.
13. Highlight the code under the comment that begins **Step 6a** and press Ctrl+L to examine the estimated execution plan for a simple query within a step range.
14. Execute the code under the comment that begins **Step 7** to examine the header of the **IX_Person_LastName_FirstName_MiddleName** statistics object.
15. Execute the code under the comment that begins **Step 7a** to examine the density vector of the **IX_Person_LastName_FirstName_MiddleName** statistics object.
16. Highlight the code under the comment that begins **Step 7b** and press Ctrl+L to examine the estimated execution plan for a parameterized query.
17. Execute the code under the comment that begins **Step 7c** to demonstrate how the estimated number of rows from the plan in the previous step is calculated.
18. Highlight the code under the comment that begins **Step 7d** and press Ctrl+L to demonstrate that the calculation is not affected by the parameter value.
19. Highlight the code under the comment that begins **Step 8** and press Ctrl+L to examine the estimated execution plan when a function is used in the predicate.
20. Highlight the code under the comment that begins **Step 8a** and press Ctrl+L to examine the estimated execution plan when two columns from the same table are compared. Note that this is the same as the plan in the previous step.
21. Highlight the code under the comment that begins **Step 9** and press Ctrl+L to examine the estimated execution plan for a query with multiple predicates.
22. Execute the code under the comment that begins **Step 9a** to show that a new statistics object has been created by AUTO_CREATE_STATISTICS.
23. Execute the code under the comment that begins **Step 9b** to view the histogram for the new statistics object.
24. Execute the code under the comment that begins **Step 9c** to demonstrate the cardinality calculation used to generate the estimated row count for the query under the heading **Step 9**.
25. Leave SSMS open for the next demonstration.

Check Your Knowledge

Question	
Which of the following cannot be returned by the DBCC SHOW_STATISTICS command?	
Select the correct answer.	
	AUTO_UPDATE_STATISTICS setting
	Statistics stream
	Statistics header
	Density vector
	Histogram

Lesson 2
Index Internals

This lesson goes into detail about indexes in the SQL Server Database Engine. It covers the internal structure of different index types, and the criteria you might use to create indexes and select index keys.

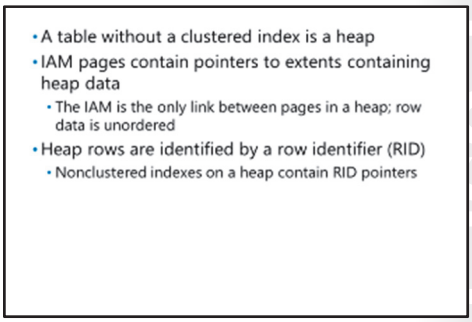
Lesson Objectives

At the end of this lesson, you will be able to:

- Describe the structure of a heap.
- Describe the structure of clustered and nonclustered indexes.
- Select an appropriate index key.
- Explain the differences between single column and multicolumn indexes.
- Use filtered indexes.
- Explain how the query optimizer determines which indexes to use.
- Describe how indexes are affected when data is modified.
- Identify index fragmentation.
- Explain the importance of index column order.
- Identify and create missing indexes.

Heap Internals

A heap is a table without a clustered index, although it may have one or more nonclustered indexes. A table without a clustered index is still a heap, even if it has a primary key defined on a nonclustered index.



Tables that are heaps can be identified by querying the **sys.partitions** view for partitions where **index_id** = 0.

A join to the **sys.objects** view is used to filter out system tables that are heaps.

Identify User Heaps

```
SELECT DISTINCT o.name AS heap_name
FROM sys.partitions AS p
JOIN sys.objects AS o
ON o.object_id = p.object_id
WHERE index_id = 0
AND o.is_ms_shipped = 0;
```

Although the data is usually stored in the order in which the rows were inserted, data rows stored in a heap do not have any specific physical order. SQL Server may move data at any time, to optimize storage; the row order of a heap is not guaranteed. Individual pages allocated to a heap are not linked to each other.

The Index Allocation Map (IAM) keeps track of the data in a heap. The IAM is made up of one or more pages that contain pointers to the extents that contain data pages for the heap. Serial scans of the heap are carried out by accessing the IAM pages sequentially to locate the extents that contain row data; therefore, the sequence of IAM pages controls the sequence in which row data is read.

Individual rows in a heap are identified by a row identifier (RID), made up of the data file number, data page number, and the page slot numbers. A nonclustered index on a heap is made up of pointers to RID values.

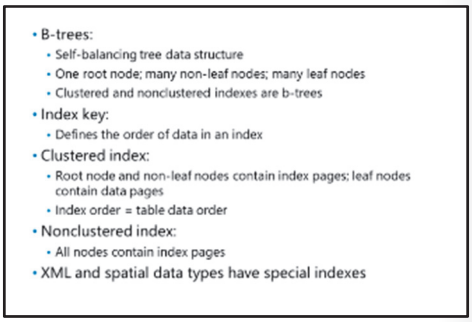
For more information on heaps, see the topic *Heaps (Tables without Clustered Indexes)* in Microsoft Docs:

Heaps (Tables without Clustered Indexes)

http://aka.ms/ttbauu

Index Structure

Row-based clustered and nonclustered indexes in SQL Server are built using the same data structure—b-trees.



B-Trees

A b-tree is a sorted, self-balancing data tree structure. A b-tree is made up of multiple nodes, organized in a hierarchical tree structure; there is a single root node with zero or more child nodes, each of which has zero or more child nodes of its own, and so on. A node with no child nodes is referred to as a leaf node. A node that has child nodes, but is not the root node, is referred to as a non-leaf node. Each non-leaf node contains a number of keys that act as pointers to the values contained by its child nodes.

When data is added to and removed from a b-tree, nodes may be split or merged as appropriate. All leaf nodes in a b-tree must be at the same depth—meaning they must all have the same number of non-leaf node parents up to the root of the b-tree.

Index Key

The order of data in an index is determined by the index key. The index key may be composed of up to 16 columns, with a size of no more than 900 bytes. Index keys are discussed in more depth later in this course.

Clustered Indexes

A clustered index determines the order in which data is stored in a table. A clustered index is a b-tree with the following characteristics:

- The root node has one page, called the root page.
- Leaf nodes contain the data pages of the table.

- The root nodes and non-leaf nodes contain index pages; each row in an index page points to either an intermediate level page or a data row in the leaf node.
- Each non-leaf page and leaf page contains a pointer to the previous page and the next page in the index; this improves performance for sequential read operations.
- In a table with multiple partitions, each partition has its own b-tree structure.
- Pages in a clustered index are ordered by the clustered index key value; there can only be one clustered index in a table.

A clustered index can be identified by **index_id** = 1 in the **sys.indexes** and **sys.partitions** system views.

Nonclustered Indexes

The order of data in a nonclustered index is independent of the order in which data is stored. A nonclustered index is a b-tree similar to a clustered index, with the following differences:

- Leaf nodes contain index pages—unlike a clustered index, where the leaf nodes contain data pages.
- Each leaf node page contains a pointer to the previous leaf node page and the next leaf node page in the index; this improves performance for sequential read operations.
- Each index row in a nonclustered index contains the nonclustered index key and row locator. The row locator points to the data row in a heap or a clustered index. A row locator has one of the following structures:
  - If nonclustered is on a heap, the row locator is of the format fileid:pageid:rowid.
  - If nonclustered is on a table with a clustered index, the row locator is the clustered index key of the row. If the clustered index has duplicate key values, SQL Server makes them unique by adding an internal 4-byte value to the duplicate rows.
- The index structure is stored separately from table structure; a nonclustered index requires storage space additional to that occupied by the table data.
- A table may have up to 999 nonclustered indexes.

A nonclustered index can be identified by **index_id** > 1 in the **sys.indexes** and **sys.partitions** system views.

Included Columns

In addition to the index key columns, a nonclustered index can be defined with nonkey columns known as included columns. Included column values are stored at the leaf level of a nonclustered index. Adding included columns to a nonclustered index makes it possible for more queries to be answered without reference to the table data rows, and without expanding the index key.

For more information on clustered and nonclustered indexes, see the topic *Clustered and Nonclustered Indexes Described* in Microsoft Docs:

Clustered and Nonclustered Indexes Described

http://aka.ms/uks0a8

XML and Spatial Indexes

The **xml** data type and the spatial data types (**geometry** and **geography**) can be indexed. Because XML and spatial data types are compound data types—each data item contains multiple pieces of information—SQL Server must decompose them into a tabular structure so that they can be indexed.

There are two kinds of XML index:

- **Primary XML index**. The XML data is decomposed into a tabular structure in the primary XML index. This index can be large.
- **Secondary XML index**. Indexes added to the primary index tabular structure. Secondary XML indexes may index by PATH, VALUE, or PROPERTY.

Spatial data types have one index type.

For more information on spatial indexes, see the topic *Spatial Indexes Overview* in Microsoft Docs:

Spatial Indexes Overview

http://aka.ms/x5ey2f

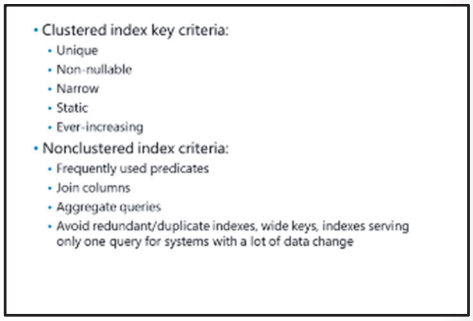
For more information on XML indexes, see the topic *XML Indexes (SQL Server)* in Microsoft Docs:

XML Indexes (SQL Server)

http://aka.ms/kwaz64

Picking the Right Index Key

Different criteria apply when you are selecting the index key for a clustered index, as opposed to a nonclustered index.



Clustered Index Key Criteria

The column or columns selected for a clustered index key should have the following properties:

- **Unique**. SQL Server does not require that the key of a clustered index should be unique. However, if the clustered index key is not unique, SQL Server will add a unique 4-byte value to each value in the index key so it can be uniquely identified. To minimize storage requirements, it is advisable that a clustered index key should be unique.
- **Non-nullable**. In addition to reasons of uniqueness, a nullable clustered index key should ideally be avoided, because managing NULLs requires an additional three to four bytes of storage per row for the NULL block. Indexes with non-nullable keys do not have the overhead of a NULL block.
- **Narrow**. A clustered index key should be as narrow as possible. The depth of the index (the number of non-leaf nodes) depends on the data type of the clustering key. The clustered index key will appear on every index page (on the non-leaf nodes of a clustered index, or the leaf nodes of a nonclustered index). The narrower the key, the more index references can fit on each index page. This will result in fewer I/O operations than a broader index key.

- **Static**. The values in a clustered index key should be as static as possible; ideally, they should never be updated. If a clustered index key value is changed, the table data pages must be reordered to maintain the correct sequence of values. Any nonclustered indexes on the table are also updated accordingly. Therefore, to avoid the overhead of modification, it is recommended to create a clustered index on a static column.
- **Ever-increasing**. An ever-increasing key has two main benefits:
  - Fast inserts. With an ever-increasing key, the rows will always be added to the most recently allocated page. This can improve the performance of INSERT operations.
  - Reduced fragmentation. A non-sequential key insert may result in page splits, causing fragmentation.

Nonclustered Index Criteria

A nonclustered index speeds up data searches for queries focusing on a subset of data. You might consider creating nonclustered indexes for columns used in the following situations:

- **Frequently used predicates**. Columns frequently used in query predicates are excellent candidates for nonclustered index keys.
- **Join columns**. A nonclustered index on columns used in JOIN clauses will improve query performance.
- **Aggregate queries**. Performance of aggregate queries, including COUNT, MIN, and MAX, can benefit from a nonclustered index on the aggregated column(s).

Systems with large volumes of data that are updated infrequently will benefit most from the addition of nonclustered indexes. When designing nonclustered indexes for systems where data is updated frequently, beware of the following issues:

- **Redundant indexes**. One or more indexes on the same set of columns might decrease the performance of query compilation, because the query optimizer must choose between two similar indexes. There is also an overhead of maintaining and updating the indexes when data is modified.
- **Indexes with wide keys**. As with clustered indexes, a wide nonclustered index key increases the depth of the index, so can reduce performance because more pages must be read to access the index. Index maintenance is also costly, because many pages must be updated when indexed values change. This is a particular problem with composite keys (index keys composed of more than one column). Consider including columns in an index—which only increases storage at leaf level—rather than using a wide index key. This is particularly important on systems where data is frequently updated.
- **Index for one query**. In most circumstances, a nonclustered index should benefit more than one query. Creating an index to optimize a single query can result in many indexes on a single table. This will speed up the select queries; however, the insert and update queries will be adversely affected. There are clearly exceptions to this guidance; if a query is executed frequently, you might be justified in creating an index for it, even if the index benefits only one query.

For more information on index design, see the topic *SQL Server Index Design Guide* on Microsoft Technet:

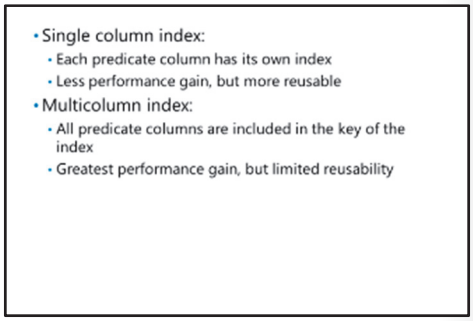
SQL Server Index Design Guide

http://aka.ms/vpdlz7

Single Column and Multicolumn Indexes

When you design nonclustered indexes for a table that will be queried with multiple predicates, you could:

- Give each column filtered by a predicate its own index—single column indexes.
- Create a single index with a compound key made up of all the predicate columns—a multicolumn index.



Both designs offer a performance improvement over no index at all—the query optimizer can select a query execution plan that uses single column indexes in serial to meet a predicate filter; in a multicolumn index, multiple predicates can be evaluated in a single query execution plan operator.

For example, the following query uses three predicates on the **Person.Person** table:

Three-Predicate Query

```
SELECT BusinessEntityID
FROM Person.Person
WHERE FirstName = N'Xavier'
AND EmailPromotion = 1
AND ModifiedDate < '2015-01-01';
```

Following a single column indexing strategy, you would create an index on each predicate column individually.

Create an index on each predicate column.

Single Column Indexing

```
CREATE NONCLUSTERED INDEX ix_person_firstname ON Person.Person (FirstName);
CREATE NONCLUSTERED INDEX ix_person_emailpromotion ON Person.Person (EmailPromotion);
CREATE NONCLUSTERED INDEX ix_person_modifieddate ON Person.Person (ModifiedDate);
```

Following a multicolumn indexing strategy, you would create a single index with a composite key on all three predicate columns.

Indexes with composite keys.

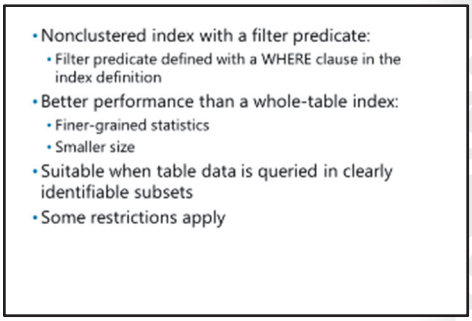
Multicolumn Indexing

```
CREATE NONCLUSTERED INDEX ix_person_firstname_emailpromotion_modifieddate
ON Person.Person (FirstName, EmailPromotion, ModifiedDate);
```

In the context of the example query, a multicolumn index gives the greatest performance gain; the optimal query execution plan is a single index seek operation on the multicolumn index. However, depending on the profile of the database workload, this index may not be of use to any other queries that reference the table. Although single column indexes give a lesser gain in performance, they might be preferable because they are more reusable. Even if a multicolumn index is applicable only to one query, you might decide to create it, if the performance gain is sufficiently great.

Filtered Indexes

A nonclustered index may optionally be created with a filter; the filter has the effect of applying the index to a subset of rows in the table. The filter predicate is defined by including a WHERE clause in the index definition. You can use a filtered index to index only the subset of the data in a table that you know will be frequently queried.



For example, the following code creates a filtered index on the **Color** column of **Production.Product**, where **Color** is not NULL and **ReorderPoint** is less than 500:

Creating a filtered index.

Filtered Index Example

```
CREATE INDEX ix_product_color_filtered
ON Production.Product (Color)
WHERE Color IS NOT NULL
AND ReorderPoint < 500;
```

Filtered indexes offer the following benefits:

- **Better query plans and query performance.** A filtered index has a statistics object with the same filter. The statistics histogram will have a finer grain than a full-table index, which will give more accurate cardinality estimates for queries where the filtered index can be used.
- **Reduced index size**. Reducing the number of rows covered by the index reduces the size of the index on disk. As the index is smaller, maintenance operations on the index will be quicker to run.

Note: Filtered indexes have many similarities to filtered statistics, which you learned about in the previous lesson.

Filtered indexes are useful when a table contains subsets of data which can be clearly identified, and which are commonly referenced in queries. Examples include:

- Nullable columns which contain few non-NULL values.
- Columns that contain categories of data, such as analysis or status codes.
- Columns containing ranges of values, such as currency amounts, times, and dates.

Filtered indexes have some limitations, including:

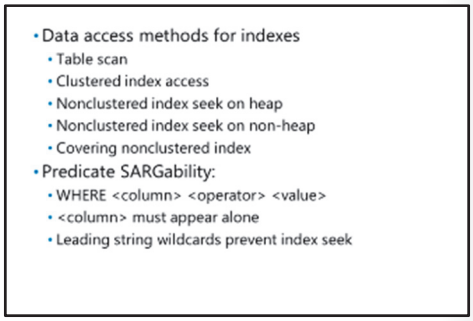
- Filtered indexes can only be created on tables; they cannot be created on views.
- The filter predicate supports simple comparison operators only.
- Data conversion can only occur on the right-hand side of the predicate operator.

For more information on filtered indexes, see the topic *Create Filtered Indexes* in Microsoft Docs:

Create Filtered Indexes

http://aka.ms/mudq8y

The Query Optimizer's Choice of Indexes



The query optimizer tries to select the lowest-cost method to read data from tables. One of the main criteria to determine the cost of a table or index access method is the number of logical reads, which is the number of pages read from buffer cache to complete the operation.

As you have learned, the query optimizer uses cardinality estimation to predict the number of rows returned by a predicate and how many logical reads would be needed to retrieve the qualifying rows. It might select an index to fetch the qualifying rows, or it might consider a table scan to be a better option. It tries to find the access method with the lowest number of logical reads.

Data Access Methods for Indexes

Some of the common data access methods include:

- **Table scan.** A table scan sequentially scans the data pages of a table to filter out the rows that are part of a result set. The optimizer might choose a table scan when a table does not have indexes, or if it decides not to use existing indexes. The number of logical reads in a table scan is equal to the number of data pages in a table.
- **Clustered index access**. The number of logical reads in clustered index access is equal to the number of levels in the index, plus the number of data pages to scan. A clustered index scan or seek—unlike a table scan—does not read all the table data pages to fetch the result set. Only the pages matching the predicate are read. In most cases, this considerably decreases the logical reads, increasing the query performance.
- **Nonclustered index seek on a heap**. This access method reads the qualifying nonclustered index pages, and then follows the row identifier (RID) to read the data pages of the underlying table. The number of logical reads is the sum of the number of index levels, the number of the leaf pages, and the number of qualifying rows (each of which requires a lookup to the data page of the underlying table). The same data page might be retrieved many times from the cache; therefore, the number of logical reads can be much higher than the number of pages in the table.
- **Nonclustered index seek on a table with clustered index**. For this access method, the number of logical reads is the sum of the number of index levels, the number of leaf nodes, and the number of qualifying rows, multiplied by the cost of the clustered index key lookup. The access method reads the qualifying pages of the nonclustered index, then locates the corresponding data row(s) using the clustered index key.
- **Covering nonclustered index**. A nonclustered index is said to be a covering index if it has all necessary information in the index key and a lookup to a data page is not required. The logical read in this case is the sum of the number of index levels and the number of leaf index pages. The number of leaf index pages is equal to the number of qualifying rows, divided by the number of rows per page. You can INCLUDE columns in a nonclustered index so it can cover a query.

Predicate SARGability

A Search Argument (SARG) is a predicate that is suitable for use with an index. Even when an index exists on a filtered column, it can only be used if the predicate value can be converted to a SARG—or is SARGable.

If a predicate is not SARGable, the query optimizer may select an index scan instead of an index seek, or opt to ignore an index entirely. To be SARGable, a predicate must take the following form:

```
WHERE <column> <operator> <value>
```

The column value must appear alone on one side of the operator; any calculation on the column value renders the predicate non-SARGable.

The following are examples of SARGable predicates:

SARGable Predicates

```
WHERE LastName = N'Accah'

WHERE 100 < BusinessEntityID

WHERE ReorderPoint BETWEEN 250 AND 750

WHERE ModifiedDate = '2015-01-02'
```

The following are examples of predicates which are not SARGable:

Non-SARGable Predicates

```
WHERE LEFT(LastName,2) = N'Ac'

WHERE 100 < ABS(BusinessEntityID)

WHERE ReorderPoint - 500 BETWEEN 250 AND 750

WHERE CONVERT(varchar(20), ModifiedDate, 112) = '20150102'
```

Some predicates are not SARGable, even though they meet these criteria, if the value part of the predicate is formatted in a way that prevents efficient use of an index. This is most common when using wildcards with string functions such as LIKE and PATINDEX. A leading wildcard in the search string prevents the query optimizer selecting an index seek operation.

In the following example, an index seek might be used if the **LastName** column was indexed:

SARGable Wildcard

```
SELECT LastName
FROM Person.Person
WHERE LastName LIKE N'L%';
```

In the following example, an index seek could never be used:

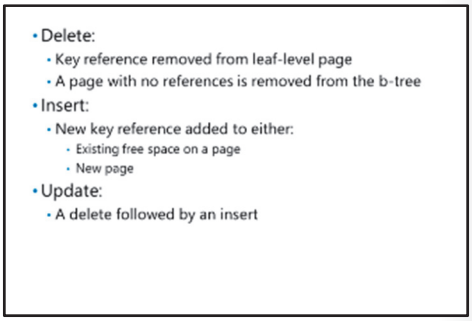
Non-SARGable Wildcard

```
SELECT LastName
FROM Person.Person
WHERE LastName LIKE N'%L';
```

Rules for SARGability are not documented by Microsoft; they may change with new releases of SQL Server.

Data Modification Internals

When data in a table with indexes is changed, SQL Server must immediately update the table's indexes to reflect the changes, so that the sequence of keys in the index remains in the correct order. The way the index is updated depends on the data operation.



Delete

When a row is deleted, references to the row in the relevant leaf-level index pages are removed. For a clustered index, the data row is removed; for a nonclustered index, the index key and row pointer are removed. The empty space in the page is not immediately reclaimed; it may be reused by a subsequent insert operation, or reclaimed when the index is rebuilt or reorganized.

If a deleted key is the only key on an index page, the page will no longer make up part of the index. Logically adjacent leaf pages will be updated to correct their previous page/next page pointers. Non-leaf level pages will be updated to remove references to the page from the index b-tree. The unreferenced page will be unallocated from the table; this can take place immediately or as part of a background process, depending on the locks taken by the delete operation.

Insert

When a row is inserted, references to the row are added to leaf-level index pages. Depending on where the value falls in relation to existing keys in an index, the new reference can be:

- Added in free space in an existing leaf page. If the key is added to an existing page, no change to the non-leaf level pages of the index is required—unless the new key changes the upper bound for key values held on the leaf level page, in which case the boundary values of the non-leaf level pages are updated.
- Added to a new page allocated to the index. Non-leaf level pages are updated to reference the new page. If the key value is added in between existing key values, existing values can be reorganized to occupy parts of the new page—this is a page split. Previous page/next page pointers on logically adjacent index pages are updated to reference the new page.

Update

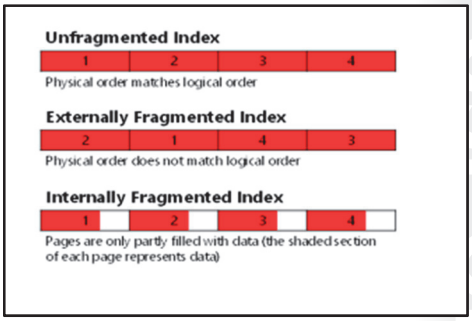
From the perspective of index maintenance, an update operation is carried out as a delete, followed by an insert.

Index Fragmentation

As data in a table changes through inserts, updates, and deletes, indexes on the table can become fragmented. Index fragmentation in SQL Server can take two different forms:

- External fragmentation
- Internal fragmentation

An index may be affected by external fragmentation and internal fragmentation at the same time.



External Fragmentation

External—or logical—fragmentation occurs when the logical order of the pages that make up the index is different from the physical order of the pages that make up the index.

- **Physical order**. The physical order of the pages in an index is determined by the order in which pages are added to the index. Each new page will have a higher page id than all the previous pages in the index. The physical order of an index is the sequence that the data is written to I/O storage.
- **Logical order**. The logical order of an index is determined by the sequence in which index key values are stored in the pages that make up the index. Index pages are linked to one another in a logical sequence by pointers that link each index page to the next and previous pages in the sequence.

When the next logical page is not the next physical page, the index is considered to be logically fragmented.

Logical fragmentation has no effect on index pages that are cached in the buffer pool, but can negatively affect the performance of operations that read or write index pages to I/O storage. External fragmentation is more likely to affect query performance for indexes on large tables that are too large to be cached in memory.

The **avg_fragmentation_in_percent** column in the **sys.dm_db_index_physical_stats** system DMF records the level of logical fragmentation for each index. You can use this to identify the fragmented indexes.

The following query identifies the 10 most externally fragmented indexes in the current database:

Top 10 Externally Fragmented Indexes

```
DECLARE @db int = db_id();
SELECT TOP 10 ips.avg_fragmentation_in_percent, i.name
FROM sys.indexes AS i
CROSS APPLY sys.dm_db_index_physical_stats(@db, i.object_id, i.index_id, NULL, NULL) AS
ips
ORDER BY ips.avg_fragmentation_in_percent DESC;
```

Internal Fragmentation

Internal fragmentation occurs when pages in an index are not completely full; some empty space remains in the index pages. Internal fragmentation can occur for a number of reasons:

- **Page splits**. When a page is too full to accommodate data changes from a new insert or update command, SQL Server moves approximately 50 percent of the data in the page to a newly allocated page to make space for the change. Page splits can also cause external fragmentation.
- **Delete operations**. When rows are deleted from index pages, the empty space left behind causes logical fragmentation.
- **Large row size**. If the index key is large, fewer rows can fit in each index page. A remainder of unusable space might be left in the page. For example, if an index has a 900-byte key (the largest key permitted), eight rows will fit into the 8,060-byte data storage space of a page. This means that 860 bytes (more than 10 percent of the page) is left unfilled. A nonclustered index row can exceed 900 bytes if columns are included in the index.
- **Index fill factor**. To reduce the incidence of page splits, SQL Server allows you to specify that a percentage of each leaf-level page in an index is left unfilled when the page is first allocated. This free space can subsequently be used to accommodate data changes without requiring a page split. The FILL FACTOR setting controls the percentage of the leaf pages which will be filled with data. FILL FACTOR can be specified as part of an index definition, or set as a default at database level. The default FILL FACTOR is 0, meaning that leaf pages are completely filled and no free space is allocated.

When an index is internally fragmented it occupies more space, both in memory and on disk, than it would if it were not logically fragmented; the number of physical reads and logical reads needed to access the index will be higher than they might otherwise be.

The **avg_page_space_used_in_percent** column in the **sys.dm_db_index_physcial_stats** DMF records the internal fragmentation. You can use these to identify internal fragmentation.

The following query identifies the 10 most internally fragmented indexes in the current database:

Top 10 Internally Fragmented Indexes

```
DECLARE @db int = db_id();
SELECT TOP 10 ips.avg_page_space_used_in_percent , i.name
FROM sys.indexes AS i
CROSS APPLY sys.dm_db_index_physical_stats(@db, i.object_id, i.index_id, NULL,
'DETAILED') AS ips
WHERE ips.avg_page_space_used_in_percent > 0
ORDER BY ips.avg_page_space_used_in_percent ASC;
```

For more information about the **sys.dm_db_index_physical_stats** DMF, see the topic *sys.dm_db_index_physical_stats (Transact-SQL)* in Microsoft Docs:

sys.dm_db_index_physical_stats (Transact-SQL)

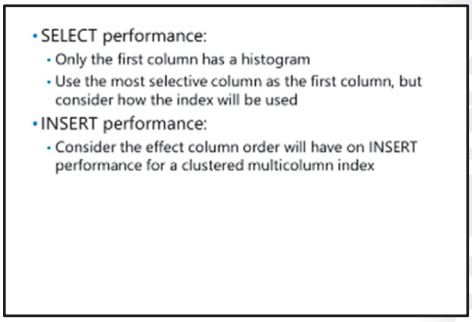
http://aka.ms/lzvjmq

For more information on index FILL FACTOR, see the topic *Specify Fill Factor for an Index* in Microsoft Docs:

Specify Fill Factor for an Index

http://aka.ms/osvify

Index Column Order



When you create a multicolumn index, the sequence in which the columns appear in the index definition can impact the effectiveness of the index.

SELECT Performance

As you have learned, SQL Server only generates a statistics histogram for the first column in a multicolumn index; detailed cardinality estimates can only be made for the first column. Cardinality estimates for the second and subsequent columns in the index are made, based on the density vector. The accuracy of the statistics histogram for the first index column can therefore have a significant effect on query performance.

Remember that a multicolumn index can only be used by queries that reference the first column of the index as a predicate; a multicolumn index cannot be used for index seek operations by queries that reference the second or subsequent index columns—but not the first column.

For best SELECT performance, the general advice when ordering columns in a multicolumn index is to use the most selective column—the column with the greatest number of distinct values—as the first column in the index. You should also consider the expected usage pattern of the index; there is little value in putting the most selective column first in the index if very few queries will reference it.

If your queries use only one of the columns in a multicolumn index as a predicate with the equal to (=), greater than (>), less than (<), or BETWEEN comparators, or in a JOIN to another table, that column should be the first column in the index.

Note: The order of columns in an INCLUDE clause in a nonclustered index definition has no effect on performance; included columns may appear in any order.

INSERT Performance

When you design a multicolumn clustered index, consider how new data will be added to the table when selecting column order. If the order of the columns in the clustered index does not match the sequence in which new data will be inserted, each insert will require that the data in the index be rearranged. This can have a severe negative effect on insert performance.

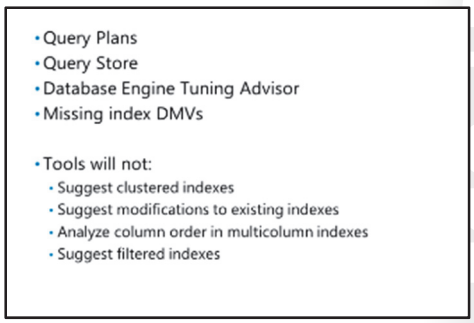
For example, a reporting system includes a table that contains sales information aggregated by date and customer identifier. Date and customer identifier have been identified as keys for a multicolumn clustered index. The customer identifier is the more selective attribute, so it might seem like the natural choice for the first column of the clustered index. However, new data will always be added to the table by date. If customer identifier is selected as the first column of the index, the index will quickly become fragmented and performance will drop. Date is a better choice for the first index column, so that new data is always added to the end of the table.

For more information on index design, see the topic *SQL Server Index Design Guide* on Microsoft Technet:

SQL Server Index Design Guide

http://aka.ms/vpdlz7

Identify and Create Missing Indexes



SQL Server provides tools to assist you in identifying missing indexes—that is, indexes that do not exist, but which the database engine has identified as having the potential to improve the performance of a query.

Query Execution Plans

Where a missing index is identified, estimated and actual query execution plans will include a suggestion for a nonclustered index to improve query performance.

Query Store

When the Query Store is enabled, it will capture query execution plans—including any missing index suggestions—for later review and action.

For more information on the Query Store, see the topic *Monitoring Performance By Using the Query Store* in Microsoft Docs:

Monitoring Performance By Using the Query Store

http://aka.ms/rqkfgg

Database Engine Tuning Advisor

The Database Engine Tuning Advisor can make index suggestions based on a workload derived from a SQL Profiler trace, the query plan cache, or a file of Transact-SQL statements.

For more information on the Database Engine Tuning Advisor, see the topic *Start and Use the Database Engine Tuning Advisor* in Microsoft Docs:

Start and Use the Database Engine Tuning Advisor

http://aka.ms/rcxhfe

DMVs

Several system DMVs provide a view of the missing index information collected by SQLOS. The information presented by these DMVs is used as the basis for missing index suggestions by the other tools mentioned in this topic.

For more details of the missing index DMVs, see the topic *Index Related Dynamic Management Views and Functions (Transact-SQL)* in Microsoft Docs. Missing index DMVs have names beginning **sys.dm_db_missing_index...**.

Index Related Dynamic Management Views and Functions (Transact-SQL)

http://aka.ms/vched5

When using any of these tools, you should beware that they have some limitations. They will not:

- Suggest clustered indexes.
- Suggest modifications to existing indexes—new indexes will always be suggested.
- Properly analyze column order in multicolumn indexes.
- Suggest filtered indexes.

You should closely analyze missing index suggestions before implementing them.

Demonstration: Picking the Right Index Key

In this demonstration, you will see:

- The effects of selecting different data types as a clustered index key.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **Demo.ssmssln** solution, in Solution Explorer, open the **Demo 2 - indexes.sql** script file.
3. Execute the code under the comment that begins **Step 1** to use the **AdventureWorks** database.
4. Execute the code under the comment that begins **Step 2** to create a table with a GUID clustered primary key.
5. Execute the code under the comment that begins **Step 3** to insert 10,000 rows into the new table.
6. Execute the code under the comment that begins **Step 4** to view index fragmentation.
7. Execute the code under the comment that begins **Step 5** to view the page sequence of the new table.
8. Execute the code under the comment that begins **Step 6** to create a table with an integer IDENTITY primary key.
9. Execute the code under the comment that begins **Step 7** to insert 10,000 rows into the new table.
10. Execute the code under the comment that begins **Step 8** to view index fragmentation.
11. Execute the code under the comment that begins **Step 9** to view the page sequence of the new table; fragmentation is much lower.
12. Leave SSMS open for the next demonstration.

Check Your Knowledge

Question	
What does the abbreviation "HoBT" stand for?	
Select the correct answer.	
	Heap or bulk table
	Heap or b-tree
	Head of block table
	Highly organized block tree

Lesson 3
Columnstore Indexes

The indexes discussed so far in this module are row-based indexes; until SQL Server 2012, this was the only type of index supported by SQL Server. In this lesson, you will learn about column-oriented indexes, called columnstore indexes.

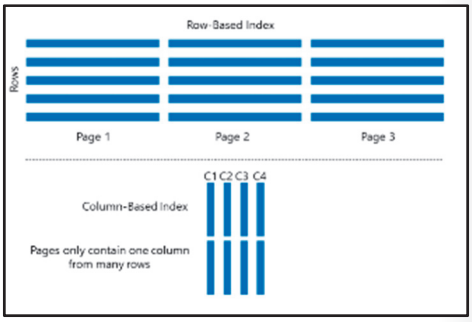
Lesson Objectives

At the end of this lesson, you will be able to:

- Explain how columnstore indexes differ from traditional row-based indexes.
- Describe the features of columnstore indexes.
- Understand how columnstore indexes work.

What Is a Columnstore Index?

Columnstore indexes are designed to increase performance when querying very large tables, typically in decision support and data warehouse systems. As you have learned, row-based clustered indexes are stored on disk in pages; each page contains a number of rows, and includes all the associated columns with each row. Columnstore indexes also store data in pages, but they store all the values for one column in a group of pages.



Consider a data warehouse containing fact tables that are used to calculate aggregated data across multiple dimensions. These fact tables might consist of many rows, perhaps numbering 10s of millions.

Using a code example:

Totaling Sales Orders by Product

```
SELECT ProductID
            ,SUM(LineTotal) AS ProductTotalSales
FROM Sales.OrderDetail
GROUP BY ProductID
ORDER BY ProductID;
```

In this example, if the **Sales.OrderDetail** table has a row-based clustered index on **OrderDetailID**, the database engine will need to read every leaf-level page with a scan operation to aggregate the values for **ProductID** and **LineTotal**; for a large table, this could mean many millions of logical reads. With a column-based index, the database engine needs only to read the pages associated with the two referenced columns, **ProductID** and **LineTotal**. This makes columnstore indexes a good choice for large data sets.

Using a columnstore index can significantly improve the performance for a typical data warehouse query; by up to 10 times. This gain derives from two characteristics of columnstore indexes:

- **Storage**. Columnstore indexes store data in a compressed columnar data format instead of by row. This achieves compression ratios of seven times greater than a standard row-based table.
- **Batch mode execution**. Columnstore indexes process data in batches (of 1,000-row blocks) instead of row by row. Depending on filtering and other factors, a query might also benefit from "segment elimination," which involves bypassing million-row chunks (segments) of data and further reducing I/O.

Columnstore indexes perform well for scan operations of large data sets because:

- Columns often store matching data—for example, a set of states, enabling the database engine to compress the data better. This compression can reduce or eliminate any I/O bottlenecks in your system, while also reducing the memory footprint as a whole.
- High compression rates improve overall query performance because the results have a smaller in-memory footprint.
- Instead of processing individual rows, batch execution also improves query performance. This can typically be a performance improvement of around two to four times, because processing is undertaken on multiple rows simultaneously.
- Aggregation queries often select only a few columns from a table, which reduces the number of physical reads required to retrieve data from I/O storage.

 **Note:** Nonclustered and clustered indexes are supported in Azure SQL Database V12 Premium Edition. For more information about the columnstore features supported by different versions of SQL Server, see the topic *Columnstore Indexes Versioned Feature Summary* in Microsoft Docs.

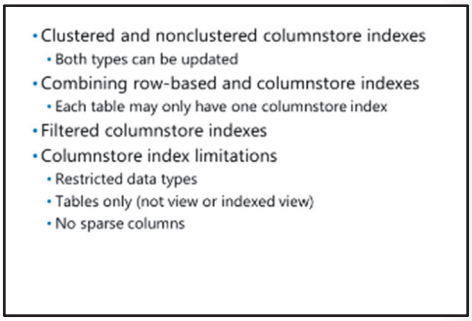
 **Columnstore Indexes Versioned Feature Summary**

http://aka.ms/uzm5ac

 **Best Practice:** Columnstore indexes are most effective on tables with many millions of rows, where the column values have low selectivity and can be grouped into sets that can be effectively compressed. Columnstore indexes are not suitable for tables with many highly-selective or unique data values.

Columnstore Index Features

In SQL Server, columnstore indexes can be used in several different ways.



Clustered and Nonclustered Indexes

As with row-based indexes, columnstore indexes may be either clustered or nonclustered.

- A clustered columnstore index causes all the columns in the table to be stored in columnar format. Unlike a row-based clustered index, there is no explicit sequence to the rows—data is stored in an order that optimizes columnstore compression.
- A nonclustered columnstore index creates a copy of some or all of the columns in a table and stores the data in a columnar format. A nonclustered columnstore index cannot be made up of more than 1,024 columns.

Columns contained in a clustered or nonclustered columnstore index can be updated.

In-memory optimized tables may be defined with a columnstore clustered index.

Note: SQL Server supports updatable clustered and nonclustered columnstore indexes. Previous versions of SQL Server do not support the same range of functionality. SQL Server 2012 supports read-only nonclustered columnstore indexes; SQL Server 2014 supports read-only nonclustered columnstore indexes and updatable clustered columnstore indexes.

Combining Row-Based and Columnstore Indexes

SQL Server supports using a mixture of columnstore and row-based indexes on the same table. It is possible to add:

- Nonclustered row-based indexes to a table with a columnstore clustered index.
- Nonclustered columnstore indexes to a table with a row-based clustered index.

The restriction that a table may only have one clustered index still applies; a table can have a columnstore or a row-based clustered index, but not both.

Using this feature, you can benefit from the features of both columnstore and row-based indexes for different query workloads, at the cost of additional storage space and CPU resources. Use this feature to enforce unique constraints (such as a primary key) on a table with a clustered columnstore index.

You can convert a table with a columnstore clustered index to a row-based clustered index, and vice versa.

Each table can only have one columnstore index.

Filtered Columnstore Indexes

You can create nonclustered columnstore indexes with a filtering WHERE clause. The rules and restrictions for filters on columnstore indexes are similar to those that apply to filtered row-based indexes, which you learned about in the previous lesson.

Use this feature to create an index on only the cold data of an operational workload. This will greatly reduce the performance impact of having a columnstore index on an online transaction processing (OLTP) table.

For more information about the features of columnstore indexes, see the topic *Columnstore Indexes Guide* in Microsoft Docs:

Columnstore Indexes Guide

http://aka.ms/kheieo

Columnstore Index Limitations

Restrictions apply to the use of columnstore indexes. These include:

- A columnstore index cannot contain any of the following data types:
  - **ntext**, **text**, and **image**
  - **varchar(max)** and **nvarchar(max)**
  - **rowversion** and **timestamp**
  - **sql_variant**
  - CLR types (**hierarchyid**, **geography**, and **geometry**)
  - **xml**
- A columnstore index cannot be created on a view or indexed view.
- A columnstore index cannot contain sparse columns.
- A columnstore index cannot participate in replication.

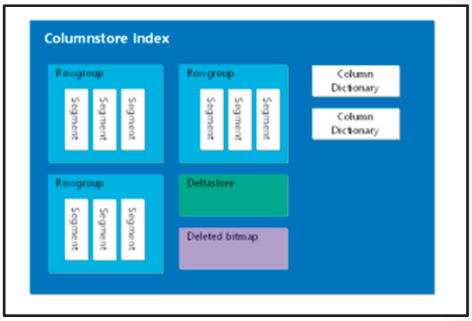
For full details of the limitations of columnstore indexes, see the topic *CREATE COLUMNSTORE INDEX (Transact-SQL) – Limitations and Restrictions* in Microsoft Docs:

CREATE COLUMNSTORE INDEX (Transact-SQL) - Limitations and Restrictions

http://aka.ms/fwkobx

Columnstore Index Internals

As you might expect from their different behavior, the internal structure of columnstore indexes is unlike the internal structure of row-based indexes.



Rowgroup

A columnstore index is composed of one or more rowgroups. A rowgroup contains the compressed columnar values for a subset of rows in the index—each rowgroup can contain up to 1,048,576 rows. The maximum size of a rowgroup is selected to be an optimal trade-off between rate of compression and a size suitable for in-memory operation.

You can view details of the rowgroups in a database using the **sys.column_store_row_groups** system view.

sys.column_store_row_groups

```
SELECT * FROM sys.column_store_row_groups;
```

For more information on **sys.column_store_row_groups**, see the topic *sys.column_store_row_groups* (Transact-SQL) in Microsoft Docs:

sys.column_store_row_groups (Transact-SQL)

http://aka.ms/u3m015

Segment

Each rowgroup is made up of a group of column segments; one segment for each column in the index. A segment contains all the values for a specific column in a rowgroup.

You can view details of all the segments in a database using the **sys.column_store_segments** system view:

sys.column_store_segments

```
SELECT * FROM sys.column_store_segments;
```

A statistics object is not maintained for a columnstore index—running the DBCC SHOW_STATISTICS command for a columnstore index returns an empty result set. The segment metadata found in **sys.column_store_segments** provides some information about the data values in each segment.

For more information on **sys.column_store_segments**, see the topic *sys.column_store_segments (Transact-SQL)* in Microsoft Docs:

sys.column_store_segments (Transact-SQL)

http://aka.ms/an5w0a

Dictionary

Some data types require encoding and decoding when their values are written to or retrieved from a columnstore index segment. Dictionaries are maintained at column level to improve the performance of this process. The primary dictionary applies to all of a column's segments; other secondary dictionaries may be maintained for individual column segments.

Details of all dictionaries in a database can be found in the system view **sys.column_store_dictionaries**:

sys.column_store_dictionaries

```
SELECT * FROM sys.column_store_dictionaries;
```

For more information about **sys.column_store_dictionaries**, see the topic *sys.column_store_dictionaries (Transact-SQL)* in Microsoft Docs:

sys.column_store_dictionaries (Transact-SQL)

http://aka.ms/eajfnx

Deltastore

When new rows are added to a columnstore index using INSERT operations, existing column segments are not modified. Instead, modifications are recorded in an uncompressed rowgroup called a deltastore. Inserts are recorded in the deltastore until it reaches the maximum rowgroup size (1,048,576 rows), at which point it is closed, converted to segments, and compressed by the tuple-mover process.

When a columnstore index is queried, the contents of the deltastore are combined with the contents of the relevant segments to produce the query result.

Bulk inserts of greater than 102,400 rows bypass the deltastore and are written directly to new compressed rowgroups.

You can view deltastores in the output of **sys.column_store_row_groups**; they have a non-NULL **delta_store_hobt_id** value:

Viewing Deltastores

```
SELECT * FROM sys.column_store_row_groups WHERE delta_store_hobt_id IS NOT NULL;
```

Deleted Bitmap

When rows are deleted from a columnstore index using DELETE operations, existing column segments are not modified. Instead, deleted rows are recorded in a b-tree index called the deleted bitmap. Rows with an entry in the deleted bitmap are ignored when you query a columnstore index.

If a deleted row is found in the deltastore, it is removed without being added to the deleted bitmap.

UPDATE operations against columnstore indexes are processed as a DELETE followed by an INSERT.

When a columnstore index is rebuilt, rows in the deleted bitmap are permanently deleted and the deleted bitmap is cleared.

You can view deleted bitmaps in the output of **sys.internal_partitions**:

Viewing Deleted Bitmaps

```
SELECT * FROM sys.internal_partitions
WHERE internal_object_type_desc = 'COLUMN_STORE_DELETE_BITMAP';
```

For more information on **sys.internal_partitions**, see the topic *sys.internal_partitions (Transact-SQL)* in Microsoft Docs:

sys.internal_partitions (Transact-SQL)

http://aka.ms/t38mib

Demonstration: Implementing a Columnstore Index

In this demonstration, you will see:

- How to implement a clustered columnstore index.
- Columnstore index internals.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. If it is not already running, start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine instance using SQL Server authentication.
3. If it is not already open, open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod06\Demo** folder. If the Solution Explorer pane is not visible, on the **View** menu, click **Solution Explorer**.
4. Open the **Demo 3 - columnstore.sql** script file.
5. Execute the code under the comment that begins **Step 1** to use the **AdventureWorks** database.
6. Execute the code under the comment that begins **Step 2** to create a table with a clustered columnstore index.
7. Execute the code under the comment that begins **Step 3** to add 10 rows to the table.
8. Execute the code under the comment that begins **Step 4** to examine the table rowgroups.
9. Execute the code under the comment that begins **Step 5** to insert 1.1 million rows in blocks of 1,000 rows. This will close the deltastore.
10. Execute the code under the comment that begins **Step 6** to examine the table rowgroups again.
11. Execute the code under the comment that begins **Step 7** to bulk insert 2 million rows into the table in a single step. This will bypass the deltastore.
12. Execute the code under the comment that begins **Step 8** to examine the rowgroups again.
13. Execute the code under the comment that begins **Step 9** to examine the rowgroup segments.
14. Execute the code under the comment that begins **Step 10** to delete 10 rows from the table.
15. Execute the code under the comment that begins **Step 11** to examine the deltastore and deleted bitmap.
16. Execute the code under the comment that begins **Step 12** to drop the table.
17. Close SSMS without saving changes to any files.

Check Your Knowledge

Question	
On which of the following table types could you not create a nonclustered columnstore index?	
Select the correct answer.	
	A heap
	A table with a row-based clustered index
	An in-memory optimized table without any indexes
	A table with a columnstore clustered index
	A temporary table

Lab: Statistics and Index Internals

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler and retailer of cycle products. The owners have decided to start a new direct marketing arm of the company. It has been created as a new company named Proseware Inc. Even though it has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that is capable of supporting both the existing workload and the workload from the new company.

While investigating the general slow speed of the new SQL Server instance, you came across a few workloads with poor execution performance, due to cardinality estimation issues and inappropriate indexes. In this lab, you will improve the performance of those workloads by fixing cardinality estimation errors and creating the correct indexes, including columnstore indexes.

Objectives

After completing this lab, you will be able to:

- Identify and fix cardinality estimation errors.
- Identify and review the indexing strategy.
- Create columnstore indexes.

Estimated Time: 90 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Fixing Cardinality Estimation Errors

Scenario

While investigating the general slow speed of the new SQL Server instance, you came across a few workloads that had cardinality estimation issues. In this exercise, you will fix cardinality estimation error for this type of workload and improve the performance.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Run the Workload
3. List Statistics Objects
4. Examine Statistics in Detail
5. Update Statistics
6. Rerun the Workload

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** as Administrator in the **D:\Labfiles\Lab06\Starter** folder.

▶ Task 2: Run the Workload

1. In the **D:\Labfiles\Lab06\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell™. If a message is displayed asking you to confirm a change in execution policy, type **Y**. Wait a few minutes for the workload to finish.
2. Note the elapsed time reported by the script.

▶ Task 3: List Statistics Objects

1. Start **SQL Server Management Studio (SSMS)** and connect to the **MIA-SQL** instance using Windows authentication.
2. Open the project file **D:\Labfiles\Lab06\Starter\Project\Project.ssmssln** and the script file **Lab Exercise 01 - Cardinality.sql**.
3. Execute the query under the comment that begins **Task 2** to list the statistics objects for the **Proseware.WebResponse**, **Proseware.Campaign**, and **Proseware.CampaignAdvert** tables in the **AdventureWorks** database.
4. Do any of the statistics look like they might be out of date?

▶ Task 4: Examine Statistics in Detail

1. Under the comment that begins **Task 3**, amend and then execute the first query to display the detailed statistics information for the **IX_WebResponse_CampaignAdvertID** statistics object linked to **Proseware.WebResponse**. How many rows does the table contain, according to the statistics?
2. Execute the second query to find the actual number of rows in the **Proseware.WebResponse** table. Do these results suggest that the query might be subject to a cardinality estimation error?

▶ Task 5: Update Statistics

1. Under the comment that begins **Task 4**, amend and then execute the first query to update all the statistics objects linked to **Proseware.Webresponse** by sampling all of the rows in the table.
2. Execute the second query under the comment that begins **Task 4** to examine the new statistics for the **IX_WebResponse_CampaignAdvertID** statistics object.
3. Amend and then execute the third query under the comment that begins **Task 4** to update all the statistics objects linked to **Proseware.CampaignAdvert** using 50 percent of the rows in the table.

▶ Task 6: Rerun the Workload

1. In the **D:\Labfiles\Lab06\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell. Wait for the workload to finish before continuing.
2. Compare the elapsed time reported by the script to the elapsed time reported in the first step of this exercise.

Results: At the end of this lab, statistics in the **AdventureWorks** database will be updated.

Exercise 2: Improve Indexing

Scenario

While investigating the general slow speed of the new SQL Server instance, you came across a few workloads that did not have the correct indexes in the underlying tables. In this exercise, you will modify existing indexes to improve the performance of the workload.

The main tasks for this exercise are as follows:

1. Execute the Workload
2. Examine Existing Indexes
3. Use the Database Engine Tuning Advisor
4. Implement a Covering Index
5. Rerun the Workload

▶ Task 1: Execute the Workload

- Execute the Transact-SQL script in the **Lab Exercise 02 - Workload.sql** file.

▶ Task 2: Examine Existing Indexes

1. In SSMS Solution Explorer, open the script file **Lab Exercise 02 - Indexing.sql**.
2. Under the comment that begins **Task 2**, write a query to return details of the indexes on the **Proseware.WebResponse** table in the **AdventureWorks** database.

▶ Task 3: Use the Database Engine Tuning Advisor

1. In SSMS, start the Database Engine Tuning Advisor from the **Tools** menu, and connect to the **MIA-SQL** instance using Windows authentication.
2. Run a performance analysis session against the **AdventureWorks** database using the file **D:\Labfiles\Lab06\Starter\Project\Project\ Lab Exercise 02 - Workload.sql** as a workload file.
3. When the analysis completes, examine the definition of the index suggested for the **Proseware.WebResponse** table, but do not implement it. Close the Database Engine Tuning Advisor when you are finished. Is the suggested index similar to any of the existing indexes on the table?

▶ Task 4: Implement a Covering Index

1. In SSMS, in the **Lab Exercise 02 - Indexing.sql** file, under the comment that begins **Task 5**, execute the first query to drop the index **IX_WebResponse_log_date_CampaignAdvertID**.
2. Amend and then execute the second query under the comment that begins **Task 5** to create a new index with the key columns:
    - **log_date**
    - **CampaignAdvertID**
    - **browser_name**
3. and include: **page_visit_time_second**

▶ Task 5: Rerun the Workload

1. In SSMS, execute the script in the **Lab Exercise 02 - Workload.sql** file.
2. Return to the **Lab Exercise 02 - Indexing.sql** file, and under the comment that begins **Task 6**, execute the query against the system DMV **sys. dm_db_index_usage_stats** to verify that the new index was used.

Results: At the end of this exercise, the indexing of the **Proseware.WebResponse** table in the **AdventureWorks** database will be improved.

Exercise 3: Using Columnstore Indexes

Scenario

The **Proseware.WebResponse** table is expected to grow significantly in the future. Because the table will be used in many range-based analytical queries, you decide to create a nonclustered columnstore index on some of the columns in the table. You will also create a new table, **Proseware.Demographic**, to hold a very large dataset; this table will have a clustered columnstore index.

The main tasks for this exercise are as follows:

1. Add a Nonclustered Columnstore Index to a Row-Based Table
2. Create a Table with a Clustered Columnstore Index
3. Add a Nonclustered Row-Based Index to a Table with a Clustered Columnstore Index

▶ Task 1: Add a Nonclustered Columnstore Index to a Row-Based Table

1. In SSMS Solution Explorer, open the script file **Lab Exercise 03 - Columnstore.sql**.
2. Under the comment that begins **Task 1**, amend and then execute the query to add a nonclustered columnstore index to **Proseware.WebResponse** in the **AdventureWorks** database. The columnstore index should cover the following columns:
   - log_date
   - page_url
   - browser_name
   - page_visit_time_seconds
3. Name the index **IX_NCI_WebResponse**.

▶ Task 2: Create a Table with a Clustered Columnstore Index

1. Under the comment that begins **Task 2,** amend and then execute the query to create a table called **Proseware.Demographic** with a clustered columnstore index called **PK_Proseware_Weblog**.
2. Execute the second statement under the comment that begins **Task 2** to add a single row to **Proseware.Demographic**.
3. Query the table to verify that one row has been inserted.

▶ Task 3: Add a Nonclustered Row-Based Index to a Table with a Clustered Columnstore Index

1. Under the comment that begins **Task 3**, amend and then execute the first query to add a nonclustered unique index to **Proseware.Demographic**. Call the index **IX_Demographic_DemographicID**. The index key should be **DemographicID**.
2. Execute the second statement under the comment that begins **Task 3** to rerun the INSERT statement from the previous step. Notice that the nonclustered index prevents you from inserting duplicate data.

Results: At the end of this exercise, the **Proseware.WebResponse** in the **AdventureWorks** database will have a nonclustered columnstore index. A new table—**Proseware.Demographic**—will be created with a clustered columnstore index.

Check Your Knowledge

Question	
When AUTO_UPDATE_STATISTICS is on, approximately what percentage of a table's data must change to prompt a statistics update?	
Select the correct answer.	
	5 percent
	10 percent
	15 percent
	20 percent
	25 percent

Module Review and Takeaways

In this module, you have learned about how statistics are used to estimate the cardinality of result sets returned by SQL Server. You have learned how to create, view, and update statistics objects.

You have learned about the internal structure of heaps, clustered indexes, and nonclustered indexes, and about how index design can affect query performance.

You have also learned about the internal structure of columnstore indexes, how they differ from row-based indexes, and about circumstances where a columnstore index is an appropriate choice.

Categorize Activity

Categorize each factor by the index type that best addresses it. Indicate your answer by writing the category number to the right of each item.

Items	
1	Highly selective data
2	Data with low selectivity
3	Many queries for single values
4	Many aggregate and range queries
5	Replication will be used
6	Large data volume (millions or billions of rows)

Category 1		Category 2
Row-based index		Columnstore index

Module 7

Query Execution and Query Plan Analysis

Contents:

Module Overview

For any non-trivial query run against a Microsoft® SQL Server® instance, there are many different sequences of operation which will lead to the result. A subcomponent of the SQL Server Database Engine, the query optimizer, is responsible for selecting which sequence of operations to use to satisfy a query; this sequence of operations is referred to as the query execution plan.

An understanding of how the query optimizer selects a query execution plan, and how to interpret query execution plans, can be invaluable when investigating issues of query performance.

This module covers query execution and query plan analysis. It focuses on architectural concepts of the query optimizer and how to identify and resolve query plan issues.

Objectives

After completing this module, students will be able to:

- Describe query optimizer internals.
- Capture query execution plans.
- Analyze, identify, and fix query execution plan issues.

Lesson 1
Query Execution and Query Optimizer Internals

This lesson introduces query optimizer internals. The query optimizer is an important SQL Server component that provides an optimal query execution plan for the queries. A better understanding of query optimizer internals can help you improve the performance of the database and applications.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe logical query processing phases.
- Describe physical query processing phases.
- Explain the query optimizer and optimization phases.

Logical Phases of Query Processing

Statement Clause	Processing Sequence
SELECT	8
DISTINCT <select_list>	9
FROM <left_table>	1
<join_type> JOIN <right_table>	3
ON <join_condition>	2
WHERE <where_condition>	4
GROUP BY <group_by_list>	5
WITH { CUBE | ROLLUP }	6
HAVING <having_condition>	7
ORDER BY <order_by_list>	10
<TOP_specification>	11

Transact-SQL is a *declarative* language. This means that a Transact-SQL statement describes the logic of a query, but does not specify the sequence of individual operations that will be used to apply that logic; the clauses of a Transact-SQL statement will not necessarily be processed in the sequence that they appear in the query.

Note: The opposite of a declarative language is an *imperative* language. Many common programming languages, such as Visual C++, Visual Basic .NET, and Ruby are imperative languages. Imperative programming language code defines the sequence of operations required to define the program's logic. Commands will be executed in the sequence that they appear in the code, following the programming language rules for flow control.

Despite Transact-SQL including some imperative statements for flow control—for example IF…THEN and WHILE—most Transact-SQL statements, such as SELECT, INSERT, and UPDATE, are declarative. It is the job of the database engine to interpret Transact-SQL statements and determine the sequence of operations required to carry out the statement's logic.

Logical Processing Example

The following code is a generalized example of a Transact-SQL SELECT statement:

The SELECT statement is declarative

SELECT statement syntax example

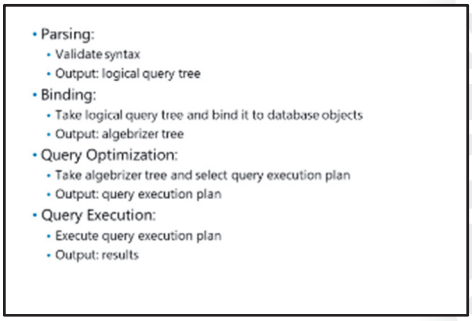
```
SELECT DISTINCT <TOP_specification> <select_list>
FROM <left_table>
<join_type> JOIN <right_table>
ON <join_condition>
WHERE <where_condition>
GROUP BY <group_by_list>
WITH { CUBE | ROLLUP }
HAVING <having_condition>
ORDER BY <order_by_list>
```

The clauses of this code are processed in the following order. The output of each step in the sequence is used as the input to the next step:

```
1. FROM <left_table>
2. ON <join_condition>
3. <join_type> JOIN <right_table>
4. WHERE <where_condition>
5. GROUP BY <group_by_list>
6. WITH { CUBE | ROLLUP }
7. HAVING <having_condition>
8. SELECT
9. DISTINCT <select_list>
10. ORDER BY <order_by_list>
11. <TOP_specification>
```

Physical Phases of Query Processing

When a Transact-SQL statement is submitted to a database engine instance, it undergoes the following processes:



1. Parsing
2. Binding
3. Query Optimization
4. Query Execution

Parsing

A subcomponent of the relational engine, the command parser validates the statement's syntax and parses it into a logical query tree. The sequence of actions in the logical query tree will correspond to the logical phases of query processing discussed in the previous topic. If a syntax error is discovered in the statement, an *invalid syntax* error message is raised.

The output of the parsing phase is the logical query tree, also referred to as the parse tree.

Binding

A component called the algebrizer takes the query tree generated by the parsing phase, and attempts to link references in the query tree to database objects in a process known as binding. As part of the binding process, the permissions of the security context under which the statement is executed are tested to confirm that the user has permission to access the relevant database objects.

If any of the objects referenced in the query do not exist, or the user does not have sufficient permissions to access them, an *invalid object name* error will be raised.

The output of the binding phase is the algebrizer tree.

Query Optimization

The query optimizer component takes the algebrizer tree and uses it, together with the database schema and database object statistics, to select a query execution plan. The query optimizer and query execution plans are the subject of the rest of this module.

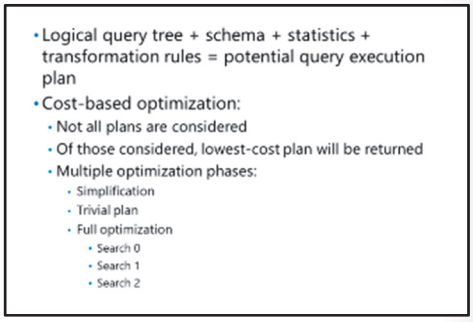
Query Execution

The query execution plan selected by the query optimizer is put into action by the Query Executor, and results returned.

The Query Optimizer

As discussed in the previous topic, the query optimizer receives a logical query tree bound to database objects from the algebrizer. Using the list of database objects in the algebrizer tree, the query optimizer collects the following information for each object:

- **Object schema**. For instance, column data types, indexes, and constraints.
- **Object statistics**. Information about the number of rows in tables and indexes, and distribution of data values within table and index columns.



Additional Reading: For more information about database object statistics, see Module 6 of this course, *Statistics and Index Internals*.

Using the available data, the query optimizer then selects a query execution plan.

Optimization Phases

The query optimizer goes through several phases designed to optimize the queries as quickly as possible. These optimization phases are designed to avoid more expensive and more complex options, unless they are really necessary. These phases are as follows:

- **Simplification**. Reduces the query to a simpler form to make optimization quicker. Simplifications include:
  - Converting subqueries to joins.
  - Removing redundant joins.
- **Trivial Plan Generation**. A trivial plan is a logical query plan for which very few possible query execution plans exist. For queries of this kind, the query optimizer will use a single plan instead of spending any time to pick an optimal plan.
- **Full Optimization**. Full cost-based optimization (detailed here).

Cost-Based Optimization

The query optimizer is a cost-based optimizer; it will generate a group of query execution plans that could satisfy the requirements of the logical query plan, and assign costs to them, based on the available database object statistics. The query optimizer will calculate costs for several plans and return the query execution plan that has the lowest cost, from the plans it has evaluated.

The more complex the Transact-SQL statement, the more potential query execution plans could satisfy the logical query plan. The query optimizer does not always evaluate the cost of every potential query execution plan; this is because each plan evaluation consumes CPU and memory, and takes time to create. Evaluating all potential plans could take too long or consume too many resources. Instead, the query optimizer finds a balance between the optimization time and the quality of the plan; it may select a suboptimal plan for a query if it finds that running a suboptimal plan will take less time than evaluating many plans.

Transformation Rules

To select which potential query execution plans to evaluate, and to calculate a cost for those plans, the query optimizer uses a set of transformation rules. Transformation rules associate logical operations in the logical query tree with the physical operators that can implement them; physical operators are the steps that make up a query execution plan.

Each operator in the logical query tree can be associated with one or more physical operators. For instance, a logical join might be implemented by one of three physical operators—a nested loop join, a merge join, or a hash join (physical join operators are discussed in more detail later in this module).

The combinations of the logical query tree with transformation rules are stored in a Memo—an internal data structure.

Full Optimization Searches

Some queries may have a large number of possible query plans and it would take a long time to explore them. Therefore, in addition to applying transformation rules, the query optimizer uses heuristics to control the search strategy.

Full optimization is performed in three stages. The optimization process finishes if a good enough plan is found at the end of any stage. However, if the plan is still expensive (when compared to the query optimizer's internal threshold) the next stage is processed. If a plan has not been returned by the earlier stages, the lowest-cost query execution plan identified at the end of the last stage will be returned, even if it is considered expensive, based on the query optimizer's internal threshold.

The stages are:

- Search 0, also known as the *transaction processing phase*.
- Search 1, also known as the *quick plan phase*.
- Search 2, also known as the *full optimization phase*.

Each search considers more transformation rules and uses more processing resources than the last.

Demonstration: Analyzing Query Optimizer Internals

In this demonstration, you will see methods for examining query optimizer internals.

Demonstration Steps

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are running and log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Demofiles\Mod07** folder as Administrator.
3. In the **User Account Control** dialog box, click **Yes**, wait until the script completes, and then press any key.

4. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine instance using Windows authentication.
5. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod07** folder.
6. In Solution Explorer, double-click the **Demo 1 - query optimizer.sql** script file.
7. Select the query under the comment that begins **Module 7 - Demo 1**, and click **Execute**.
8. Select the query under the comment that begins **Step 1** and on the **Query** menu, click **Display Estimated Execution Plan**. Note that no reference to **Sales.SalesOrderHeader** appears in the plan.
9. Select the query under the comment that begins **Step 2** and on the **Query** menu, click **Display Estimated Execution Plan**. The plan is simplified because a check constraint prevents any data from matching the filter.
10. On the **Query** menu, click **Include Actual Execution Plan**, and then select the query under the comment that begins **Step 3**, and click **Execute**.
11. On the **Execution plan** tab, click the **SELECT** operator, and then in the Properties pane, note that **Optimization Level** is **TRIVIAL**.
12. Select the query under the comment that begins **Step 4**, and click **Execute**. This returns a result set showing all the transformation rules used in producing the query plan.
13. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
At which of the following optimization phases will a plan be selected for a simple query, such as **SELECT * FROM Sales.Customer**?	
Select the correct answer.	
	Simplification
	Trivial plan
	Full optimization search 0
	Full optimization search 1
	Full optimization search 2

Lesson 2
Query Execution Plans

In the previous lesson, you learned about how query execution plans are selected by the query optimizer. In this lesson, you will learn about the different types of query execution plans, and different methods to capture and view query execution plans, so that you can analyze them.

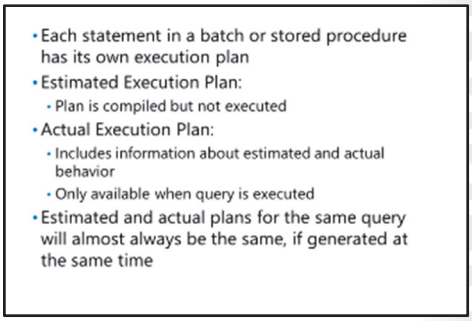
Lesson Objectives

At the end of this lesson, you will be able to:

- Explain the difference between estimated and actual query execution plans.
- Describe the different ways of viewing a query execution plan.
- Capture query execution plans in different ways.

Estimated Execution Plans and Actual Execution Plans

Execution plans show the step-by-step description of how a query is executed or should be executed. Each statement in a Transact-SQL batch or stored procedure will have its own execution plan.



As discussed in the previous lesson, the query optimizer processes a query to generate an optimal query execution plan. The execution plan is then passed to the storage engine where the query is actually executed, according to the execution plan. Query execution plans can be accessed in two forms:

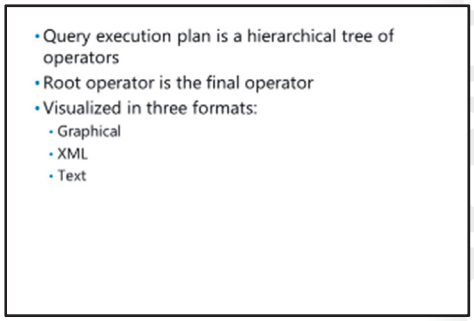
- **Estimated execution plan**: An estimated execution plan is generated by passing the query through the query optimizer but not executing it. This is the plan that SQL Server will most probably use to execute the query. Estimated execution plans can be useful for:
  - Inspecting a plan for a long running query without actually executing the query.
  - Displaying a plan for a query that modifies data (for example, an UPDATE query) without changing data.
- **Actual execution plan**: When an actual execution plan is requested, the query is executed and the plan is returned along with the query result. This is the plan that SQL Server uses to execute the query. The actual execution plan is useful when you have to know how a query actually performs. The actual execution plan contains the same information as appears in the estimated execution plan. It also contains performance information such as parameter values, actual row counts, elapsed time, CPU consumption, and memory consumption.

The estimated and actual execution plans for a given query, if generated at the same time, will almost always show the same plan; however, to be absolutely certain of the plan that a query uses, you must collect the actual execution plan.

Note: An actual execution plan can only be collected at the time a query is executed. When troubleshooting query performance issues, you will often be attempting to diagnose and address problems after they have occurred, in which case the actual execution plan used at the time the problem was reported will not be available.
Executing a query again later, to retrieve the actual execution plan, is not guaranteed to give you the same execution plan. Table or index statistics may have changed, which might result in a new execution plan.

Note: Comparing estimated and actual row counts in an actual execution plan can help you to identify areas where table statistics are out of date. The closer the estimated and actual row counts are, the more accurate your statistics.

Query Execution Plan Formats



A query execution plan consists of a hierarchical tree of query execution operators, linked by data flows. The operator at the top of the tree (the operator that has no parent operators) is the final execution step. All the other operators in the plan hierarchy feed data—directly or indirectly—to the final operator.

There are three formats in which query execution plans can be viewed. All three formats are capable of displaying the estimated execution plan or the actual execution plan for a Transact-SQL batch.

Graphical Format

A graphical view of a query execution plan can be displayed using SQL Server Management Studio (SSMS). The flow of data in a graphical query plan is from right to left, the left-most node being the final execution step. Query execution operators are represented as nodes in the diagram. The flow of data between operators is represented by arrows that connect the nodes to each other. The thickness of a connecting arrow represents the number of rows passed from one operator to another; a thicker arrow represents more rows. More detailed information about each operator and flow of data is available from each item's properties. Where two data flows meet as inputs to one operator, the upper data flow is processed first.

XML Format

In XML format, the tree of operators in the plan is represented as a structure of XML elements; the final execution step is the outermost element, with all the other operators in the query plan appearing as child elements of the outermost element. The relationship between operators on a branch of the plan tree is indicated by the further nesting of XML elements. Flow of data is from innermost elements to outermost elements. Detailed properties for each operator are included as XML attributes of the element associated with the operator.

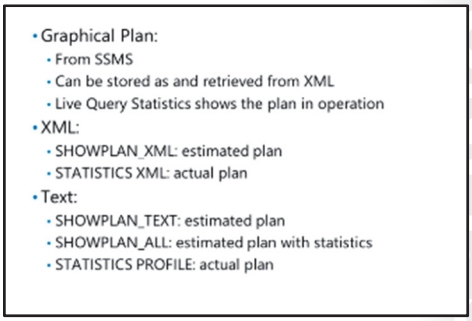
Text Format

In text format, the hierarchy of operators is represented as an indented list. Each operator occupies one row in the list, and line sequence and indentation is used to illustrate the hierarchical relationship between operators. The final operator appears at the top of the list and is not indented; other operators in the query plan appear on later lines and are indented to show their relationships. Pipe and dash characters are used to illustrate the connections between branches of the plan tree. Flow of data is from the most-indented lines to the least-indented lines. Detailed properties for each operator (such as row counts and data sizes) are not available in the text representation of the query plan.

Note: If you are new to SQL Server query plans, you will probably find the graphical execution plan format easier to understand than XML format or text format.
Less detail is available from query plans in text format; in general, you should use XML or graphical plans to access the most complete information.

Capturing Execution Plans

The methods for capturing a query execution plan vary by the format in which you wish to inspect the plan.



Note: Collecting query execution plans requires the SHOWPLAN permission, and permissions to objects referenced in the query. It is not possible to generate an execution plan for a query which references objects to which you do not have permission.

Graphical Plan

SSMS can display graphical query plans.

- To display the estimated execution plan, on the **Query** menu, click **Display Estimated Execution Plan** (Keyboard shortcut Ctrl + L). A graphical view of the estimated execution plan will be displayed on the **Execution plan** tab of the Results pane.
- To display the actual execution plan, on the **Query** menu, click **Include Actual Execution Plan** (Keyboard shortcut Ctrl + M). A graphical view of the actual execution plan will be displayed on the **Execution Plan** tab of the Results pane when the query is executed.

You can save graphical execution plans (right-click the graphical plan in the Results pane, then click **Save Execution Plan As**). The plan is automatically saved in XML format with a .sqlplan extension. By default, the .sqlplan extension is associated with SSMS, and opening an .sqlplan file from Windows Explorer will display the graphical plan in SSMS.

You can also view the XML plan from which the graphical plan is rendered in SSMS (right-click the graphical plan in the Results pane, then click **Show Execution Plan XML**). The format of the XML used for graphical query plans is similar, but not identical, to the XML produced when generating an XML format plan.

Live Query Statistics

Live Query Statistics offers a dynamic way to view the actual execution plan for a query. When Live Query Statistics is enabled (on the **Query** menu, click **Include Live Query Statistics**), counts of rows and other properties of the actual execution plan are displayed and updated in real time.

Note: Be aware that gathering the information required to run Live Query Statistics will place additional load on your SQL Server instance.

For more information on Live Query Statistics, see the topic *Live Query Statistics* in Microsoft Docs:

Live Query Statistics

http://aka.ms/yokck9

XML Plan

Any client that can execute Transact-SQL statements can also generate XML query execution plans by using the SET options.

An estimated execution plan in XML format can be generated with the SHOWPLAN_XML option:

SHOWPLAN_XML example

```
SET SHOWPLAN_XML ON;
GO
```

An actual execution plan in XML format can be generated with the STATISTICS XML option:

STATISTICS XML example

```
SET STATISTICS XML ON;
GO
```

When saved into a file with a .sqlplan extension, XML query plans can be opened as graphical plans with SSMS.

Text Plan

Any client that can execute Transact-SQL statements can also generate text query execution plans for Transact-SQL by using SET options.

An estimated execution plan in text format can be generated with the SHOWPLAN_TEXT option:

SHOWPLAN_TEXT example

```
SET SHOWPLAN_TEXT ON;
GO
```

An estimated execution plan in text format with a wider result set giving additional details, such as row counts and data volumes, can be generated with the SHOWPLAN_ALL option:

SHOWPLAN_ALL example

```
SET SHOWPLAN_ALL ON:
GO
```

An actual execution plan in text format, with a wider result set giving additional details, such as actual row counts and data volumes, can be generated with the STATISTICS PROFILE option:

STATISTICS PROFILE example

```
SET STATISTICS PROFILE ON;
GO
```

The methods for capturing query execution plans discussed so far all relate to plans for queries in a session over which you have control. You can also capture query execution plans for other sessions by using several different methods:

- SQL Profiler
- Extended Events
- Plan cache

Additional Reading: These methods are covered in more detail in Module 8 of this course, *Plan Caching and Recompilation*.

Demonstration: Capturing Query Execution Plans

In this demonstration, you will see methods for capturing an execution plan.

Demonstration Steps

1. In SQL Server Management Studio, in Solution Explorer, double-click the **Demo 2 - Capture Execution Plan.sql** script file.
2. Select the query under the comment that begins **Module 7 – Demo 2,** and click **Execute**.
3. Select the query under the comment that begins **Step 1** and on the **Query** menu, click **Display Estimated Execution Plan**. Click on each operator to examine its properties in the Properties pane.
4. Click the query pane, and then on the **Query** menu, click **Include Actual Execution Plan**.
5. Select the query under the comment that begins **Step 2**, and click **Execute**.
6. On the **Execution plan** tab, examine the execution plan. Click on each operator to examine its properties.
7. Right-click on the execution plan and click **Show Execution Plan XML** to review the XML plan.
8. In the **Demo 2 - Capture Execution Plan** query window, right-click the actual execution plan generated in the previous step, and select **Save Execution Plan As**. Save the plan as **D:\Demofiles\Mod07\demo2.sqlplan**.
9. On the **File** menu, point to **Open**, and then click **File**. In the **Open File** dialog box, select **D:\Demofiles\Mod07\demo2.sqlplan**, and then click **Open**. The plan will open in a new SSMS pane. Review the plan, and note that it was opened in a graphical format.
10. In the **Demo 2 - Capture Execution Plan** query window, on the **Query** menu, click **Include Actual Execution Plan**.
11. On the **Query** menu, click **Include Live Query Statistics**. Select the query under the comment that begins **Step 4**, and click **Execute**. Watch the live query statistics. This query will take some time to complete; you can stop it before it finishes when you are ready to move on.
12. On the **Query** menu, click **Cancel Executing Query**.
13. On the **Query** menu, click **Include Live Query Statistics** to disable Live Query Statistics.
14. Select the query under the comment that begins **Step 5**, and click **Execute** to demonstrate an XML estimated execution plan.
15. In the Results pane, click the XML result; it will open in a new SSMS pane as a graphical plan. Right-click the plan and click **Show Execution Plan XML** to review the XML.
16. In the **Demo 2 - Capture Execution Plan** query window, select the query under the comment that begins **Step 6**, and click **Execute** to demonstrate an XML actual execution plan.
17. In the Results pane, click the XML result; it will open in a new SSMS pane as a graphical plan. Right-click the plan and click **Show Execution Plan XML** to review the XML.
18. In the **Demo 2 - Capture Execution Plan** query window, select the query under the comment that begins **Step 7**, and click **Execute** to review a text estimated execution plan generated by SHOWPLAN.
19. Select the query under the comment that begins **Step 8**, and click **Execute** to review a text estimated execution plan generated by SHOWPLAN_ALL.
20. Select the query under the comment that begins **Step 9**, and click **Execute** to review a text actual execution plan.

21. Leave SSMS open for the next demonstration.

Categorize Activity

Place each SET option into the appropriate category. Indicate your answer by writing the category number to the right of each item.

Items	
1	SHOWPLAN_XML
2	STATISTICS XML
3	SHOWPLAN_TEXT
4	STATISTICS Profile
5	SHOWPLAN_ALL

Category 1		Category 2
Estimated Execution Plan		Actual Execution Plan

Lesson 3
Analyzing Query Execution Plans

In the previous lesson, you looked at methods for capturing query execution plans. This lesson will provide an introduction to interpreting and analyzing query execution plans, so that you can gain insight into how the database engine processes your queries, and use that information to troubleshoot query performance.

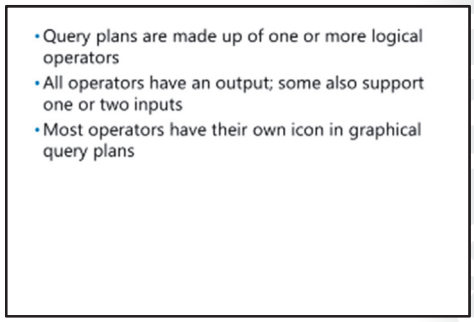
Lesson Objectives

At the end of this lesson, you will be able to:

- Identify and describe common query plan operators.
- Explain the difference between scan and seek operators.
- Describe the different join operators used in query execution plans.
- Identify query execution plans which use parallelism.
- Find and act on query execution plan warnings.
- Compare query plans using SSMS.

Query Execution Plan Operators

As you learned in the previous lesson, query execution plans are composed of one or more query plan operators, each of which corresponds to a logical transformation performed on a stream of data rows. There are more than 100 different operators which may appear in query execution plans. Every operator must have an output stream of data rows; some operators may have both an input and an output stream of data rows.
Operators can be characterized as having:



- One output data stream.
- Zero, one or two input data streams.

In graphical query plans, many (but not all) operators have their own icon to help you to distinguish between them.

Note: Because a query execution plan is a logical representation of a Transact-SQL statement, it is possible that multiple different Transact-SQL statements will generate the same sequence of operators in the related query plan.

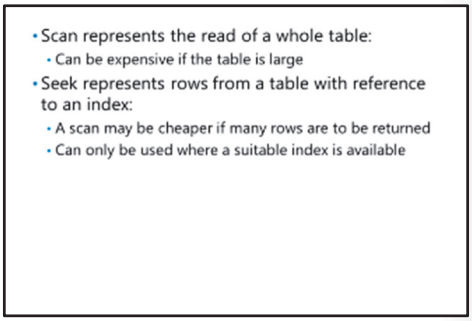
For a complete list of query plan operators, their meanings, and their icons when they appear in graphical plans, see the topic *Showplan Logical and Physical Operators Reference* in Microsoft Docs:

Showplan Logical and Physical Operators Reference

http://aka.ms/ifow8r

Data Retrieval Operators: Scan and Seek

All queries that retrieve data from a database table will have a query plan that has a scan or a seek operator as its ultimate source; this will be represented as the innermost operator in a text or XML plan, and the rightmost operator in a graphical query plan. Scan and seek operators represent the retrieval of data rows from storage; they have an output data stream and no input data stream.



There are several different kinds of scan operator and seek operator, but the core difference between scan and seek operators can be characterized as:

- A scan represents the process of reading all the data in a table.
- A seek represents the process of reading a subset of rows from a table by looking them up in an index.

Both scan and seek operators may output some or all of the rows they read, depending on the details of any filters applied in the query.

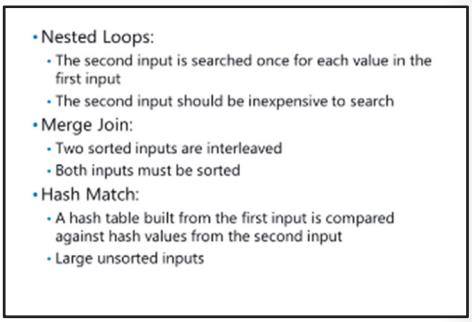
An individual seek operation will almost always be faster than an individual scan operation over the same table, especially when the volume of a data increases. However, there are cases when a single scan operation is faster than many iterations of a seek operation to return the same data.

Any query against the data in a table can be answered by a scan, whereas a seek is only available when a suitable index containing columns relevant to the filter, sometimes called a *covering index*, is available.

Note: When performance-tuning queries, do not expect to be able to turn every scan operation into a seek operation. When table statistics are accurate, the query optimizer will almost always correctly pick the faster operator.

Join Operators

When a query uses data from more than one table (by using a JOIN clause, for example), the query execution plan must include one or more join operators to combine the data returned by scans or seeks of the source tables into a single data stream suitable for output.



When building a query execution plan, the query optimizer can use one of three join operators, each of which takes two input data streams and produces one output data stream in the plan. Each join operator is optimal for different relative data volumes in the input streams.

Nested Loops

A nested loop join will perform a search from the second input data stream for each row in the first input data stream. This means that, in the scenario where the first input data stream has 1,000 rows, the second input will be searched once for each row—that is, 1,000 searches. In a graphical query plan, the upper input is the first input and the lower input is the second input. In an XML or text query plan, the second input will appear as a child of the first input.

Nested loop joins are optimal when the second input is inexpensive to search, either because it is small or because it has a suitable covering index for the search.

Merge Join

A merge join combines two sorted inputs by interleaving them. The sequence of the input streams has no impact on the cost of the join.

Merge joins are optimal when the input data streams are already sorted and are of similar volumes.

Hash Match

In a hash match, a hash table of values for each input data stream is calculated and the hash values are compared. The details of operation vary based on the details of the source query, but typically a complete hash table is calculated for the first input, then the hash table is searched for individual values from the second input.

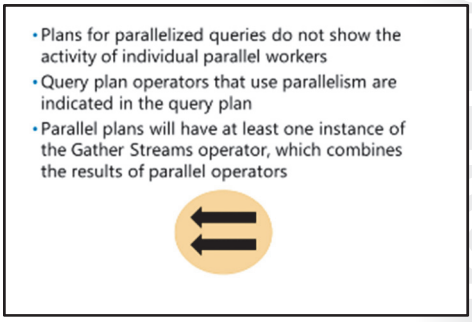
Hash matches are optimal for large, unsorted input data streams, and for aggregate calculations.

Note: There is a class of query performance tuning problems that are caused when the query optimizer selects one join type when another would give better performance. This can be a particular problem where data distribution is uneven, and some values in the first input data stream have very large numbers of rows in the second data stream, where others have very few.

While you can use a join hint to force a particular join operator to be used, the query optimizer will typically select the optimal join type when table statistics are up to date.

Parallel Query Execution Plans

On a multiprocessor system, the SQL Server query optimizer might attempt to speed up queries that require large numbers of rows to be processed by running parts of the task in parallel on more than one CPU at the same time. This process is known as parallelism.



Additional Reading: For more information about how SQL Server operates on multiprocessor systems, see Module 1 of this course: *SQL Server Architecture, Scheduling, and Waits*.

The query execution plan for queries that are executed in parallel are different from other execution plans.

Although you might expect that the activity of individual threads participating in a parallel query plan would be visible in the query plan, this is not the case. Query execution plans show a logical sequence of operators and do not go into sufficient detail to show the activity of individual worker threads.

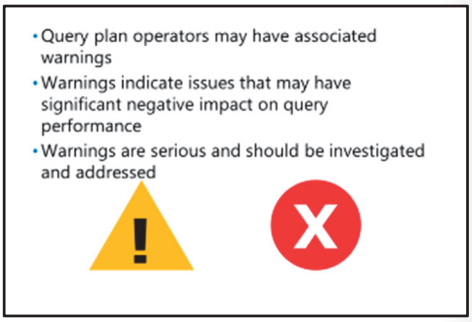
Instead, in a parallel query execution plan, operators that use parallelism are flagged as such:

- In a graphical query plan, parallelized operators have a small orange circle containing two arrows overlaid on the bottom right-hand corner of the operator icon.
- In XML query plans, parallelized operators have the "Parallel" attribute set to "true".
- In text query plans generated by SHOWPLAN_ALL or STATISTICS PROFILE, the result set contains a **Parallel** column with a value of 1 for parallelized operators.

Parallel query plans will also contain at least one instance of the Gather Streams operator, which carries out the work to combine the results of parallelized operators earlier in the query execution plan.

Warnings in Query Execution Plans

If the query optimizer finds a problem that could affect the performance of your query, it might include a warning on one or more of the operators in the query plan.



- In a graphical query plan, operators with associated warnings are indicated by an exclamation or error symbol overlaid on the bottom right-hand of the icon. The detail of the error is available in the operator properties.
- In an XML query plan, operators with associated warnings are indicated by the "Warnings" subelement having one or more attributes with a "true" value.
- In text query plans generated by SHOWPLAN_ALL or STATISTICS PROFILE, the result set contains a **Warnings** column with a value of the warning text.

Some typical warnings include:

- **Type conversion in expression [ColumnExpression] may affect "CardinalityEstimate" in query plan choice**. This warning indicates that a type conversion may be preventing statistics from being used correctly. This will typically occur where values in a WHERE or JOIN clause are not of the same data type.
- **Columns with no statistics: [Column name]**. In databases where AUTO CREATE STATISTICS is OFF, this warning will appear when a column is referenced in a query filter that has no statistics. In databases where AUTO CREATE STATISTICS is ON this warning will never be shown; instead, the missing statistics will be automatically created.
- **No join predicate**. This warning will be shown when your query has no ON clause (or equivalent) when two or more tables are referenced in the query. This is more common in non-ANSI 92 compliant queries where referenced tables appear in a comma-separated list in the FROM clause.

When warnings appear in query plans, you should take steps to understand the cause and address them.

Demonstration: Working with Query Execution Plans

In this demonstration, you will see:

- An example of a parallel query plan.
- An example of a query plan warning.
- How to compare query plans using SSMS.
- How to resolve a query plan warning about missing statistics.
- How to change an execution plan by adding an index.

Demonstration Steps

1. In SQL Server Management Studio, in Solution Explorer, double-click the **Demo 3 - Working with plans.sql** script file.
2. Select the query under the comment that begins **Module 7 – Demo 3,** and click **Execute.**
3. Select the query under the comment that begins **Step 1** and, on the **Query** menu, click **Display Estimated Execution Plan**. Note the warning indicator.
4. Right-click the plan generated in the previous step, and then click **Compare Showplan**.
5. In the **Open** dialog box, click **D:\Demofiles\Mod07\demo2.sqlplan**, and then click **Open** to display two plans together.
6. In the **Demo 3 - Working with plans.sql** query window, select the query under the comment that begins **Step 3**, and on the **Query** menu, click **Display Estimated Execution Plan**. Note that both statements have the same estimated query plan.
7. On the **Query** menu, click **Include Actual Execution Plan**. Select the query under the comment that begins **Step 3**, and click **Execute**.
8. On the **Execution plan** tab, note that the actual execution plans are the same. Hover over the **Clustered Index Seek** of the Orders table (no statistics) icon and note the warning. Also note the discrepancy between the actual and estimated number of rows for the Clustered Index Seek of the Orders table.
9. Select the query under the comment that begins **Step 4**, and click **Execute** to generate a statistics object.
10. Select the query under the comment that begins **Step 3**, and on the **Query** menu, click **Display Estimated Execution Plan**. Notice that the missing statistics warning is no longer shown. If the warning is still there, execute the query again.
11. Select the query under the comment that begins **Step 5**, and click **Execute** and examine the actual execution plan.
12. On the **Execution plan** tab, note the **Estimated Subtree Cost** from the SELECT operator, and the missing index suggestion.
13. Select the query under the comment that begins **Step 6**, and click **Execute** to create a covering index for the query.
14. Select the query under the comment that begins **Step 7**, and click **Execute** and examine the actual execution plan.
15. On the **Execution plan** tab, note that the **Estimated Subtree Cost** from the SELECT operator has reduced.

16. Close SQL Server Management Studio without saving any changes.

Check Your Knowledge

Question	
Which query execution plan join operator requires that both input data streams are sorted?	
Select the correct answer.	
	Merge Join
	Hash Match
	Nested Loops

Lesson 4
Adaptive Query Processing

SQL Server 2017 introduces Adaptive Query Processing. This is a suite of technologies that improve performance by improving the accuracy of query cost predictions. The technologies in Adaptive Query Processing are:

- Batch mode memory grant feedback.
- Batch mode adaptive joins.

Interleaved execution.

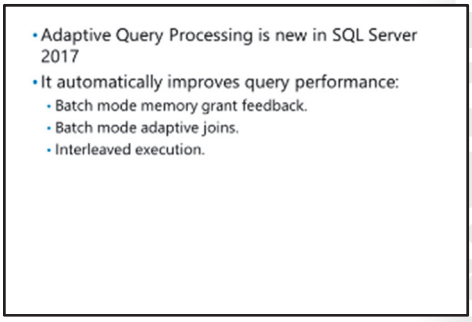
Lesson Objectives

At the end of this lesson, you will be able to:

- Explain the goals of Adaptive Query Processing.
- Describe batch mode memory grant feedback.
- Describe batch mode adaptive joins.
- Describe interleaved execution.

About Adaptive Query Processing

Adaptive Query Processing is new in SQL Server 2017, and is designed to improve the performance of queries that execute less than optimally.



As discussed earlier in this module, the query optimizer converts Transact-SQL statements into a series of logical operations called the query execution plan. Assuming it is not a trivial query, more than one possible plan is created and costed, and the least expensive plan is used. However, for a number of reasons, the query may not run in the most efficient way.

Adaptive Query Processing is designed to automatically improve query performance in three specific areas:

- **Batch mode memory grant feedback**. Memory grant size is sized incorrectly.
- **Batch mode adaptive joins**. Join types benefit from being selected at query execution time.
- **Interleaved execution**. Multi-statement table-valued functions require accurate statistics.

Each of these areas are discussed in the following topics.

Enabling Adaptive Query Processing

Adaptive Query Processing is run automatically for databases with a compatibility level of 140. Compatibility level 140 is only available for SQL Server 2017 databases. If necessary, set the compatibility level using the ALTER DATABASE command.

If you are running SQL Server 2017, use ALTER DATABASE to set the compatibility level to 140:

Set Compatibility Level to 140

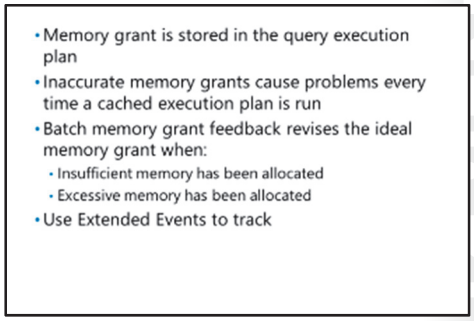
```
ALTER DATABASE <database_name> SET COMPATIBILITY_LEVEL = 140;
```

For more information about Adaptive Query Processing see Microsoft Docs:

Adaptive query processing in SQL databases

https://aka.ms/Gwykz1

Batch Mode Memory Grant Feedback



For a query to execute efficiently, all the rows must fit in memory, with no rows spilling to disk. The query optimizer uses two things to estimate the amount of memory required:

- Estimated row counts.
- Width of the rows in bytes.

This is known as the ideal memory grant and is stored with the query execution plan.

When a query is executed, the allocated memory should match the ideal memory grant. However, if insufficient memory is granted, rows which do not fit into memory spill to a temporary space on disk. This affects query performance because writing data to disk, and accessing it, is much slower than accessing it from memory. Overestimated ideal memory grant potentially starves other queries of memory, and so should also be avoided.

Every time a cached query plan is reused, because the inaccurate memory grant is stored in the query plan, the same memory grant is used. Match mode memory grant feedback solves this problem.

Batch mode memory grant feedback compares the ideal memory grant with the actual query data size. If the actual value is much larger or smaller than the ideal memory grant, the value stored in the cached execution plan is updated. If the query plan is subsequently reused, the more accurate memory grant is used.

Batch mode memory grant feedback is used when:

- There is an insufficient memory grant. Cached query plans that cause a spill to disk will be updated with a revised ideal memory grant.
- There is an excessive memory grant. Cached query plan that allocate more than twice the actual memory are updated with a revised ideal memory grant. For excessive memory grants, the original memory grant must be greater than 1 MB.

If the memory requirements of a cached query plan fluctuate significantly because the query plan is executed for different parameter values, batch mode memory grant feedback is not used.

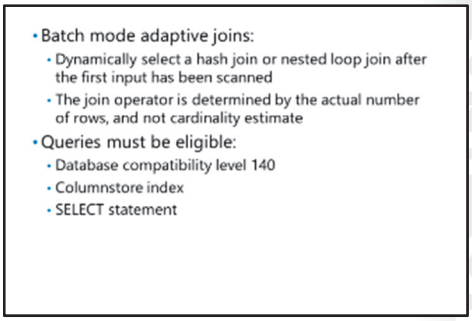
You can monitor the activity of batch mode memory grant feedback using the following Extended Events:

- *spilling_report_to_memory_grant_feedback* records when excessive memory grants are detected.
- *memory_grant_updated_by_feedback* records when a cached query plan ideal memory grant is updated.
- *memory_grant_feedback_loop_disabled* records when the feedback process is disabled due to inconsistent actual memory usage.

Batch Mode Adaptive Joins

Queries can be executed with different logical join operators, including nested loop and hash joins.

- Nested loop joins take one of the data streams and execute it once for each of the rows in the other data stream. This is typically optimal when the outer data stream contains a small number of rows.
- Hash joins create a table of hashes for each data stream, then use the hash tables to match values. This is typically optimal when both streams contain roughly equal number of rows.



For queries that include a table with a Columnstore index, batch mode adaptive joins allows the choice between a hash join or nested loop join to be deferred until after the first input has been processed. This is known as an adaptive join operator, and it uses a row-count threshold to determine which join operator to use. The choice of join operators is made dynamically for each execution of the query plan.

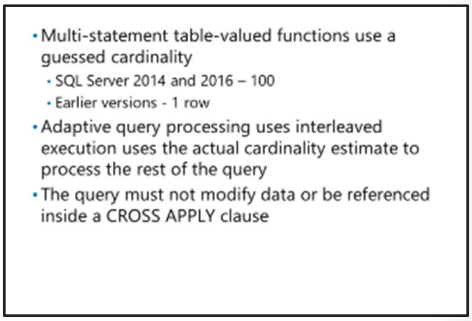
- If the count of rows in the first join input is less than the adaptive join operator's threshold, a nested loop join operator is used.
- If the count of rows in the first join input more than the adaptive join operator's threshold, a hash join operator is used.

Batch mode adaptive joins can improve performance for queries where the number of rows returned from a Columnstore index varies above and below the adaptive join threshold. The choice between join types is based on the actual number of rows affected by the query, irrespective of cardinality estimation. Query plans using the adaptive join operator require more memory to execute than an equivalent nested loop plan.

A query is eligible for an adaptive join if it meets the following conditions:

- Database compatibility level is 140.
- The query is a SELECT statement.
- It is possible to use a nested loop join or a hash join to resolve the query, and the same data stream is used as the outer reference.
- The hash join supports batch mode, because the query references a Columnstore index.

Interleaved Execution



Functions that comprise more than one Transact-SQL statement and return a table are known as multi-statement table-valued functions (MSTVFs).

MSTVFs are executed once for each row of input values, and each execution can return zero or more rows. For the purposes of query execution plan selection, MSTVFs have a guessed cardinality value. In SQL Server 2014 and 2016, this is 100 rows; in earlier versions, it is one row.

With adaptive query processing, interleaved execution allows a statement referencing an MSTVF to be revised during execution. The first time a statement eligible for interleaved execution is executed, the query optimizer identifies the sub-tree of the query plan that references the MSTVF and executes it. It uses the cardinality information from the result as an input to create a query execution plan for the remaining portions of the query. The final query plan is stored in the query plan cache for possible later reuse.

A statement referencing an MSTVF is eligible for interleaved execution when it meets the following criteria:

- Database compatibility level of 140.
- The query does not modify data.
- The MSTVF is not referenced inside a CROSS APPLY clause.

Check Your Knowledge

Question	
Which is a benefit of adaptive query processing?	
Select the correct answer.	
	Better optimized queries for SQL Server 2016 and later
	Faster queries that could use hash joins or merge joins
	Queries that have incorrect memory grants stored in the execution plan
	Queries that do not make sufficient use of indexes

Lab: Query Execution and Query Plan Analysis

Scenario

While investigating a new SQL Server instance, you have come across some workloads that are running slowly. You decide to analyze execution plans for those workloads. In this lab, you will analyze execution plans and identify plan issues. You will then fix them to improve execution performance of workloads.

Objectives

At the end of this lab, you will be able to:

- Improve the performance of a SELECT statement by analyzing the query plan.
- Improve the performance of a stored procedure by analyzing the query plan.

Estimated Time: 60 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Improving SELECT Performance for Historical Marketing Campaign Data

Scenario

The Proseware Inc. team has acquired some historical data about marketing campaigns run by a similar company. This data has been loaded into the Proseware schema in the AdventureWorks database.

Data users are complaining that their queries are running slowly, and have provided an example of a query which runs more slowly than they expect. Your task is to examine the query plan for clues to the source of its poor performance.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Collect an Actual Execution Plan

3. Rebuild Table Statistics

4. Compare the New Actual Execution Plan

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQ**L as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

2. Run **Setup.cmd** in the **D:\Labfiles\Lab07\Starter** folder as Administrator.

▶ Task 2: Collect an Actual Execution Plan

1. Start **SQL Server Management Studio**, then open the project file **D:\Labfiles\Lab07\Starter\Project\Project.ssmssln** and the Transact-SQL file **Lab Exercise 01 - tuning 1.sql**.

2. Collect an actual execution plan for the query shown under the comment that begins **Task 1**. Note the execution time.

3. Save the actual query plan as **D:\Labfiles\Lab07\plan1.sqlplan**.
   Looking at the actual query execution plan, do you see any potential causes of the performance issue? Hint: compare estimated row count to actual row count for several of the operators in the query plan.

▶ Task 3: Rebuild Table Statistics

- Execute the query under the comment that begins **Task 2** to rebuild the statistics held for the **Proseware.Campaign** and **Proseware.CampaignResponse** tables.

▶ Task 4: Compare the New Actual Execution Plan

1. Re-run the query under the comment that begins **Task 1** and collect a new actual execution plan.
2. Compare the new execution plan to the plan you saved in **Task 1** (as **D:\Labfiles\Lab07\plan1.sqlplan**).
3. Is the run time of the query reduced? Does the actual row count now match the estimated row count more closely?
4. Are there more improvements you could make to the performance of this query based on the data returned in the execution plan?

Results: At the end of this exercise, you will have improved the performance of a SELECT query by analyzing the query plan.

Exercise 2: Improving Stored Procedure Performance

Scenario

The Proseware Inc. team has decided to continue to add new data to the **Proseware.CampaignResponse** table. A stored procedure—**Proseware.up_CampaignResponse_Add**—has been created for this purpose. Because the Proseware Inc. team expects to be calling this stored procedure many times a day, you will check the performance of the stored procedure code by examining the query plan. For the purposes of this exercise, you can add rows to the campaign named **1010000**, because this campaign name is being used for testing.

The main tasks for this exercise are as follows:

1. Collect an Actual Execution Plan
2. Add a Covering Index
3. Change the Data Type of the @CampaignName Parameter

▶ Task 1: Collect an Actual Execution Plan

1. In SSMS Solution Explorer, open the Transact-SQL file **Lab Exercise 02 - tuning 2.sql**.
2. Amend the query under the comment that begins **Task 1**, to execute the stored procedure **Proseware.up_CampaignResponse_Add** with the following parameter values:
   - @CampaignName = 1010000
   - @ResponseDate = '2016-03-01'
   - @ConvertedToSale = 1
   - @ConvertedSaleValueUSD = 100.00

3. Execute the query, collecting an actual execution plan.
4. Save the actual query plan as **D:\Labfiles\Lab07\plan2.sqlplan**.

   What two changes might you suggest, to the database or the stored procedure code, to improve the performance of this stored procedure?

▶ Task 2: Add a Covering Index

1. Under the heading for **Task 2**, edit the query to add a unique nonclustered index to the **Proseware.Campaign** table on the **CampaignName** column.
2. Execute the code under the **Task 1** heading again. Save the actual query plan as **D:\Labfiles\Lab07\plan3.sqlplan**.

   What do you notice about the new plan compared to the plan you collected in **Task 1** (saved as **D:\Labfiles\Lab07\plan2.sqlplan**)? Why is an index scan used on ix_Campaign_CampaignName in the first query?

▶ Task 3: Change the Data Type of the @CampaignName Parameter

1. Under the comment that begins **Task 3**, amend the query to change the data type of the @CampaignName parameter of the stored procedure **Proseware.up_CampaignResponse_Add** to **varchar(20)**.
2. Execute the code under the **Task 1** heading again. Save the actual query plan as **D:\Labfiles\Lab07\plan4.sqlplan**.
   What do you notice about the new plan compared to the plan you collected in **Task 1** (saved as **D:\Labfiles\Lab07\plan2.sqlplan**)?

Results: At the end of this lab, you will have examined a query execution plan for a stored procedure and implemented performance improvements by adding a covering index and eliminating an implicit data type conversion.

Question: How might you determine that a performance problem is due to a query execution plan, not a server-level resource problem (such as I/O, CPU or memory)?

Review the Exercise Two Execution Plans

Using the Compare Showplan feature in SSMS, compare the actual execution plans you saved during Exercise 2:

- D:\Labfiles\Lab07\plan2.sqlplan.
- D:\Labfiles\Lab07\plan3.sqlplan.
- D:\Labfiles\Lab07\plan4.sqlplan.

Make sure you understand how the changes made to the database caused the query plan to change.

Module Review and Takeaways

In this module, you learned how to capture, read, and analyze execution plans. You have also learned about the logical and physical query processing and internals of query optimizer.

Best Practice: Use execution plans to find high-cost operations which may be limiting the performance of your queries.

Review Question(s)

Check Your Knowledge

Question	
In which direction is the flow of data in a graphical query execution plan?	
Select the correct answer.	
	Left to right
	Top to bottom
	Bottom to top
	Right to left

Module 8

Plan Caching and Recompilation

Contents:

Module Overview

As a mechanism to improve performance, Microsoft® SQL Server® stores query execution plans for reuse in an area of memory called the plan cache. This module describes caching and recompilation of query execution plans. It focuses on architectural concepts, troubleshooting scenarios, and best practices related to the plan cache.

Objectives

After completing this module, you will be able to:

- Analyze plan cache internals.
- Troubleshoot common plan cache issues.
- Use the Query Store to identify when query plans have changed, and to force the use of a query plan.

Lesson 1
Plan Cache Internals

When a Transact-SQL statement is executed, the query optimizer, a component of the SQL Server Database Engine, is used to convert the Transact-SQL statement into a sequence of logical operators that can be used to carry out the commands in the Transact-SQL statement. This sequence of operators is known as the query execution plan.

Additional Reading: For more details of the process used by the query optimizer to compile a query execution plan for a Transact-SQL statement, see Module 7 of this course, *Query Execution and Query Plan Analysis*.

Compiling a query execution plan can be a CPU-intensive process. The CPU cost of compiling a query execution plan will, in general, rise in line with the complexity of the Transact-SQL statement; the more complex the statement, the more CPU resources are required to compile an execution plan for it.

On the assumption that a Transact-SQL statement might be executed more than once, compiled query execution plans are stored in an area of memory called the plan cache. If the Transact-SQL statement is executed again, the query execution plan in the plan cache can be retrieved and reused, saving the cost of compiling a new query execution plan.

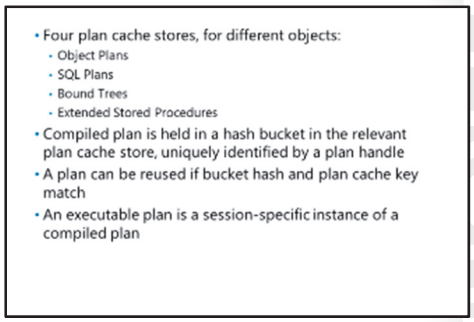
Understanding the plan cache structure and caching techniques will not only help you to write better queries, it will also help you to find and troubleshoot poorly performing queries.

Lesson Objectives

After completing this lesson, you will be able to:

- Explain how query execution plans are added to and retrieved from the query plan cache.
- Describe how the plan cache is managed.
- Identify the types of query that will not have an entry in the plan cache.
- Analyze the contents of the plan cache.

Query Plan Caching and Retrieval



The plan cache is made up of four memory areas, known as plan cache stores. Each plan cache store contains different types of cached query execution plans:

- **Object Plans**. This stores plans for stored procedures, functions, and triggers.
- **SQL Plans**. This stores ad hoc plans, auto-parametrized plans, and prepared plans.
- **Bound Trees**. These are the structures produced by the algebrizer for views, defaults, and constraints.
- **Extended Stored Procedures**. Extended stored procedures (stored procedures that call a method in a dynamic link library file) have their own plan cache store.

You can view high level information about the plan cache stores using the **sys.dm_os_memory_cache_counters** system dynamic management view (DMV):

sys.dm_os_memory_cache_counters

```
SELECT *
FROM sys.dm_os_memory_cache_counters
WHERE name in ('Object Plans','SQL Plans','Bound Trees','Extended Stored Procedures');
```

Plan Cache Store Organization

Each plan cache store is a hash table, made up of a series of *buckets*. Each bucket contains zero or more cached query execution plans. A hash algorithm is used to assign each query execution plan to a cache store bucket:

- For the **Object Plans**, **Bound Trees**, and **Extended Stored Procedures** cache stores, the hash value is calculated, based on the **database_id** and **object_id** of the database object with which the query execution plan is associated.
- For Transact-SQL statements in the **SQL Plans** cache store, the hash value is based on the **database_id** where the statement is executed, in addition to the statement text.

You can use the following query to view information about the number of hash buckets in each plan cache store:

Plan Cache Store Buckets

```
SELECT cc.name, buckets_count
FROM sys.dm_os_memory_cache_hash_tables AS ht
JOIN sys.dm_os_memory_cache_counters AS cc
ON ht.cache_address = cc.cache_address
WHERE cc.name IN ('Object Plans','SQL Plans','Bound Trees','Extended Stored Procedures');
```

Individual query plans in a plan cache store bucket are identified by a unique binary value called the *plan handle*.

Finding a Query Execution Plan in a Plan Cache Store

When a Transact-SQL statement is executed, the query optimizer will search the plan cache for a query execution plan that might have been cached during an earlier execution of the statement. This is done by:

1. Identifying which of the four plan cache stores would contain the cached query execution plan. This is based on the type of Transact-SQL statement.
2. Calculating the hash value for the plan cache store bucket to which the Transact-SQL statement would belong. This step uses the same hash algorithm that is used when a query execution plan is added to the plan cache store.
3. Searching all the query execution plans in the plan cache store bucket for a query execution plan with a cache key that matches the Transact-SQL statement. The plan cache key is a compound key made up of a number of plan cache attributes. Only if the plan cache key matches will the plan be reused.

The plan cache key includes several attributes that are properties of the client session from which the Transact-SQL statement is executed.

The following query gives a complete list of the attributes that make up the plan cache key:

Plan Cache Key

```
SELECT pa.*
FROM (SELECT TOP(1) plan_handle FROM sys.dm_exec_cached_plans) AS cp
CROSS APPLY sys.dm_exec_plan_attributes(cp.plan_handle) AS pa
WHERE is_cache_key = 1;
```

For more detail on the output of this query, see the topic *sys.dm_exec_plan_attributes (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_plan_attributes (Transact-SQL)

http://aka.ms/ptp3vz

Executable Plans

The query execution plans stored in the plan cache are *compiled plans*. These are generalized, reusable query plans. A single compiled plan might be used simultaneously by several different client sessions. An instance of a compiled plan in use by a session is referred to as an *executable plan*, or an *execution context*. An executable plan contains metadata specific to the instance of the compiled plan, such as user id, parameter values, and local variable values.

For example, if 10 client sessions execute the same Transact-SQL statement simultaneously, there will be 10 executable plans; each executable plan might be an instance of the same compiled plan. Each client session could be using a different user id and different parameter values.

Executable plans can be cached and reused, but they are much less costly to create than compiled plans.

Plan Cache Management

The contents of the plan cache cannot grow indefinitely, because the amount of memory available to SQL Server is finite. If the size of the plan cache is not managed, it might grow to occupy all the memory available to the SQL Server process. Several methods are used to manage the size of the plan cache.

- Plan cache housekeeping will begin when the cache stores hit threshold values
- The least-expensive plans are removed first from the plan cache
- Manually clear the plan cache with DBCC commands
- Clear the cached plans for a single object with **sp_recompile**
- Other operations clear the plan cache

Plan Cache Eviction Policy

SQL Server uses a measure calculated as a percentage of the total amount of memory available to the instance, called the *cache store pressure limit*, before taking action to reduce the size of the plan cache.

In all versions of SQL Server since SQL Server 2005 SP2, the cache store pressure limit is calculated as:

- 75 percent of visible target memory from 0 to 4 GB, **plus**.
- 10 percent of visible target memory from 4 GB to 64 GB, **plus**.
- 5 percent of visible target memory > 64 GB.

In this context, *visible target memory* means the usable portion of the maximum amount of memory that the SQL Server instance is configured to consume.

The following query can be used to find out the visible target memory of a SQL Server instance:

Visible Target Memory

```
SELECT visible_target_kb FROM sys.dm_os_sys_info;
```

Memory pressure can come from three sources:

- **Local memory pressure**:
  - The size of a single plan cache store exceeds 62.5 percent of the cache store pressure limit.
  - The number of cached query plans in a plan cache store is four times the number of hash buckets.
- **Internal global memory pressure**:
  - Virtual address space is low.
  - The total size of all plan cache stores exceeds 80 percent of the cache store pressure limit.
- **External global memory pressure**:
  - The operating system reduces the amount of memory available to the SQL Server instance.

In any of these circumstances, the size of the plan cache will be reduced by evicting query execution plans from the plan cache.

Query Execution Plan Cost

When query execution plans are evicted from the plan cache, the plans that would be least expensive to recreate are evicted first. Query plan cost is calculated as:

- For ad hoc plans, the initial cost is 0. The cost is incremented by 1 every time the ad hoc plan is reused.
- For other plans (all plans that are not ad hoc), the cost is calculated in terms of ticks. The cost consists of I/O cost, CPU cost and memory cost. The cost is evaluated as:
  - Total cost = I/O cost + CPU cost + Memory cost.
    - I/O cost: 2 I/O operations = 1 tick (with a maximum of 19 ticks).
    - CPU cost: 2 context switches = 1 tick (with a maximum of 8 ticks).
    - Memory cost: 128 KB (16 pages) = 1 tick (with a maximum of 4 ticks).

The following query uses the **sys.dm_os_memory_cache_entries** DMV to show cost information about query plans in the **SQL Plans** and **Object Plans** plan cache stores:

The following query uses the **sys.dm_os_memory_cache_entries** DMV to show cost information about query plans in the **SQL Plans** and **Object Plans** plan cache stores:

sys.dm_os_memory_cache_entries

```
SELECT e.[type] AS cache_type, st.[text], p.objtype, p.usecounts,
p.size_in_bytes,e.disk_ios_count, e.context_switches_count,
e.pages_kb AS memory_kB, e.original_cost, e.current_cost
FROM sys.dm_exec_cached_plans AS p
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
JOIN sys.dm_os_memory_cache_entries AS e
ON p.memory_object_address = e.memory_object_address
WHERE p.cacheobjtype = 'Compiled Plan'
AND e.type IN ('CACHESTORE_SQLCP','CACHESTORE_OBJCP')
ORDER BY e.[type], p.objtype, e.current_cost DESC
```

Manually Removing Plan Cache Entries

Query execution plans can be removed from the plan cache using DBCC commands:

- **DBCC FREEPROCCACHE**. This command is used to:
  - Clear all plans from the cache.
  - Clear a particular plan by specifying a **plan_handle**.
  - Clear all plans for a particular Transact-SQL statement by specifying a **sql_handle**.
  - Clear all cached plans for a specific Resource Governor resource pool.
- **DBCC FREESYSTEMCACHE**. Removes all unused entries from the plan cache or from a specific Resource Governor resource pool.

 **Note:** It is not recommended that either of these DBCC commands be issued on a production SQL Server instance. The cost of recompiling a large number of query execution plans could have a significant impact on performance.

For more information on DBCC FREEPROCCACHE, see the topic *DBCC FREEPROCCACHE (Transact-SQL)* in Microsoft Docs:

 **DBCC FREEPROCCACHE (Transact-SQL)**

http://aka.ms/h36m9a

For more information on DBCC FREESYSTEMCACHE, see the topic *DBCC FREESYSTEMCACHE (Transact-SQL)* in Microsoft Docs:

 **DBCC FREESYSTEMCACHE (Transact-SQL)**

http://aka.ms/bcscjb

Query execution plans for a database object (a stored procedure, trigger, table, view, or user-defined function) can be removed from the plan cache using the **sp_recompile** system stored procedure.

- If the object name passed to **sp_recompile** is a stored procedure, trigger, or user-defined function, the query execution plan(s) for the object are removed from the plan cache.
- If the object name passed to **sp_recompile** is a table or a view, query execution plans for all stored procedures, triggers, and user-defined functions that reference the table or view, are removed from the plan cache.

For more information on **sp_recompile**, see the topic *sp_recompile (Transact-SQL)* in Microsoft Docs:

 **sp_recompile (Transact-SQL)**

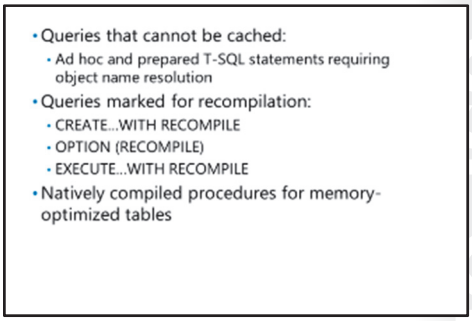
http://aka.ms/rdaclt

Other operations will clear the plan cache:

- Stopping the SQL Server instance.
- Restoring a database.
- ALTER DATABASE MODIFY FILEGROUP command.
- ALTER DATABASE COLLATE command.
- Detaching a database.
- RECONFIGURE command.
- Changing database compatibility level.
- Using the database auto-close option.

Queries Without Plan Cache Entries

Not all Transact-SQL statements executed on a SQL Server instance will have an entry in the plan cache. This can be because a Transact-SQL statement is unsuitable for caching, as it is explicitly marked for recompilation, or because the statement is not executed by the query execution engine.



Transact-SQL Statements Unsuitable for Caching

Query execution plans for some Transact-SQL statements are not cached when they require object name resolution to be performed at execution time. Object name resolution is required when a database object is referred to by a one-part name—that is, the object schema name is not specified. The binding phase of query compilation must use the default schema of the security context under which the Transact-SQL statement is being executed to determine which object the statement references. This is important because, in SQL Server, database objects of the same name can exist in different schemas.

The following query could refer to the Person table in many different schemas; for example, the **Person.Person** table or the **HumanResource.Person** table, depending on the default schema of the user executing the query:

Object Name Resolution Example

```
SELECT * FROM Person;
```

Query execution plans for ad hoc queries and prepared statements that require object name resolution are not cached. This limitation does not apply to Transact-SQL statements in stored procedures, functions, or triggers.

Note: You should get into the habit of using two-part names (Schema.Object) in your Transact-SQL statements. This increases the chances of cached query execution plan reuse.

Transact-SQL Statements Explicitly Marked for Recompilation

Database objects and Transact-SQL statements can be explicitly marked for recompilation in their code.

The query plan for stored procedures created with the CREATE PROCEDURE...WITH RECOMPILE option is not cached. The procedure is compiled every time it is executed.

For more information on CREATE PROCEDURE...WITH RECOMPILE, see the topic CREATE PROCEDURE (Transact-SQL) in Microsoft Docs:

CREATE PROCEDURE (Transact-SQL)

http://aka.ms/psgsyy

Individual Transact-SQL statements, whether they are ad hoc statements or part of the definition of a database object, can be configured to be recompiled on every execution using the OPTION (RECOMPILE) query hint.

For more information on the OPTION (RECOMPILE) query hint, see the topic *Query Hints (Transact-SQL)* in Microsoft Docs:

Query Hints (Transact-SQL)

http://aka.ms/t97w4b

It is also possible to mark a stored procedure for recompilation at run time using the EXECUTE...WITH RECOMPILE option. Note that this only affects the outermost procedure when procedure calls are nested.

For more information on using EXECUTE...WITH RECOMPILE, see the topic *EXECUTE (Transact-SQL)* in Microsoft Docs:

EXECUTE (Transact-SQL)

http://aka.ms/t7pp5z

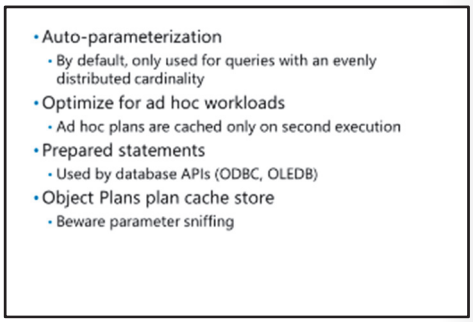
Transact-SQL Statements Not Executed by the Query Execution Engine

Natively compiled stored procedures that interact with memory-optimized tables are compiled to machine code when they are created, and are executed by the In-Memory OLTP Engine. These procedures do not have query execution plans stored in the plan cache.

Maximizing Plan Cache Efficiency

SQL Server employs several techniques to make the best use of the plan cache:

- Auto-parameterization.
- Ad hoc query caching.
- Prepared queries.
- The Object Plans plan cache store.



Auto-parameterization

With this feature, the query optimizer can automatically replace a literal value in a Transact-SQL statement with a parameter in the associated query execution plan. If the query execution plan is added to the plan cache, other statements, which run the same query with different literal values, can reuse the cached plan.

By default, auto-parameterization is carried out conservatively; it will only occur for queries where the cardinality statistics available to the query optimizer indicate that a parameterized query execution plan will be equally effective for all values of the parameter. An example of a good candidate for auto-parameterization would be a query selecting a single row from a table by a column with a unique index; because all the values in the column must be unique, any value passed to the query will have the same cardinality.

It is possible to make auto-parameterization more aggressive by using the FORCED PARAMETERIZATION option, which can be configured at database level or on the level of individual Transact-SQL statements. Using this setting is not normally recommended, because parameterizing statements that do not operate on equally-distributed data is likely to cause performance problems.

Ad Hoc Query Caching

For SQL Server instances on which large numbers of ad hoc queries are executed, the **SQL Plans** plan cache store may fill up with single-use query plans. To reduce the amount of cache space occupied by such single-use query execution plans, you can turn on the **optimize for ad-hoc workloads** option at database level. Turning on this option has the effect that a query execution plan for an ad hoc query will not be added to the plan cache until it is used for the second time.

Prepared SQL Statements

With the database engine, client applications can prepare Transact-SQL statements before they are executed, using the database API exposed through the ODBC and OLEDB interfaces. When a client application prepares a Transact-SQL statement, the statement's query execution plan is compiled once. Each subsequent execution of the prepared statement will reuse the precompiled query execution plan.

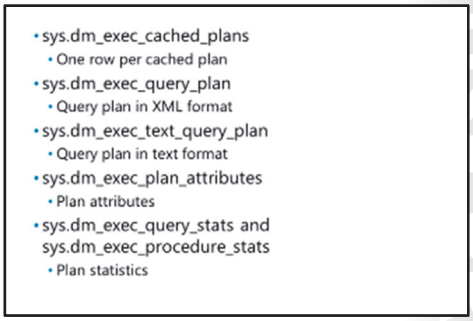
The Object Plans Plan Cache Store

Stored procedures, triggers, and user-defined functions have cached query execution plans stored in the **Object Plans** plan cache store. Each Transact-SQL statement within a stored procedure, trigger, or user-defined function will have its own query execution plan in the **Object Plans** plan cache store; all the plans related to an object will be held in the same plan cache store hash bucket. The query execution plan for each Transact-SQL statement in a stored procedure, trigger, or user-defined function is compiled the first time the statement is executed.

When Transact-SQL statements in a stored procedure trigger, or user-defined functions that reference parameter values are executed for the first time, the query execution plan is compiled, based on the values of the parameters supplied at the first execution. Subsequent executions of the object code will use the cached query execution plan, whether or not the same parameter values are used.

Depending on the cardinality of the supplied parameter values, the compiled query execution plan might or might not be optimal for use with other values. When this behavior results in a query execution plan that is suboptimal for some parameter values, it is referred to as *parameter sniffing*.

Examining the Plan Cache



You can use a variety of system DMVs and dynamic management functions (DMFs) to inspect the contents of the plan cache. These objects can be used together to query many different views of the contents of the plan cache.

sys.dm_exec_cached_plans

This DMV returns a row for each query execution plan in the plan cache. For more information on this DMV, see the topic *sys.dm_exec_cached_plans (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_cached_plans (Transact-SQL)

http://aka.ms/b54fha

sys.dm_exec_query_plan

This DMF returns the query execution plan linked to a **plan_handle** in XML format. For more information on this DMF, see the topic *sys.dm_exec_query_plan (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_query_plan (Transact-SQL)

http://aka.ms/pcafan

sys.dm_exec_text_query_plan

This DMF returns the query execution plan linked to a **plan_handle** in text format.

This can be useful because the DMF accepts optional arguments for statement start offset and statement end offset; this means that you can extract a query plan for a single Transact-SQL statement in a multistatement batch or procedure.

For more information on this DMF, see the topic *sys.dm_exec_text_query_plan (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_text_query_plan (Transact-SQL)

http://aka.ms/ir7znb

sys.dm_exec_plan_attributes

This DMF returns the attributes associated with a **plan_handle**. This is useful to check the values of the plan cache key when you are trying to determine which of several cached plans for the same Transact-SQL statement was used by a client application.

Note that the value of the **set_options** column is a bitmask storing the values of 19 different SET options. Some SET options make up part of the plan cache key. For more information on this DMF, see the topic *sys.dm_exec_plan_attributes (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_plan_attributes (Transact-SQL)

http://aka.ms/vuh5gd

sys.dm_exec_query_stats

This DMV returns performance information linked to query execution plans in the plan cache. This can be useful when you are trying to investigate the historical behavior of a cached plan, or looking for the most commonly executed or most expensive query execution plans in the cache.

For more information on this DMV, see the topic *sys.dm_exec_query_stats (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_query_stats (Transact-SQL)

http://aka.ms/n3fhua

sys.dm_exec_procedure_stats

This DMV is similar to **sys.dm_exec_query_stats**, but returns data only for stored procedures.

For more information on this DMV, see the topic *sys.dm_exec_procedure_stats (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_procedure_stats (Transact-SQL)

http://aka.ms/bpirkm

Demonstration: Analyzing the Query Plan Cache

In this demonstration, you will see:

- Methods for collecting information about the plan cache.
- A method for retrieving a query execution plan for an ad hoc Transact-SQL statement from the plan cache.
- The effects of DBCC FREEPROCCACHE and **sp_recompile**.
- An example of auto-parameterization.

Demonstration Steps

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Demofiles\Mod08** folder as Administrator. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.
3. Start **SQL Server Management Studio** (SSMS) and connect to the **MIA-SQL** database engine instance using Windows authentication.
4. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod08** folder.
5. In Solution Explorer, expand the queries folder and open the **Demo 1 - plan cache.sql** script file.
6. Execute the query under the comment that begins **Step 1** to view high level information about the plan cache stores.
7. Execute the query under the comment that begins **Step 2** to show the hash bucket count in each plan cache store.
8. Execute the query under the comment that begins **Step 3** to show the attributes that make up a plan cache key.
9. Execute the query under the comment that begins **Step 4** to show cost information about query plans in the **SQL Plans** and **Object Plans** plan cache stores.
10. Execute the query under the comment that begins **Step 5** to illustrate clearing the plan cache using DBCC FREEPROCCACHE. Emphasize that this should only be done on nonproduction systems.
11. Execute the query under the comment that begins **Step 6** to execute a stored procedure to add a plan to the cache.
12. Execute the query under the comment that begins **Step 7** to examine the cached plan for **uspGetOrderTrackingBySalesOrderID**. Click on the XML value retuned in the **query_plan** column to show a graphical query plan.
13. In the **Demo 1 - plan cache.sql** pane, execute the query under the comment that begins **Step 8** to change set options for this session.
14. Execute the query under the comment that begins **Step 9** to execute the stored procedure again.
15. Execute the query under the comment that begins **Step 7** to examine the cached plan for **uspGetOrderTrackingBySalesOrderID** again.
16. Execute the query under the comment that begins **Step 11** to recompile the stored procedure.
17. Execute the query under the comment that begins **Step 7** to examine the cached plan for **uspGetOrderTrackingBySalesOrderID** again—there will be no plans in the cache.

18. Execute the query under the comment that begins **Step 13** to execute a Transact-SQL statement that will be auto-parameterized.
19. Execute the query under the comment that begins **Step 14** to examine the cached plan for the statement you just ran. Click on the XML value returned in the **query_plan** column to show a graphical query plan.

    Notice that the graphical plan looks unusual—it has only a SELECT operator.
20. Right-click the graphical query plan and click **Show Execution Plan XML**. Notice from the value of the **ParameterizedText** attribute that the query plan has been parameterized. Notice from the value of the **ParameterizedPlanHandle** attribute that this plan references another plan handle. Copy the value of the **ParameterizedPlanHandle** attribute.
21. In the **Demo 1 - plan cache.sql** pane, edit the query under the comment that begins **Step 15** to replace **<plan handle here>** with the value of the plan handle that you copied in the previous step. Execute the query, and click on the XML value returned in the **query_plan** column to show a graphical query plan. Notice that this is the full plan.
22. Keep SSMS open for the next demonstration.

Check Your Knowledge

Question	
In which plan cache store are query execution plans for ad hoc Transact-SQL statements cached?	
Select the correct answer.	
	Object Plans
	Bound Trees
	Extended Stored Procedures
	SQL Plans

Lesson 2
Troubleshooting with the Plan Cache

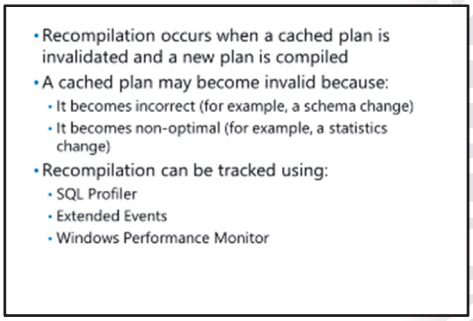
The plan cache can be a useful source of information when you are troubleshooting query performance problems. This lesson covers issues you should be aware of when using the plan cache as a troubleshooting tool, in addition to performance problems that may arise in the plan cache itself.

Lesson Objectives

At the end of this lesson, you will be able to:

- Describe query execution plan recompilation, and reasons that recompilation might occur.
- Explain the limitations of using the plan cache as a performance troubleshooting tool.
- Address performance problems of the plan cache.
- Use the plan cache to guide query optimization.

Query Execution Plan Recompilation



The process of generating a query execution plan is called *compilation*. As you have learned in the previous lesson, in most circumstances, a compiled query plan will be added to the plan cache for later reuse.

Query execution plan *recompilation* occurs when a cached plan is discarded, and a new query execution plan is compiled for a Transact-SQL statement for which a query execution plan existed in the plan cache. The existing query execution plan is replaced in the plan cache by the newly-generated query execution plan.

Additional Reading: For more details of the process used by the query optimizer to compile a query execution plan for a Transact-SQL statement, see Module 7 of this course, *Query Execution and Query Plan Analysis*.

Note: The plan cache contains only the most recently-compiled query execution plan for a Transact-SQL statement. When you are troubleshooting query performance issues, remember that the query execution plan for the Transact-SQL statement you are investigating, which appears in the plan cache, may not be the same query execution plan that was in the plan cache when the issue occurred. The **creation_time** column in the **sys.dm_exec_query_stats** DMV tells you the date and time that a cached plan was compiled.

In the previous lesson, you learned about several methods you can use to remove query execution plans from the plan cache manually, or in response to memory pressure. In addition to those methods, the query optimizer can discard a cached plan when it detects that the cached plan might no longer be useful. A cached plan can cease to be useful when:

- It is no longer correct (for example, if the cached plan refers to objects that have had schema changes).
- It is no longer optimal (for example, when the statistics used at the time the cached plan was last compiled have changed).

The full list of reasons for the recompilation of a cached plan is:

- Schema changed.
- Statistics changed.
- Deferred compile. A referenced object did not exist during compilation but was created at run time; for example, a temporary table created as part of a stored procedure.
- SET option changed.
- Temporary table changed.
- Remote rowset changed.
- FOR BROWSE permission changed.
- Query notification environment changed.
- Partitioned view changed.
- Cursor options changed.
- OPTION (RECOMPILE) requested.

 **Note:** Recompilation takes place at the Transact-SQL statement level; for stored procedures, and other database objects that can contain multiple Transact-SQL statements, the cached plan for each Transact-SQL statement can be recompiled independently of the other Transact-SQL statements in the object definition. This can result in each Transact-SQL statement in a stored procedure having a cached plan with a different compilation date.

There are a number of mechanisms you can use to get information about recompilations, including SQL Profiler, Extended Events, and Windows Performance Monitor.

SQL Profiler

The **SQL:StmtRecompile** event class tracks recompiles at statement level; an event is logged each time a statement is recompiled. The **EventSubClass** of the event class identifies the reason for recompilation.

For more information on the **SQL:StmtRecompile** profiler event class, see the topic *SQL:StmtRecompile Event Class* in Microsoft Docs:

 **SQL:StmtRecompile Event Class**

http://aka.ms/pjilap

Extended Events

The **sql_statement_recompile** event tracks recompiles at statement level; an event is logged each time a statement is recompiled. The reason for recompilation is provided in the **recompile_cause** column.

For more information on the **sql_statement_recompile** Extended Event object, use the Extended Event DMVs:

sql_statement_recompile Extended Event

```
SELECT * FROM sys.dm_xe_object_columns WHERE object_name='sql_statement_recompile';
```

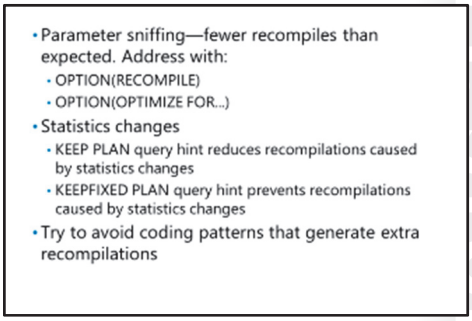
Windows Performance Monitor

High level information about statement-level recompiles can be collected using the following performance counters:

- SQL Server: SQL Statistics
  - Batch requests/sec
  - SQL compilations/sec
  - SQL recompilations/sec

Recompilation Issues

You might encounter performance issues caused by cached plan recompilation happening more often than you expect, or not as often as you expect.



Parameter Sniffing

As discussed in the previous lesson, parameter sniffing can occur for stored procedures, user-defined functions, and triggers when a cached plan is based on unrepresentative parameter values. Later executions that reuse the cached plan have poor performance because the data has significantly different cardinality for different parameter values. The solution to parameter sniffing is either to force more recompilations, or to force the generation of an alternative query execution plan using:

- OPTION (RECOMPILE). As discussed in the previous lesson, adding the OPTION (RECOMPILE) query hint to a Transact-SQL statement prevents the creation of a cached plan for the statement; other statements in the object are not recompiled for every execution. In a multistatement stored procedure, OPTION (RECOMPILE) is preferable to CREATE...WITH RECOMPILE because it limits recompilation to the statement affected by parameter sniffing.
- OPTION (OPTIMIZE FOR...). By adding the OPTION (OPTIMIZE FOR...) query hint to a Transact-SQL statement, you can specify that the query execution plan should be compiled based on other values than the parameter values used on first execution. OPTION (OPTIMIZE FOR...) has three forms:
  - OPTION (OPTIMIZE FOR *@parameter = literal value*). Specifies that a literal value should be used for a specified parameter. Multiple parameters and literal values may be specified as a comma-delimited list. You might use this hint when your data is heavily skewed to a particular value.
  - OPTION (OPTIMIZE FOR *@parameter* UNKNOWN). Specifies that the query plan should base the value of a parameter on statistics rather than a run-time value. Multiple parameters and literal values can be specified as a comma-delimited list. The delimited list may contain both *@parameter = literal value* and *@parameter* UNKNOWN styles.
  - OPTION (OPTIMIZE FOR UNKNOWN). Specifies that the query plan should base the value of all parameters on statistics rather than run-time values.

Using either form of the hint with UNKNOWN might cause the cached plan to be non-optimal for all possible values.

For more information on query hints, see the topic *Query Hints (Transact-SQL)* in Microsoft Docs:

Query Hints (Transact-SQL)

http://aka.ms/t97w4b

Statistics Changes

If statistics on your tables are regularly changing, either because the AUTO_UPDATE_STATISTICS database option is on and being triggered, or because you are updating statistics manually, cached plans that reference the objects will be recompiled to keep the plan optimal for the available statistics. In busy systems, this may result in large numbers of recompilations. Using the KEEP PLAN query hint relaxes the frequency with which recompilation for a statement occurs in response to statistics changes. The KEEPFIXED PLAN hint will prevent a cached plan ever being recompiled in response to statistics changes. You should use these options only as a last resort, because in most circumstances a better query plan will be produced using the current statistics.

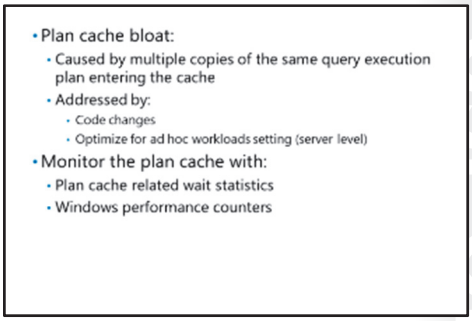
Coding Patterns That Increase Recompilation

The following list is by no means exhaustive, but gives some examples of coding patterns that will increase instances of plan recompilation:

Issue	Troubleshooting Tip
Changing SET options in a batch or stored procedure.	SET options should be configured when a connection is opened.
DDL statements in a batch or stored procedure.	Avoid statements that amend the database schema in the body of stored procedures or batches of Transact-SQL statements.
Temporary tables shared between subprocedures.	When a nested stored procedure references a temporary table created by the outer procedure, the session ID becomes part of the plan cache key. Each new session ID that executes the outer procedure will have its own version of the query plan. Consider using a permanent table or a table-valued parameter to share data between subprocedures.
Dynamic SQL statements executed with EXEC().	Use **sp_executesql** to create parameterized dynamic SQL statements.
Client applications using different SET options.	As far as possible, configure client applications to use the same SET options, so cached plans can be shared.

Problems of the Plan Cache

Plan Cache Bloat



A cached plan cannot be reused unless the plan cache key matches exactly between executions. This can particularly affect SQL Server instances where many ad hoc queries are executed, because changes in query formatting and different SET options between clients can result in many cached plans for queries that are, in most respects, identical. This can cause the size of the plan cache to increase considerably—a phenomenon known as *plan cache bloating*.

Your SQL Server instance may be suffering plan cache bloat if it has a high number of single-use queries in the plan cache.

To explore the plan cache:

Find the Count and Size of Single-Use Plans

```
SELECT COUNT(*) AS single_use_plans, SUM(size_in_bytes) / 1024.0 / 1024.0 AS size_in_mb
FROM sys.dm_exec_cached_plans
CROSS APPLY sys.dm_exec_sql_text(plan_handle)
WHERE objtype IN ('Adhoc', 'Prepared')
AND usecounts = 1
```

Another way to detect plan cache bloat is to aggregate the data in the **sys.dm_exec_query_stats** DMV. Cached plans for queries that differ only in literal values will all have the same **query_hash** value.

An aggregate query over **sys.dm_exec_query_stats** can be used to find cached plans for similar statements. Note that, in this example, any query with more than one similar plan is returned. If you use this code on a production system, this may not return helpful results; and you should consider raising the value.

To explore the plan cache further:

Similar Queries: sys.dm_exec_query_stats

```
WITH planCTE
AS
(
      SELECT query_hash, MAX(plan_handle) AS plan_handle, COUNT(*) AS cnt
      FROM sys.dm_exec_query_stats
      GROUP BY query_hash
      HAVING COUNT(*) > 1 --change this value to suit your system
)
SELECT p.* , st.[text]
FROM planCTE AS p
CROSS APPLY sys.dm_exec_sql_text(p.plan_handle) AS st;
```

A similar query using **query_plan_hash** in place of **query_hash** will return cached plans that have the same execution plan.

Plan cache bloat can be addressed in the SQL Server instance configuration by turning on the **optimize for ad-hoc workloads** option. When this option is enabled, on the first execution of an ad hoc query, a small stub for the query execution plan is added to the plan cache. The full query execution plan will not be cached until the ad hoc query is executed for the second time.

Wait Statistics Related to the Plan Cache

There are several wait statistics types that can be related to the plan cache. These include CMEMTHREAD, SOS_RESERVEDMEMBLOCKLIST, and RESOURCE_SEMAPHORE_QUERY_COMPILE.

CMEMTHREAD. This wait type occurs when a worker is waiting for a thread-safe memory object. Although this is not exclusively related to the plan cache, it can be a result of large numbers of ad hoc query execution plans being added to the plan cache at the same time.

The techniques previously described, to identify plan cache bloat, will confirm if high numbers of CMEMTHREAD waits are caused by the plan cache.

SOS_RESERVEDMEMBLOCKLIST. This wait type occurs when a worker is waiting for the allocation of a block of memory, known as a *multipage unit*. The allocation of multipage units can indicate that Transact-SQL statements with large numbers of values in the IN clause are being added to the plan cache.

RESOURCE_SEMAPHORE_QUERY_COMPILE. This wait occurs when a worker is waiting for memory to compile a large query plan. It can appear when many large query plans are being compiled at the same time.

Information about the size of cached plans can be collected from **sys.dm_exec_query_stats**:

Memory Grants: sys.dm_exec_query_stats

```
SELECT last_grant_kb, st.[text]
FROM sys.dm_exec_query_stats AS p
CROSS APPLY sys.dm_exec_sql_text(p.plan_handle) AS st
ORDER BY last_grant_kb DESC;
```

To address the RESOURCE_SEMAPHORE_QUERY_COMPILE wait, you must determine which query plans are consuming large amounts of memory, and either rewrite the associated queries or attempt to reschedule query compilation.

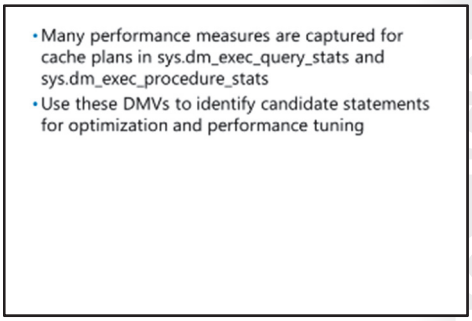
Windows Performance Monitor

Two performance monitor counters can be used to monitor the plan cache:

- **SQL Server Plan Cache: Cache Pages (_Total)**: This counts the total number of pages in the plan cache. A sudden increase in the value of this counter may be an indication of plan cache bloating, whereas a sudden decrease may indicate internal or external memory pressure.
- **SQL Server Memory Manager: SQL Cache Memory**: This shows the total amount of dynamic memory consumed by the SQL plan cache. This is similar to the SQL Server Plan Cache: Cache Pages counter, except this gives the size in kilobytes instead of the number of pages.

Using the Plan Cache to Guide Optimization

In addition to query execution plans, SQL Server also stores performance statistics such as execution time, I/O activity, and CPU utilization for cached plans. These statistics are accessible using the **sys.dm_exec_query_stats** and **sys.dm_exec_procedure_stats** system DMVs. Use the data in these DMVs to guide you when you are troubleshooting performance problems or looking for targets for performance optimization.



Note: sys.dm_exec_query_stats returns performance information for all the cached plans. **sys.dm_exec_procedure_stats** returns a subset of the information returned by **sys.dm_exec_query_stats** returning performance information for stored procedures only.

You can use the following query to find the 10 cached plans that have the highest average run time per execution:

Top 10 Most Expensive Cached Plans by Average Execution Time

```
SELECT TOP(10) OBJECT_NAME(st.objectid, st.dbid) AS obj_name,
               qs.creation_time,
               qs.last_execution_time,
               SUBSTRING      (st.[text],
                                      (qs.statement_start_offset/2)+1,
                                      ((      CASE statement_end_offset
                                              WHEN -1 THEN DATALENGTH(st.[text])
                                              ELSE qs.statement_end_offset
                                              END - qs.statement_start_offset)/2)+1
                                      ) AS sub_statement_text,
               [text],
               query_plan,
               total_worker_time,
               qs.execution_count,
               qs.total_elapsed_time / qs.execution_count AS avg_duration
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) AS qp
ORDER BY avg_duration DESC;
```

With a small change to the previous example, you can use it to find the 10 cached plans that require the highest average CPU per execution.

Top 10 Most Expensive Cached Plans by Average CPU Consumption

```
SELECT TOP(10) OBJECT_NAME(st.objectid, st.dbid) AS obj_name,
              qs.creation_time,
              qs.last_execution_time,
              SUBSTRING      (st.[text],
                                    (qs.statement_start_offset/2)+1,
                                    ((      CASE statement_end_offset
                                            WHEN -1 THEN DATALENGTH(st.[text])
                                            ELSE qs.statement_end_offset
                                            END - qs.statement_start_offset)/2)+1
                                    ) AS sub_statement_text,
              [text],
              query_plan,
              total_worker_time,
              qs.execution_count,
              qs.total_worker_time / qs.execution_count AS avg_cpu_time,
              qs.total_elapsed_time / qs.execution_count AS avg_elapsed_time
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) AS qp
ORDER BY avg_cpu_time DESC;
```

 **Note:** The unit for time columns in **sys.dm_exec_query_stats** (for example **total_worker_time**) is microseconds (millionths of a second), but the values are only accurate to milliseconds (thousandths of a second).

With a small change to the previous example, you can use it to find the 10 most expensive queries by the average of logical reads per execution.

Top 10 Most Expensive Cached Plans by Average Logical Reads

```
SELECT TOP(10) OBJECT_NAME(st.objectid, st.dbid) AS obj_name,
              qs.creation_time,
              qs.last_execution_time,
              SUBSTRING      (st.[text],
                                    (qs.statement_start_offset/2)+1,
                                    ((      CASE statement_end_offset
                                            WHEN -1 THEN DATALENGTH(st.[text])
                                            ELSE qs.statement_end_offset
                                            END - qs.statement_start_offset)/2)+1
                                    ) AS sub_statement_text,
              [text],
              query_plan,
              total_worker_time,
              qs.execution_count,
              qs.total_logical_reads / qs.execution_count AS avg_logical_reads,
              qs.total_elapsed_time / qs.execution_count AS avg_duration
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) AS qp
ORDER BY avg_logical_reads DESC;
```

You can easily adapt these queries to return the most expensive queries by any of the measures in **sys.dm_exec_query_stats**, or to limit the results to stored procedure cached plans by using **sys.dm_exec_procedure_stats** in place of **sys.dm_exec_query_stats**.

For full details of all the measures in **sys.dm_exec_query_stats**, see the topic *sys.dm_exec_query_stats (Transact-SQL)* in Microsoft Docs:

sys.dm_exec_query_stats (Transact-SQL)

http://aka.ms/n3fhua

Note: Remember that **sys.dm_exec_query_stats** will only return information about cached plans. Statements that do not have an entry in the plan cache will not appear. This is either because they have been recompiled, or because they have not been executed since the plan cache was last flushed.

Demonstration: Troubleshooting Ad Hoc Plan Caching

In this demonstration, you will see:

- How to turn on the **optimize for ad-hoc workloads** setting.
- The effects of turning on the **optimize for ad-hoc workloads** setting.

Demonstration Steps

1. In SSMS, expand **Queries**, and then open the **Demo 2 - ad-hoc workloads.sql** script file.
2. Execute the statement under the comment that begins **Step 1** to clear the plan cache.
3. Execute the statement under the comment that begins **Step 2** to execute three similar queries. Notice that the only difference between the first and second SELECT queries is an additional space before "43667" in the WHERE clause.
4. Execute the query under the comment that begins **Step 3** to examine the plan cache. Notice that the **query_hash** value is the same for all the plans.
5. Execute the query under the comment that begins **Step 4** to check the current value of the **optimize for ad-hoc workloads** setting. It should be off.
6. Execute the query under the comment that begins **Step 5** to turn on the **optimize for ad-hoc workloads** setting.
7. Execute the query under the comment that begins **Step 1**.
8. Execute the query under the comment that begins **Step 7** to examine the plan cache. Notice that **query_plan** is NULL; only a stub has been added to the plan cache.
9. Execute the query under the comment that begins **Step 8** to rerun one of the SELECT statements from step 2. It is important that you highlight all the text here up to and including GO (but no further) so that the query text exactly matches the first execution.
10. Execute the query under the comment that begins **Step 9** to examine the plan cache. Notice that one of the cached **query_plan** values is no longer NULL. The stub has been removed and a full plan added to the plan cache.
11. Execute the query under the comment that begins **Step 10** to return the **optimize for ad-hoc workloads** setting to its default value.
12. Leave SSMS open for the next demonstration.

Categorize Activity

Place each DMV or DMF into the appropriate category, based on the information it returns. Indicate your answer by writing the category number to the right of each item.

Items	
1	sys.dm_exec_cached_plans
2	sys.dm_exec_query_text
3	sys.dm_exec_query_stats
4	sys.dm_exec_query_plan
5	sys.dm_exec_procedure_stats
6	sys.dm_exec_text_query_plan

Category 1		Category 2		Category 3
Query Execution Plan		Query Text		Execution Statistics

Lesson 3
Automatic Tuning

Automatic tuning is a new feature in SQL Server 2017 that enables performance issues caused by SQL plan choice regressions to be fixed.

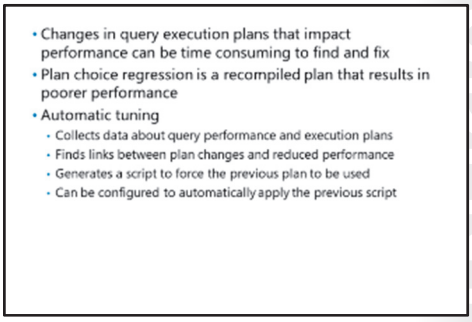
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the purpose of automatic tuning.
- Explain how query plan choice regression can cause performance issues.
- Use the **sys.dm_db_tuning_recommendations** dynamic management view to inspect query tuning recommendations.
- Carry out a manual plan choice selection, and enable automatic plan choice selection.

What is Automatic Tuning?

It can be time consuming to investigate and resolve query performance problems caused by changes in a query execution plan. Automatic tuning simplifies this process by collecting information about plan changes, and identifying and fixing performance issues.



When poor query performance can be linked to a change of query execution plan, automatic tuning reports the problem, allowing you to force the previous, better-performing plan to be used. This can either be configured to happen automatically, or the corrective action can be applied manually.

Plan Choice Regression

In most cases, a query execution plan is compiled the first time it used and then stored in the query plan cache for reuse.

However, a query plan may be recompiled and replaced with new query plan due to schema or statistics changes. If the recompilation results in poorer performance compared to the previous plan, it is referred to as plan choice regression.

Automatic tuning identifies plan choice regression by maintaining records of the relative performance of the previous and current execution plans for a query.

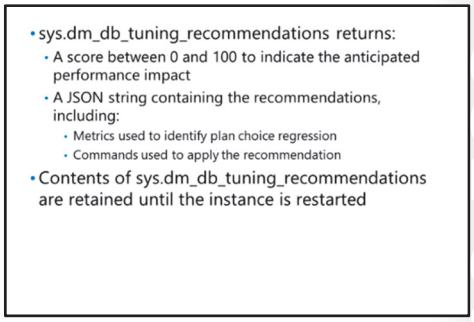
For more information about automatic tuning, see Microsoft Docs:

Automatic tuning

https://aka.ms/Chi2zp

sys.dm_db_tuning_recommendations

When automatic tuning identifies plan choice regression, the details are reported by the system dynamic management view (DMV) **sys.dm_db_tuning_recommendations**. This DMV gives tuning recommendations, and records when and how tuning recommendations are implemented.



The data returned by sys.dm_db_tuning_recommendations includes:

- A score from 0 thru 100 to indicate the anticipated performance impact of the recommendation. The higher the score value, the greater the impact.
- Information about the recommendation. This is a JSON-formatted string and includes metrics used to identify plan choice regression, and the command used to apply the recommendation.

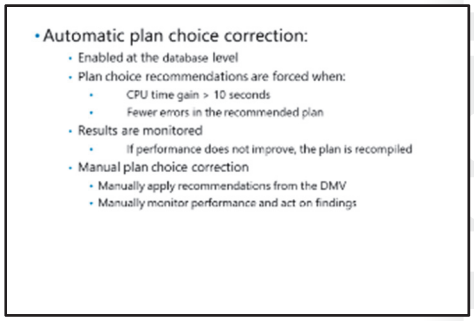
In common with the contents of the query plan cache, the results of **sys.dm_db_tuning_recommendations** are retained until the SQL Server instance is restarted.

For more information about **sys.dem_db_tuning_recommendations**, see Microsoft Docs:

sys.dm_db_tuning_recommendations (Transact-SQL)

https://aka.ms/Oiyrh6

Plan Choice Correction



Automatic Corrections

Automatic plan choice correction is disabled by default. You can change the configuration using ALTER DATABASE.

Use ALTER DATABASE to enable or disable automatic plan choice correction.

Automatic Plan Choice Correction

```
--enable automatic plan choice correction
ALTER DATABASE current SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON);

--disable automatic plan choice correction
ALTER DATABASE current SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = OFF);
```

Automatic tuning will only force a previous query plan to be used if the previous plan:

- Used 10 or more seconds of CPU time than the new plan.
- Contained fewer errors than the new plan.

Once a recommendation has been applied, performance is then monitored. If the forced plan performs no better than the plan it replaced, it is removed causing a new query execution plan to be compiled. If the new plan performs better, it is retained until the next recompile.

Manual plan choice correction

If automatic plan choice correction is disabled, you can manually apply the recommendations returned by **sys.dm_db_tuning_recommendations**. You do this by running the command contained in the JSON block held in the details column.

To extract the relevant command to apply a plan choice recommendation, use the following code:

Extra a Plan Choice Recommendation Command

```
SELECT reason, score,
        script = JSON_VALUE(details, '$.implementationDetails.script')
FROM sys.dm_db_tuning_recommendations;
```

When you manually apply a recommendation, rather than automatic tuning forcing a previous plan, you must monitor the performance yourself. If you don't get the anticipated benefit you must remove the plan yourself. This is all done automatically when automatic tuning is set to ON.

> **Question:** In your organization, how much time is spent trying to fix poor performance after query plans have changed?

Lesson 4
Query Store

The Query Store can be configured to store snapshots of information from the plan cache. The information collected by the Query Store can facilitate performance troubleshooting, because different snapshots of the plan cache can be compared; this will tell you when and why performance of the SQL Server instance, or of a particular query, changed.

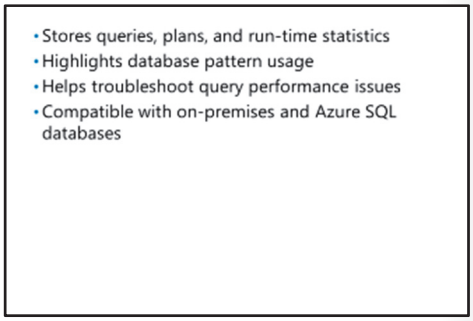
Lesson Objectives

At the end of this lesson, you will be able to:

- Understand the benefits of enabling the Query Store on your databases.
- Turn on the Query Store on a database.
- Configure data collection and data retention by the Query Store.
- Review data collected by the Query Store.
- Use the Query Store to force a query execution plan.

What is the Query Store?

The Query Store automatically gathers a history of queries, plans, and run-time statistics, which it retains for future analysis. By separating the data into time windows, you can identify database usage patterns, see when the query plan was changed on the server, and understand why it was changed.



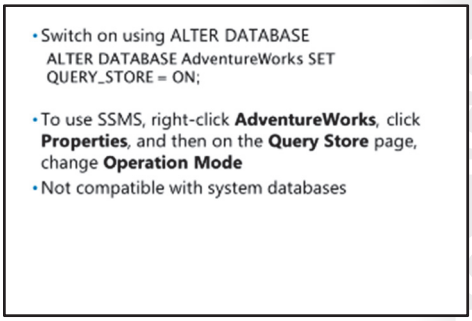
Query Store gives insight into query plan choice and performance. It helps in the troubleshooting of performance issues, because it can show you the changes in query plans.

For more information on the Query Store, see the topic *Monitoring Performance By Using the Query Store* in Microsoft Docs:

Monitoring Performance By Using the Query Store

http://aka.ms/rqkfgg

Enabling the Query Store



Because the Query Store operates at the database level, you can control how much data it gathers. It is not enabled by default, but you can switch it on by using the ALTER DATABASE SET option or the **Query Store** page in the Database Properties window in SQL Server Management Studio (SSMS). Similarly, you can use both methods to disable this feature.

You can switch on the Query Store on the **Consumer** database using the following command:

Enable the Query Store

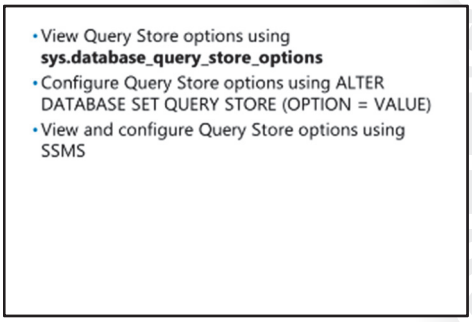
```
ALTER DATABASE AdventureWorks SET QUERY_STORE = ON;
```

To disable the Query Store, use the ALTER DATABASE SET command with the SET QUERY_STORE OFF clause.

In common with many other database-level—and server-level—settings, the state of the Query Store is represented internally by two values; the **Actual** value, and the **Requested** value. The **Actual** value contains the current running value, and the **Requested** value contains the next state requested by an administrator. Typically, both values are the same; they will only differ when an administrator has requested a change of state, but the database engine has not yet implemented the change.

The Query Store cannot be switched on for the **master**, **msdb**, or **tempdb** system databases. It is not an option on the **model** database either, so you cannot set it as a default option for new databases.

Configuring the Query Store



The **sys.database_query_store_options** DMV returns a single row result set with all the Query Store configuration settings for the current database. The DMV will return this row for all databases for which Query Store can be configured, even if the Query Store is not switched on. In system databases, for which Query Store cannot be enabled, **sys.database_query_store_options** returns an empty result set.

You can use the **sys.database_query_store_options** system DMV to query the current Query Store configuration values:

sys.database_query_store_options

```
SELECT * FROM sys.database_query_store_options;
```

For further information about **sys.database_query_store_options**, see the topic *sys.database_query_store_options (Transact-SQL)* in Microsoft Docs:

sys.database_query_store_options (Transact-SQL)

http://aka.ms/tcv61j

Query Store configuration options fall into three categories:

- General
  - **Operation mode**. Read-only, read/write, or off.
- Monitoring
  - **Data flush interval (seconds)**. The frequency with which Query Store data is written to disk.
  - **Statistics collection interval (minutes)**. The interval in which Query Store data is aggregated.
- Data Retention
  - **Maximum size (megabytes)**. The maximum size that the Query Store data may grow to. If the maximum size is reached, the Query Store goes into read-only mode and no more data is collected.
  - **Capture mode**. Controls which data is collected. ALL (all queries), AUTO (expensive queries only), or NONE (no data collected).
  - **Cleanup mode**. When turned on, data is removed from the Query Store as it approaches its maximum size. When the Query Store is at 90 percent of its maximum size, data will be removed to reduce the actual size to 80 percent of the maximum size.
  - **Stale query threshold (days)**. The maximum period that queries to which no policy applies will remain in the Query Store.

All of these settings can be altered using the ALTER DATABASE SET QUERY_STORE command.

A sample query setting the data flush interval to 600 seconds in the **Consumer** database:

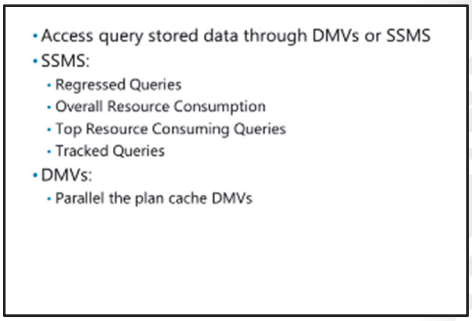
Setting Data Flush Interval

```
ALTER DATABASE Consumer SET QUERY_STORE (DATA_FLUSH_INTERVAL_SECONDS = 600)
```

Query Store settings can also be viewed and amended through the **Query Store** page of the Database Properties window in SSMS.

Accessing Query Store Data

Data collected by the Query Store can be accessed using either SSMS or Transact-SQL queries.



SSMS

When the Query Store is switched on for a database, a Query Store node appears under the database name in SSMS Object Explorer. Under the Query Store node, four further nodes appear, each of which provides access to a reporting window.

- **Regressed Queries**. Shows a report of queries whose performance has reduced during a time period. Performance can be measured by CPU time, duration, logical read count, logical write count, memory consumption, or physical reads. You can view execution plans for each query in the report.
- **Overall Resource Consumption**. Shows histograms of resource consumption during a time period. Histograms can show consumption by CPU time, duration, logical read count, logical write count, memory consumption, or physical reads.
- **Top Resource Consuming Queries**. Shows a report of most expensive queries during a time period. Query cost may be measured by CPU time, duration, logical read count, logical write count, memory consumption, or physical reads. You can view execution plans for each query in the report.
- **Tracked Queries**. Shows the historical Query Store data for a single query.

Transact-SQL

A group of system DMVs are available that expose the data collected by the Query Store. These DMVs are closely related to the plan cache DMVs discussed earlier in this module. Some of the available DMVs are:

- **sys.query_store_runtime_stats**. Similar to **sys.dm_exec_query_stats**, this DMV exposes performance information gathered by the Query Store.
- **sys.query_store_plan**. Similar to **sys.dm_exec_query_plan**, this DMV exposes query plans captured by the Query Store.
- **sys.query_store_query_text**. Similar to **sys.dm_exec_query_text**, this DMV exposes the statement text of queries captured by the Query Store.

For more information on Query Store DMVs, see the topic *Query Store Catalog Views (Transact-SQL)* in Microsoft Docs:

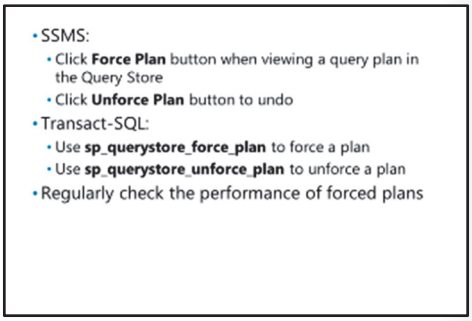
Query Store Catalog Views (Transact-SQL)

http://aka.ms/u60m6c

Forcing Query Execution Plans

You can bypass the query optimizer and specify a query plan for a SELECT statement using the USE PLAN query hint.

The Query Store simplifies this process, because you can use it to select and force a query plan from the historical information it has captured. You can force a plan using SSMS or using Transact-SQL statements.



Note: Whether you force a query plan using a USE PLAN hint or by using the Query Store, you should check that the forced plan is still performing well on a regular basis. Changes in data volume or data distribution may cause the plan to become less optimal over time.

SSMS

A query plan may be forced from three of the four nodes under the Query Store node in SSMS Object Explorer:

- Regressed Queries.
- Top Resource Consuming Queries.
- Tracked Queries.

In any of these windows, you can select a query, select a query plan, and then click the **Force Plan** button. You can also unforce a forced plan using the **Unforce Plan** button.

Transact-SQL

You use the system stored procedure **sp_query_store_force_plan** to force a query plan.

The following example forces **plan_id** 120 for **query_id** 45:

sp_query_store_force_plan example

```
EXEC sp_query_store_force_plan @query_id = 45, @plan_id = 120;
```

A query plan is unforced in a similar way, using the **sp_query_store_unforce_plan** system stored procedure.

The values for **plan_id** and **query_id** are found from the output of **sys.query_store_plan**.

Demonstration: Working with the Query Store

In this demonstration, you will see how to:

- Configure the Query Store.
- Access Query Store data.
- Force and unforce a query execution plan using the Query Store.

Demonstration Steps

1. In SSMS, open the **Demo 3 - query store.sql** script file.
2. In Object Explorer, expand **Databases**, and then expand **TSQL** to show that it has no **Query Store** node.
3. Execute the code under the comment that begins **Step 2** to switch on and configure the Query Store.
4. To start a workload, in File Explorer, right-click **D:\Demofiles\Mod08\start_load_1.ps1**, and then click **Run with PowerShell**. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press Enter. Once the workload has started, continue with the demo.
5. Switch to SQL Server Management Studio.
6. In Object Explorer, right-click **TSQL**, and then click **Properties**. On the **Query Store** page, show that the **Operation Mode (Actual)** option is now set to **Read Write**, then click **Cancel**.
7. In Object Explorer, refresh the list of objects under the **TSQL** database to display the new Query Store node. Expand the **Query Store** node to show the queries being tracked.
8. Execute the code under the comment that begins **Step 6** to expand Query Store storage.
9. In Object Explorer, under **Query Store**, double-click **Overall Resource Consumption**. In the report pane, click **Configure** (top right). In the **Time Interval** section, in the first drop-down box, select **Last hour**, and then click **OK**. You should see some bars at the right-hand side of each graph caused by the workload.
10. In Object Explorer, double-click **Top Resource Consuming Queries**. In the **Metric** drop-down box (top left), select **Execution Count**. The tallest bar in the list should be for the workload query with text starting "SELECT so.custid, so.orderdate, so.orderid, so.shipaddress...". If it is not the tallest bar, locate the bar for this query. Note the query id (visible either on the x-axis of the chart, or in the tooltip shown by hovering over the relevant bar on the chart).
11. In Object Explorer, double-click **Tracked Queries**. In the **Tracking query** box, type the query id you identified in the previous step and then press Enter. This shows the query plan history for the query.
12. In the **Demo 3 - query store.sql** pane, execute the code under the comment that begins **Step 10** to create a temp table and double the number of rows in the **Sales.Orders** table (this should prompt a statistics update and a new query plan).
13. Wait for approximately 60 seconds.
14. After 60 seconds have passed, in the **Tracked Queries** pane, click **Refresh**. You should see that a new query plan has been compiled. If you do not see a new plan, rerun the (INSERT, SELECT, FROM) statements from the previous step and check again.
15. In Object Explorer, double-click **Regressed Queries**. In the report, click **Configure** (top right). In the **Time Interval** section, in the **Recent** drop-down, select **Last 5 minutes**, and then press Enter. The SELECT statement with text starting "SELECT so.custid, so.orderdate, so.orderid, so.shipaddress..." should appear in the report.
16. In the **Tracked Queries** pane, select one of the query plans (the dots shown on the graph), and then click **Force Plan**. In the **Confirmation** dialog box, click **Yes**.
17. In the **Top Resource Consumers** report pane, click **Refresh**. Notice that executions using the forced plan have a tick in the **Tracked Queries** scatter graph.
18. Return to the **Tracked Queries** report. Click the ticked dot (representing the forced plan), then click **Unforce Plan**. In the Confirmation window, click **Yes**.
19. In the **Demo 3 - query store.sql** pane, execute the code under the comment that begins **Step 16** to stop the workload.
20. Close SSMS without saving changes. Press ENTER in the PowerShell® workload window.

Check Your Knowledge

Question	
Which Query Store report contains Transact-SQL statements that show a trend of reduced performance over time?	
Select the correct answer.	
	Overall Resource Consumption
	Tracked Queries
	Top Resource Consuming Queries
	Regressed Queries

Lab: Plan Caching and Recompilation

Scenario

Adventure Works Cycles is a global manufacturer, wholesaler, and retailer of cycle products. The owners of the company have decided to start a new direct marketing arm. It has been created as a new company named Proseware Inc. Even though it has been set up as a separate company, it will receive some IT-related services from Adventure Works and will be provided with a subset of the corporate Adventure Works data. The existing Adventure Works SQL Server platform has been moved to a new server that is capable of supporting both the existing workload and the workload from the new company.

Objectives

At the end of this lab, you will be able to:

- Use the plan cache to identify and resolve query performance issues.
- Use the Query Store to monitor performance, and to force a query plan.

Estimated Time: 60 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Troubleshooting with the Plan Cache

Scenario

Since a new Proseware Inc. application started to interact with the **AdventureWorks** database, you have noticed that the plan cache of the SQL Server instance occupies more memory than before. You will investigate this issue by examining the contents of the plan cache, and try to identify a resolution.

You know that the new Proseware Inc. application interacts with the **AdventureWorks** database using stored procedures.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Start the Workload
3. Check for Plan Cache Bloat
4. Identify the Query Causing Plan Cache Bloat
5. Identify the Stored Procedure Causing Plan Cache Bloat
6. Rewrite Proseware.up_CampaignReport to Prevent Plan Cache Bloat
7. Verify That the Stored Procedure Is Using a Single Query Plan
8. Stop the Workload

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab08\Starter** folder as Administrator.

▶ Task 2: Start the Workload

- In the **D:\Labfiles\Lab08\Starter** folder, execute **start_load_exercise_01.ps1** with PowerShell. If a message is displayed asking you to confirm a change in execution policy, type **Y**. Once the workload script is running, continue with the exercise.

▶ Task 3: Check for Plan Cache Bloat

1. Start **SQL Server Management Studio**, then open the project file D:\Labfiles\Lab08\Starter\Project\Project.ssmssln and the Transact-SQL file **Lab Exercise 01 - plan cache.sql**.
2. Under the comment that begins **Task 2**, execute the query against the **sys.dm_exec_query_stats** system DMV to find the most common **query_hash** executed on the **MIA-SQL** instance.

   Is there any indication of plan cache bloat?

▶ Task 4: Identify the Query Causing Plan Cache Bloat

1. Under the comment that begins **Task 3**, edit the query to return an example **plan_handle** from **sys.dm_exec_query_stats** related to the **query_hash** returned by the previous task. To do this, you must replace the text "<query has from task 1>" with the value of the **query_hash** column returned from task 2.

▶ Task 5: Identify the Stored Procedure Causing Plan Cache Bloat

1. Under the comment that begins **Task 4**, execute the query against **INFORMATION_SCHEMA.ROUTINES** to find the stored procedure that is causing plan cache bloat (**INFORMATION_SCHEMA.ROUTINES** contains the code for database objects).
2. Which stored procedure appears to be the best candidate for causing plan cache bloat?
3. Based on the stored procedure code, can you suggest how you might rewrite the procedure to avoid plan cache bloat?

▶ Task 6: Rewrite Proseware.up_CampaignReport to Prevent Plan Cache Bloat

1. From Solution Explorer, open the query file **Lab Exercise 01a -Proseware.up_CampaignReport.sql**. This file contains the definition of the stored procedure.
2. Change the stored procedure definition to remove the use of dynamic SQL.

▶ Task 7: Verify That the Stored Procedure Is Using a Single Query Plan

1. In SSMS, return to the query window where **Lab Exercise 01 - plan cache.sql** is open.
2. Under the comment that begins **Task 6**, execute the query against **sys.dm_exec_procedure_stats** to show the query plan for **Proseware.up_CampaignReport**.
3. Notice that only one row is returned by the query; this indicates that the stored procedure is using only one query plan.

▶ Task 8: Stop the Workload

1. Highlight the code under the comment that begins **Task 7** and execute it.
2. Press ENTER in the PowerShell workload window to close it.

Results: At the end of this exercise, you will have refactored a stored procedure to reduce plan cache bloat.

Exercise 2: Working with the Query Store

Scenario

Some of the data required by Proseware Inc. applications has been added to a new database called **Proseware**. You must configure the Query Store for the new **Proseware** database.

The main tasks for this exercise are as follows:

1. Start the Workload
2. Enable the Query Store
3. Amend the Query Store Statistics Collection Interval
4. Check the Top Resource Consuming Queries Report
5. Add a Missing Index
6. Force a Query Plan
7. Stop the Workload

▶ Task 1: Start the Workload

- In the **D:\Labfiles\Lab08\Starter** folder, execute **start_load_exercise_02.ps1** with PowerShell. Wait a few minutes before continuing.

▶ Task 2: Enable the Query Store

- Use the SSMS GUI to turn on Query Store for the **ProseWare** database.

▶ Task 3: Amend the Query Store Statistics Collection Interval

- Use the SSMS GUI to change the Query Store Statistics Collection Interval to **1 minute**.

▶ Task 4: Check the Top Resource Consuming Queries Report

1. In SSMS, open the Query Store Top Resource Consuming Queries Report for the **ProseWare** database.
2. Note the **query id** of the top query.

▶ Task 5: Add a Missing Index

1. If it is not already open, open the Transact-SQL file **Lab Exercise 02 - Query Store.sql**.
2. Execute the code under **task 5** to create a missing index.

▶ Task 6: Force a Query Plan

1. Open the Query Store Tracked Queries report, and view the details for the query ID you noted in an earlier task.
2. Select one of the query plans for this query and force it.

▶ Task 7: Stop the Workload

1. Return to the query window where **Lab Exercise 02 - Query Store.sql** is open.
2. Execute the code under **Task 7** to stop the workload.
3. Close SQL Server Management Studio without saving any changes.
4. Close the PowerShell workload window.

Results: At the end of this exercise, you will be able to:

Configure the Query Store.

Use the Query Store to investigate statement query execution plans.

Use the Query Store to force a query execution plan.

Question: Under what circumstances might you consider forcing a query plan on a production SQL Server instance?

Module Review and Takeaways

In this module, you have learned about how query execution plans are cached to enhance the performance of the SQL Server Database Engine. In addition to learning methods for viewing details of cached plans, and influencing query plan selections, you learned how to manage the content of the plan cache.

You also learned about the Query Store, and how it simplifies the process of troubleshooting query plans.

Review Question(s)

Check Your Knowledge

Question	
Which Query Store configuration option determines the frequency with which the Query Store writes data to disk?	
Select the correct answer.	
	OPERATION_MODE
	INTERVAL_LENGTH_MINUTES
	DATA_FLUSH_INTERVAL_SECONDS
	CLEANUP_POLICY

Module 9

Extended Events

Contents:

Module Overview

SQL Server Extended Events is a flexible, lightweight event-handling system built into the Microsoft® SQL Server® Database Engine. This module focuses on the architectural concepts, troubleshooting strategies and usage scenarios of Extended Events.

Note: Extended Events has been available in Microsoft Azure™ SQL Database as a preview feature since October, 2015; at the time of publication, no date has been published for the General Availability (GA) of Extended Events in Azure SQL Database.

For more information about Extended Events in Azure SQL Database, see:

Extended Events in SQL Database

http://aka.ms/tzaa5b

Objectives

After completing this module, you will be able to:

- Describe Extended Events core concepts.
- Create and query Extended Events sessions.

Lesson 1
Extended Events Core Concepts

This lesson focuses on the core concepts of Extended Events—the architectural design of the Extended Events engine and core concepts of Extended Events are covered in depth.

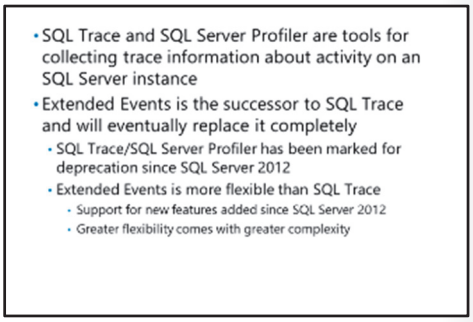
Lesson Objectives

After completing this lesson, you will be able to:

- Explain the differences between SQL Server Profiler, SQL Trace, and Extended Events.
- Describe Extended Events architecture.
- Define Extended Events packages.
- Define Extended Events events.
- Define Extended Events predicates.
- Define Extended Events actions.
- Define Extended Events targets.
- Define Extended Events sessions.
- Define Extended Events types and maps.

Extended Events, SQL Trace, and SQL Server Profiler

Extended Events, SQL Trace, and SQL Server Profiler are all tools that you can use to monitor SQL Server events.



SQL Trace

SQL Trace is a server-side, event-driven activity monitoring tool; it can capture information about more than 150 event classes. Each event returns data in one or more columns and you can filter column values. You configure the range of events and event data columns in the trace definition. You can also configure the destination for the trace data, a file or a database table, in the trace definition.

SQL Trace is included in SQL Server 7.0 and later versions.

SQL Server Profiler

SQL Server Profiler is a GUI for creating SQL traces and viewing data from them. SQL Server Profiler is included in SQL Server 7.0 and later versions.

As established parts of the SQL Server platform, SQL Server Profiler and SQL Trace are familiar to many SQL Server administrators.

Note: SQL Trace and SQL Server Profiler have been marked for deprecation since SQL Server 2012. Microsoft has declared an intention to remove both tools in a future version of SQL Server. Extended Events is now the recommended activity tracing tool.
Because it is marked for deprecation, SQL Trace does not include event classes for many features added in SQL Server 2012 onwards.

Extended Events

Like SQL Trace, Extended Events is an event-driven activity monitoring tool; however, it attempts to address some of the limitations in the design of SQL Trace by following a loose-coupled design pattern. Events and their targets are not tightly coupled; any event can be bound to any target. This means that data processing and filtering can be carried out independently of data capture. In most cases, this results in Extended Events having a lower performance overhead than an equivalent SQL Trace.

With Extended Events, you can define sophisticated filters on captured data. In addition to using value filters, you can filter events by sampling and data can be aggregated at the point it is captured. You can manage Extended Events either through a GUI in SQL Server Management Studio (SSMS) or by using Transact-SQL statements.

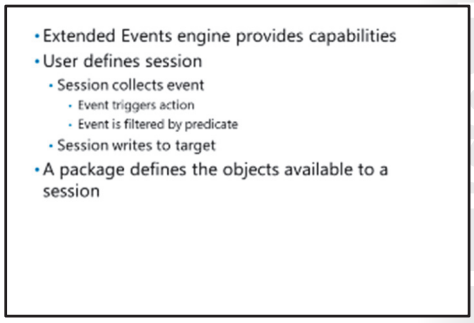
You can integrate Extended Events with the Event Tracing for Windows (ETW) framework, so that you can monitor SQL Server activity alongside other Windows® components.

Extended Events was introduced in SQL Server 2008; since the deprecation of SQL Trace and SQL Server Profiler was announced with the release of SQL Server 2012, many features introduced in SQL Server 2012, 2014 and 2016 can only be traced using Extended Events.

The additional flexibility of Extended Events comes at the cost of greater complexity.

Extended Events Architecture

The Extended Events engine is a collection of services, running in the database engine, that provide the resources necessary for events to be defined and consumed.



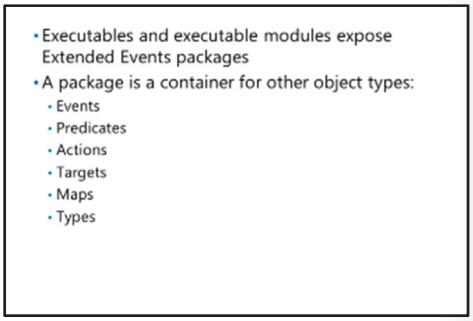
As a user of Extended Events, you might find it most helpful to think about Extended Events primarily in terms of the session object. A session defines the Extended Events data that you want to collect, how the data will be filtered, and how the data will be stored for later analysis. Sessions are the top-level object through which you will interact with Extended Events:

- User defines **session**
  - **Session** includes **event**
    - **Event** triggers **action**
    - **Event** is filtered by **predicate**
  - **Session** writes to **target**

A list of sessions is maintained by the Extended Events engine. You can define and modify sessions using Transact-SQL or in SSMS. You can view data collected by active sessions using Transact-SQL—in which case the data is presented in XML format—or using SSMS.

Packages

Packages act as containers for the Extended Events objects and their definitions; a package can expose any of the following object types:



- Events
- Predicates
- Actions
- Targets
- Types
- Maps

Packages are contained in a module that exposes them to the Extended Events engine. A module can contain one or more packages and can be compiled as an executable or DLL file.

A complete list of packages registered on the server can be viewed using the **sys.dm_xe_packages** DMV:

sys.dm_xe_packages

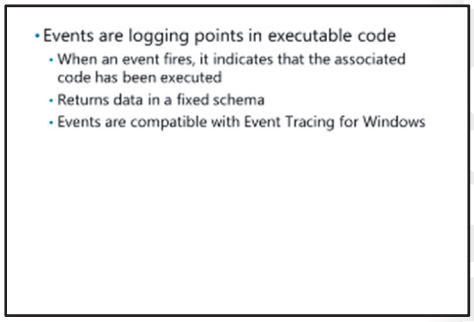
```
SELECT * FROM sys.dm_xe_packages;
```

For more information on **sys.dm_xe_packages**, see the topic *sys.dm_xe_packages (Transact-SQL)* in Microsoft Docs:

sys.dm_xe_packages (Transact-SQL)

http://aka.ms/i4j6vf

Events



Events are points in the code of a module that are of interest for logging purposes. When an event fires, it indicates that the corresponding point in the code was reached. Each event type returns information in a well-defined schema when it occurs.

All available events can be viewed in the **sys.dm_xe_objects** DMV under the **event** object_type:

sys.dm_xe_objects; events

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'event';
```

Events are defined by the Event Tracing for Windows model; This means that SQL Server Extended Events can be integrated with ETW. Like ETW events, Extended Events are categorized by:

- **Channel**. The event channel identifies the target audience for an event. These channels are common to all ETW events:
  - **Admin**. Events for administration and support.
  - **Operational**. Events for problem investigation.
  - **Analytic**. High-volume events used in performance investigation.
  - **Debug**. ETW developer debugging events.
- **Keyword**. An application-specific categorization. In SQL Server, Extended Events event keywords map closely to the grouping of events in a SQL Trace definition.

A complete list of event keywords can be returned from **sys.dm_xe_map_values**:

Extended Events event keywords

```
SELECT map_value AS keyword
FROM sys.dm_xe_map_values
WHERE name = 'keyword_map'
ORDER BY keyword;
```

When you add, amend or remove an event from a package, you must refer to it with a two-part name; *package name.event name*.

A complete list of events and their package names can be returned by joining the list of events returned by the first example in this lesson to **sys.dm_xe_packages**:

Event and package names

```
SELECT xp.name AS package_name,
xo.name AS event_name,
xo.[description] AS event_description
FROM sys.dm_xe_objects AS xo
JOIN sys.dm_xe_packages AS xp
ON xp.guid = xo.package_guid
WHERE object_type = 'event'
ORDER BY package_name, event_name;
```

To find all the attribute columns associated with an event, you should join **sys.dm_xe_objects** to **sys.dm_xe_object_columns**:

Extended Events Object Columns

```
SELECT xoc.* FROM sys.dm_xe_objects AS xo
JOIN sys.dm_xe_object_columns AS xoc
ON xoc.object_package_guid = xo.package_guid
AND xoc.object_name = xo.name
WHERE xo.object_type = 'event';
```

For more information on **sys.dm_xe_objects**, see the topic *sys.dm_xe_objects (Transact-SQL)* in Microsoft Docs:

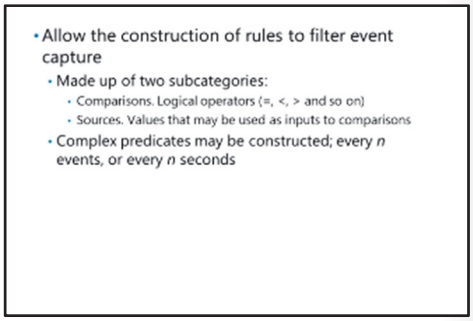
sys.dm_xe_objects (Transact-SQL)

http://aka.ms/bwkcmu

Predicates

Predicates are logical rules with which events can be selectively captured, based on criteria you specify. Predicates divide into two subcategories:

- **Predicate comparisons**. Comparison operators, such as "equal to", "greater than", and "less than", which may make up a predicate filter. All predicate comparisons return a Boolean result (true or false).
- **Predicate sources**. Data items that may be used as inputs to predicate comparisons. These are similar to the column filters available when defining a SQL trace.



In addition to building logical rules, predicates are capable of storing data in a local context, which means that predicates based on counters can be constructed; for example, every *n* events or every *n* seconds.

Predicates are applied to an event using a WHERE clause which functions like the WHERE clause in a Transact-SQL query.

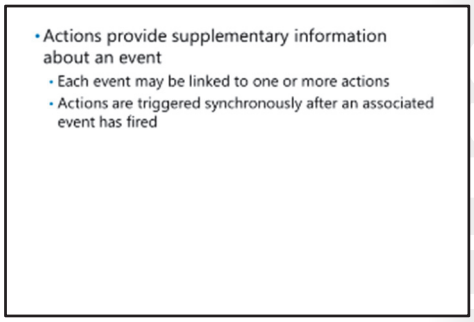
All available predicates can be viewed in the DMV **sys.dm_xe_objects** under the **object_type** values **pred_source** and **pred_compare**:

sys.dm_xe_objects; predicates

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type LIKE 'pred%'
ORDER BY object_type, name;
```

Actions

Actions are responses to an event; you can use these responses to collect supplementary information about the context of an event at the time an event occurs. Each event may have a unique set of one or more actions associated with it. When an event occurs, any associated actions are raised synchronously.



Note: You might find the name of this object to be misleading; Extended Events actions do not allow you to define responses to an event. Instead, actions are additional steps that occur within the Extended Events engine when an event is triggered. Most actions provide more data to be collected about an event.

SQL Server defines more than 50 different actions, which include:

- **Collect database ID**
- **Collect T-SQL stack**
- **Collect session ID**
- **Collect session's NT username**
- **Collect client hostname**

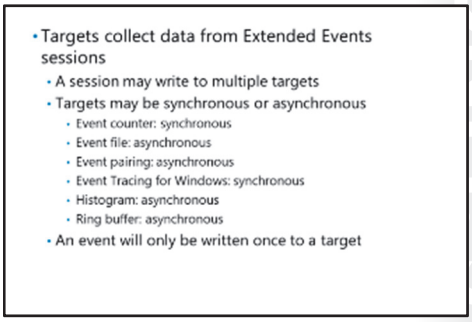
All available actions can be viewed in the DMV **sys.dm_xe_objects** under the **object_type** value action:

sys.dm_xe_objects; actions

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'action';
```

Targets

Targets are the Extended Events objects that collect data. When an event is triggered, the associated data can be written to one or more targets. A target may be updated synchronously or asynchronously. The following targets are available for Extended Events:



- **Event counter**. The counter is incremented each time an event associated with a session occurs—synchronous.
- **Event file**. Event data is written to a file on disk—asynchronous.
- **Event pairing**. Tracks when events that normally occur in pairs (for example, lock acquired and lock released) do not have a matching pair—asynchronous.
- **Event Tracing for Windows**. Event data is written to an ETW log—synchronous.
- **Histogram**. A more complex counter that partitions counts by an event or action value—asynchronous.
- **Ring buffer**. A first-in, first-out (FIFO) in-memory buffer of a fixed size—asynchronous.

The design of Extended Events is such that an event will only be written once to a target, even if multiple sessions are configured to send that event to the same target.

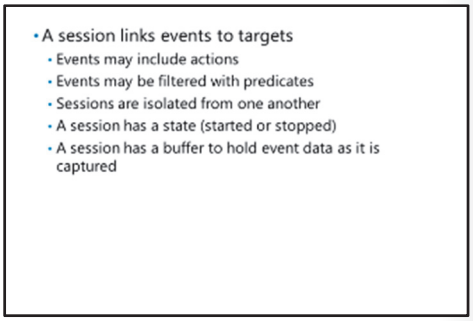
All available targets can be viewed in the DMV **sys.dm_xe_objects** under the **object_type** value **target**:

sys.dm_xe_objects; targets

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'target';
```

Sessions

A session links one or more events to one or more targets. You can configure each event in a session to include one or more actions, and to be filtered with one or more predicates. Once defined, a session can be started or stopped as required; it is possible to configure a session to start when the database engine starts.



A session may include events from more than one package. Sessions are isolated from one another; multiple sessions may use the same events and targets in different ways, without interfering with one another.

A session is configured with a buffer in which event data is held while a session is running, before it is dispatched to the session targets. The size of this buffer is configurable, as is a dispatch policy (how long data will be held in the buffer). You can also configure whether or not to permit data loss from the buffer if event data arrives faster than it can be processed and dispatched to the session target.

All active Extended Events sessions can be viewed in the DMV **sys.dm_xe_sessions**:

sys.dm_xe_sessions

```
SELECT * FROM sys.dm_xe_sessions;
```

For more information on the set of DMVs for accessing information about active Extended Events sessions, including **sys.dm_xe_sessions**, see the topic *Extended Events Dynamic Management Views* in Microsoft Docs:

Extended Events Dynamic Management Views

http://aka.ms/Imlj06

All Extended Events sessions defined on a server can be returned by querying the DMV **sys.server_event_sessions**:

sys.server_event_sessions

```
SELECT * FROM sys.server_event_sessions;
```

For more information on the set of DMVs for accessing definitions for all Extended Events sessions, including **sys.server_event_sessions**, see the topic *Extended Events Catalog Views (Transact-SQL)* in Microsoft Docs:

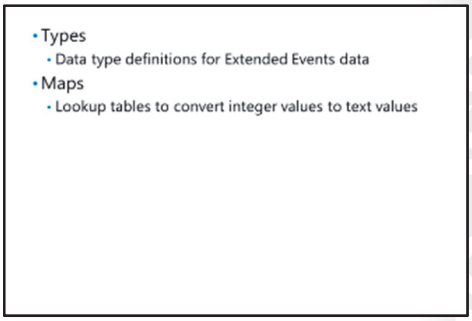
Extended Events Catalog Views (Transact-SQL):

http://aka.ms/Cqon4y

Note: A session can be created without targets, in which case the session data will only be visible through the **Watch Live Data** feature of SSMS.

Types and Maps

Types and maps are metadata objects that make it easier to work with Extended Events data. Types and maps are not directly referenced in an Extended Events session definition.



Types

Internally, Extended Events data is held in binary. A type identifies how a binary value should be interpreted and presented when the data is queried.

All available types can be viewed in the DMV **sys.dm_xe_objects** under the **object_type** value **type**:

sys.dm_xe_objects; types

```
SELECT * FROM sys.dm_xe_objects
WHERE object_type = 'type';
```

Maps

A map is a lookup table for integer values. Internally, many event and action data values are stored as integers; maps link these integer values to text values that are easier to interpret.

All available types can be viewed in the DMV **sys.dm_xe_map_values**:

sys.dm_xe_map_values

```
SELECT * FROM sys.dm_xe_map_values
ORDER BY name, map_key;
```

Demonstration: Creating an Extended Events Session

In this demonstration, you will see how to create an Extended Events session.

Demonstration Steps

1. Start the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Demofiles\Mod09** folder, run **Setup.cmd** as Administrator.
3. Click **Yes** in the User Account Control window and wait for the script to finish.
4. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine instance using Windows authentication.
5. On the **File** menu, point to **Open**, and then click **Project/Solution**. In the **Open Project** dialog box, navigate to the **D:\Demofiles\Mod09** folder, click **Demo.ssmssln**, and then click **Open**.
6. In Solution Explorer, double-click **Demo 1 – create xe session.sql**.
7. Select code under the comment that begins **-- Step 1**, and then click **Execute** to create an Extended Events session.

8. Select code under the comment that begins **-- Step 2**, and then click **Execute** to verify that the session metadata is visible.
9. Select code under the comment that begins **-- Step 3**, and then click **Execute** to start the session and execute some queries.
10. Select code under the comment that begins **-- Step 4**, and then click **Execute** to query the session data.
11. Select code under the comment that begins **-- Step 5**, and then click **Execute** to refine the session data query.
12. In Object Explorer, under **MIA-SQL (SQL Server 13.0.1000 - ADVENTUREWORKS\Student)**, expand **Management**, expand **Extended Events**, and then expand **Sessions**.
13. Expand **SqlStatementCompleted**, and then double-click **package0.ring_buffer**.
14. In the **Data** column, click the XML value, and note that this is the same data that is returned by the query under the comment that begins **-- Step 4** (note that additional statements will have been captured because you ran the code earlier).
15. In Object Explorer, right-click **SqlStatementCompleted**, and then click **Watch Live Data**.
16. In the **Demo 1 – create xe sessions.sql** query pane, select the code under the comment that begins **-- Step 7**, and then click **Execute** to execute some SQL statements.
17. Return to the **MIA-SQL – SqlStatementCompleted: Live Data** pane. Wait for the events to be captured and displayed; this can take a few seconds. Other SQL statements from background processes might be captured by the session.
18. In the Demo 1 – create xe sessions.sql query pane, select the code under the comment that begins **-- Step 8**, and then click **Execute** to stop the session.
19. In Object Explorer, right-click **SqlStatementCompleted**, and then click **Properties**. Review the settings on the **General**, **Events**, **Data Storage** and **Advanced** pages, if necessary referring back to the session definition under the comment that begins **-- Step 1**.
20. In the **Session Properties** dialog box, click **Cancel**.
21. Select the code under the comment that begins **-- Step 10**, and then click **Execute** to drop the session.
22. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Question	
Which system DMV provides the list of events configured in an active Extended Events session?	
Select the correct answer.	
	sys.dm_xe_session _targets
	sys.dm_xe_session_events
	sys.dm_xe_sessions
	sys.dm_xe_session_event_actions

Lesson 2
Working with Extended Events

This lesson discusses using Extended Events. It covers common scenarios in which you might create Extended Events sessions for troubleshooting and performance optimization, as well as the system_health Extended Events session, which captures several events relevant to performance tuning.

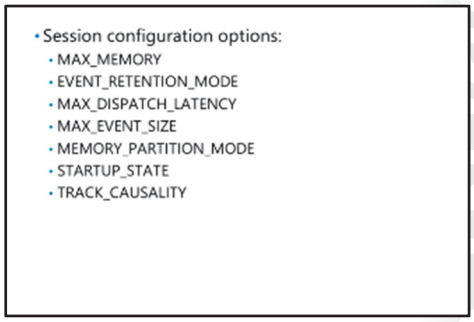
Lesson Objectives

At the end of this lesson, you will be able to:

- Configure Extended Events sessions.
- Configure Extended Events targets.
- Explain the system_health Extended Events session.
- Describe usage scenarios for Extended Events.
- Describe best practices for using Extended Events.

Configuring Sessions

As you have learned, Extended Events sessions are composed from several other object types, primarily events and targets. Sessions also have a number of configuration options that are set at session level:



- MAX_MEMORY. The amount of memory allocated to the session for use as event buffers, in kilobytes. The default value is 4 MB.
- EVENT_RETENTION_MODE. Specifies how the session will behave when the event buffers are full and further events occur:
  - ALLOW_SINGLE_EVENT_LOSS. An event can be dropped from the session if the buffers are full. A compromise between performance and data loss, this is the default value.
  - ALLOW_MULTIPLE_EVENT_LOSS. Full event buffers containing multiple events can be discarded. Minimal performance impact, but high data loss.
  - NO_EVENT_LOSS. Events are never discarded; tasks that trigger events must wait until event buffer space is available. Potential for severe performance impact, but no data loss.
- MAX_DISPATCH_LATENCY. The amount of time events will be held in event buffers before being dispatched to targets—defaults to 30 seconds. You may set this value to INFINITE, in which case the buffer is only dispatched when it is full, or the session is stopped.
- MAX_EVENT_SIZE. For single events larger than the size of the buffers specified by MAX_MEMORY, use this setting. If a value is specified (in kilobytes or megabytes), it must be greater than MAX_MEMORY.

- MEMORY_PARTITION_MODE
  - NONE. Memory is not partitioned. A single group of event buffers are created.
  - PER_NODE. A group of event buffers is created per NUMA node.
  - PER_CPU. A group of event buffers is created per CPU.
- STARTUP_STATE. When set to ON, the session will start when SQL Server starts. The default value is OFF.
- TRACK_CAUSALITY. When set to ON, an identifier is added to each event identifying the task that triggered the event. With this, you can determine whether one event is caused by another.

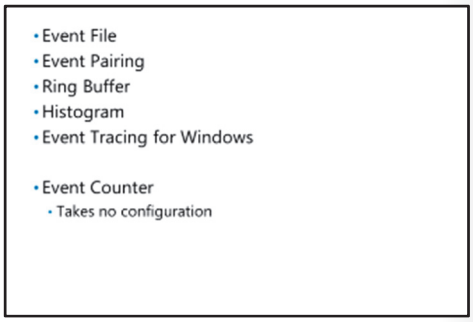
For more information about configuring a session through Transact-SQL, see the topic *CREATE EVENT SESSION (Transact-SQL)* in Microsoft Docs:

CREATE EVENT SESSION (Transact-SQL)

http://aka.ms/b2eo2i

Configuring Targets

Several Extended Events targets take configuration values when they are added to a session.



Event File

The event file target can be used to write session data to a file. It takes the following configuration parameters:

- **filename**. The file name to write to; this can be any valid file name. If a full path is not specified, the file will be created in the \MSSQL\Log folder of the SQL Server instance on which the session is created.
- **max_file_size**. The largest size that the file may grow to; the default value is 1 GB.
- **max_rollover_files**. The number of files that have reached max_file_size to retain. The oldest file is deleted when this number of files is reached.
- **increment**. The file growth increment, in megabytes. The default value is twice the size of the session buffer.

For more information on configuring the event file target, see the topic *Event File Target* in MSDN:

Event File Target

http://aka.ms/ixau4l

Event Pairing

The event pairing target is used to match events that occur in pairs (for example, statement starting and statement completing, or lock acquired and lock released), and report on beginning events that have no matching end event. It takes the following configuration parameters:

- **begin_event**. The beginning event name of the pair.
- **end_event**. The end event name of the pair.
- **begin_matching_columns**. The beginning event columns to use to identify pairs.
- **end_matching_columns**. The ending event columns to use to identify pairs.
- **begin_matching_actions**. The beginning event actions to use to identify pairs.
- **end_matching_actions**. The ending event actions to use to identify pairs.
- **respond_to_memory_pressure**. Permit the target to discard events (and so reduce memory consumption) when memory is under pressure.
- **max_orphans**. The maximum number of unpaired events the target will collect. The default value is 10,000. When this number is reached, events in the target are discarded on a first-in, first-out basis.

For more information on configuring the event pairing target, see the topic *Event Pairing Target* in the SQL Server 2016 Technical Documentation:

Event Pairing Target

http://aka.ms/lj7gng

Ring Buffer

The ring buffer target is used to write session data into a block of memory. When the allocated memory is full, the oldest data in the buffer is discarded and new data is written in its place. The ring buffer target takes the following configuration parameters:

- **max_memory**. The maximum amount of memory the ring buffer may use, in kilobytes.
- **max_event_limit**. The maximum number of events the ring buffer may hold. The default value is 1,000.
- **occurrence_number**. The number of events of each type to keep in the ring buffer. When events are discarded from the buffer, this number of each event type will be preserved. The default value, zero, means that events are discarded on a pure first-in, first-out basis.

Events are dropped from the buffer when either the max_memory or max_event_limit value is reached.

For more information on configuring the ring buffer target, see the topic *Ring Buffer Target* in MSDN:

Ring Buffer Target

http://aka.ms/y3c84h

Histogram

The histogram target is used to partition a count of events into groups based on a specified value. It takes the following configuration parameters:

- **slots**. The maximum number of groups to retain. When this number of groups is reached, new values are ignored. Optional.
- **filtering_event_name**. The event that will be counted into groups. Optional—if not supplied, all events are counted.
- **source_type**. The type of object used for grouping—0 for an event, 1 for an action.
- **source**. The event column or action that is used to create group names and partition the count of events.

For more information on configuring the histogram target, see the topic *Histogram Target* in MSDN:

Histogram Target

http://aka.ms/j9qkw9

Event Tracing for Windows

The event tracing for Windows target is used to write session data to an ETW log. It takes the following configuration parameters:

- **default_xe_session_name**. The name for the ETW session. There can only be one ETW session on a machine, which will be shared between all instances of SQL Server. The default value is XE_DEFAULT_ETW_SESSION.
- **default_etw_session_logfile_path**. The path for the ETW log file. The default value is %TEMP%\XEEtw.etl.
- **default_etw_session_logfile_size_mb**. The log file size, in megabytes. The default value is 20 MB.
- **default_etw_session_buffer_size_kb**. The event buffer size. The default value is 128 KB.
- **retries**. The number of times to retry publishing to ETW before discarding the event. The default is 0.

For more information on configuring the Event Tracing for Windows target, see the topic *Event Tracing for Windows Target* in Microsoft Docs:

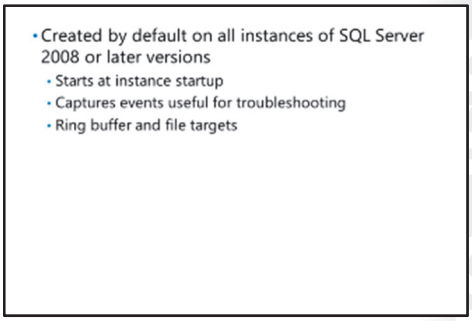
Event Tracing for Windows Target

http://aka.ms/r6e7au

Event Counter

The event counter target is used to count events in a session. It takes no configuration parameters.

The system_health Extended Events Session



The system_health Extended Events session is created by default when a SQL Server 2008 or later version database engine instance is installed. The session is configured to start automatically when the database engine starts. The system_health session is configured to capture a range of events that are relevant for troubleshooting common SQL Server issues. In SQL Server, these include:

- Details of deadlocks that are detected, including a deadlock graph.
- The **sql_text** and **session_id** when an error that has a severity of 20 (or higher) occurs.
- The **sql_text** and **session_id** for sessions that encounter a memory-related error.
- The **callstack**, **sql_text**, and **session_id** for sessions that have waited for more than 15 seconds on selected resources (including latches).
- The **callstack**, **sql_text**, and **session_id** for any sessions that have waited for 30 seconds or more for locks.
- The **callstack**, **sql_text**, and **session_id** for any sessions that have waited for a long time for preemptive waits. (A preemptive wait occurs when SQL Server is waiting for external API calls to complete. The trigger time varies by wait type).
- The **callstack** and **session_id** for CLR allocation and virtual allocation failures (when insufficient memory is available).
- A record of any nonyielding scheduler problems.
- The **ring_buffer** events for the memory broker, scheduler monitor, memory node OOM, security, and connectivity. This tracks when an event is added to any of these ring buffers.
- System component results from **sp_server_diagnostics**.
- Instance health collected by **scheduler_monitor_system_health_ring_buffer_recorded**.
- Connectivity errors using **connectivity_ring_buffer_recorded**.
- Security errors using **security_error_ring_buffer_recorded**.

The system_health session writes data to two targets:

- A ring buffer target, configured to hold up to 5,000 events and to occupy no more than 4 MB.
- An event file target, composed of up to four files of 5 MB each.

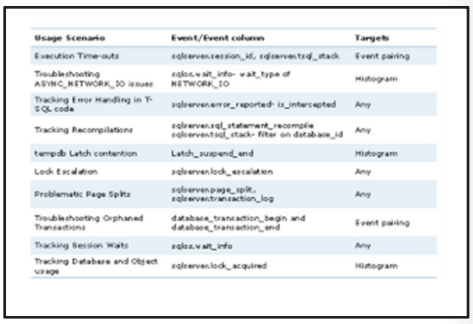
Note: The details of the system_health session are best understood by looking at its definition. You can generate a definition from SSMS:

1. Connect SSMS Object Explorer to any SQL Server instance on which you have administrative rights.
2. In the Object Explorer pane, expand **Management**, expand **Extended Events**, and then expand **Sessions**.
3. Right-click on the **system_health** node, click **Script As**, click **CREATE TO**, and then click **New Query Editor Window**. A script to recreate the system_health session will be generated.

Because both targets are configured to roll over and discard the oldest data they contain when they are full, the system_health session will only contain the most recent issues. On instances of SQL Server where the system_health session is capturing a lot of events, the targets may roll over before you can examine specific events.

Usage Scenarios for Extended Events

Extended Events can be used to troubleshoot many common performance issues.



Usage Scenario	Event/Event column	Targets
Execution Time-outs	sqlserver.session_id, sqlserver.tsql_stack	Event pairing
Troubleshooting ASYNC_NETWORK_IO issues	sqlos.wait_info- wait_type of NETWORK_IO	Histogram
Tracking Error Handling in T-SQL code	sqlserver.error_reported- is_intercepted	Any
Tracking Recompilations	sqlserver.sql_statement_recompile sqlserver.tsql_stack- filter on database_id	Any
tempdb Latch contention	Latch_suspend_end	Histogram
Lock Escalation	sqlserver.lock_escalation	Any
Problematic Page Splits	sqlserver.page_split, sqlserver.tranzaction_log	Any
Troubleshooting Orphaned Transactions	database_transaction_begin and database_transaction_end	Event pairing
Tracking Session Waits	sqlos.wait_info	Any
Tracking Database and Object usage	sqlserver.lock_acquired	Histogram

Execution Time-outs

When a Transact-SQL statement runs for longer than the client application's command time-out setting, a time-out error will be raised by the client application. Without detailed client application logging, it may be difficult to identify the statement causing a time-out.

This scenario is an ideal use case for the Extended Events event pairing target, using either of the following pairs:

- **sqlserver.sp_statement_starting** and **sqlserver.sp_statement_completed** (for systems using stored procedures for database access).
- **sqlserver.sql_statement_starting** and **sqlserver.sql_statement_completed** (for systems using ad hoc SQL for database access).

When a time-out occurs, the starting event will have no corresponding completed event, and will be returned in the output of the event pairing target.

Troubleshooting ASYNC_NETWORK_IO

The ASYNC_NETWORK_IO wait type occurs when the database engine is waiting for a client application to consume a result set. This can occur because the client application processes a result set row-by-row as it is returned from the database server.

To troubleshoot this issue with Extended Events, capture the **sqlos.wait_info** event, filtering on **wait_type** value NETWORK_IO. The histogram target might be suitable for this investigation, using either the client application name or the client host name to define histogram groups.

Tracking Errors and Error Handling in Transact-SQL

Errors may be handled in Transact-SQL code by using TRY…CATCH blocks. Every error raises an event in Extended Events; this includes the errors handled by the TRY…CATCH blocks. You might want to capture all unhandled errors, or track the most commonly occurring errors whether or not they are handled.

The **sqlserver.error_reported** event can be used to track errors as they are raised. The **is_intercepted** column can be used to identify an error is handled in a TRY…CATCH block.

Tracking Recompilations

Query execution plan recompilations occur when a plan in the plan cache is discarded and recompiled. High numbers of plan recompilations might be an indicator of a performance problem, and may cause CPU pressure. Windows performance counters can be used to track overall recompilation counts for a SQL Server instance, but more detail may be needed to investigate further.

In Extended Events, the **sqlserver.sql_statement_recompile** event can provide detailed information, including the cause of recompilation.

The histogram target can be used for tracking recompilations. Group on **source_database_id** to identify the database with the highest number of recompilations in an instance. Group on **statement/object_id** to find the most commonly recompiled statements.

tempdb Latch Contention

Latch contention in **tempdb** can occur due to contention for the allocation bitmap pages when large numbers of temporary objects are being created or deleted. This causes **tempdb** performance problems because all allocations in **tempdb** are slowed down.

The **latch_suspend_end** event tracks the end of latch waits by **database_id**, **file_id**, and **page_id**. With the predicate **divide_evenly_by_int64**, you can capture the contention specifically on allocation pages, because the different allocation bitmap pages occur at regular intervals in a database data file. Grouping the events using the histogram target should make it easier to identify whether latch waits are caused by contention for allocation bitmap pages.

Tracking Lock Escalation

Lock escalation occurs when more than 5,000 locks are required in a single session or under certain memory conditions.

The **sqlserver.lock_escalation** event provides the lock escalation information.

Tracking Problematic Page Splits

Page splits are of two types:

- Mid-page splits.
- Page splits for new allocations.

Mid-page splits create fragmentation and more transaction log records due to data movement.

Tracking page splits alone using the **sqlserver.page_split** event is inefficient as it does not differentiate the problematic mid-page splits and normal allocation splits. The **sqlserver.transaction_log** event can be used for tracking **LOP_DELETE_SPLIT** operation to identify the problematic page splits. A histogram target might be most suitable for this task, grouping either on **database_id** (to find the database with the most page splits) or, within a single database, on **alloc_unit_id** (to find the indexes with the most page splits).

Troubleshooting Orphaned Transactions

Orphaned transactions are open transactions where the transaction is neither committed nor rolled back. An orphaned transaction may hold locks and lead to more critical problems like log growth and blocking, potentially leading to a block on the whole SQL Server instance.

The **database_transaction_begin** and **database_transaction_end** events can be used with an event pairing target to identify the orphaned transactions. The **tsql_frame** action can be used to identify the line of code where the orphaned transaction started.

Tracking Session-Level Wait Stats

The wait stats available from the **sys.dm_os_wait_stats** DMV are aggregated at instance level, so it's not a fine-grained troubleshooting tool. Although you can track wait stats by session with the additional **sys.dm_exec_session_wait_stats** DMV on SQL Server, this may not be suitable for use in a busy system with many concurrent database sessions.

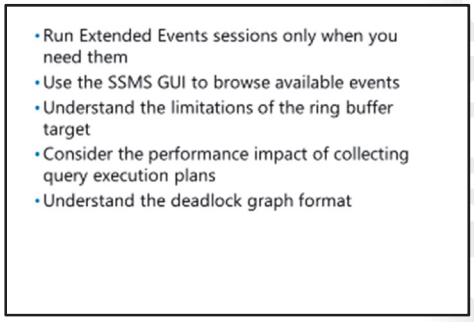
The **sqlos.wait_info** event can be used to track waits across multiple concurrent sessions.

Tracking Database and Object Usage

Tracking database and objects usage helps to identify the most frequently used database and most frequently used objects within a database. You might use this information to guide your optimization efforts, or to prioritize objects for migration to faster storage or memory-optimized tables.

The **sqlserver.lock_acquired** event can help with tracking the usage in most cases. For database usage, a histogram can target grouping on **database_id**. Object usage can be tracked by tracking SCH_M or SCH_S locks at the object resource level by grouping on **object_id** in a histogram target.

Extended Events Best Practices



Run Extended Events Sessions Only When You Need Them

Although Extended Events is a lightweight logging framework, each active session has an overhead of CPU and memory resources. You should get into the practice of only running Extended Events sessions you have created when you have to troubleshoot specific issues.

Use the SSMS GUI to Browse Available Events

The Events page of the Extended Events GUI in SSMS brings all the metadata about individual events together into one view; this view makes understanding the information that Extended Events makes available to you easier than querying the DMVs directly.

Understand the Limitations of the Ring Buffer Target

When using a ring buffer target, be aware that you might not always be able to view all the events contained in the ring buffer. This is due to a limitation of the **sys.dm_xe_session_targets** DMV; the DMV is restricted to displaying 4 MB of formatted XML data. Because Extended Events data is stored internally as unformatted binary, it is possible that the data in a ring buffer will, when converted to formatted XML, exceed the 4 MB limit of **sys.dm_xe_session_targets**.

You can test for this effect by comparing the number of events returned from a ring buffer in XML with the count of events returned in the XML document header, or check the value of the **truncated** attribute in the XML header.

In this example, the query is comparing these values for the system_health session:

Ring buffer: number of events in XML compared with header

```
SELECT x.target_data.value('(RingBufferTarget/@eventCount)[1]', 'int') AS event_count,
                           x.target_data.value('count(RingBufferTarget/event)', 'int')
AS node_count,
                           x.target_data.value('(RingBufferTarget/@truncated)[1]',
'bit') AS output_truncated
FROM  (      SELECT CAST(target_data AS xml) AS target_data
                                        FROM sys.dm_xe_sessions AS xs
                                        JOIN sys.dm_xe_session_targets AS xst
                                        ON xs.address = xst.event_session_address
                                        WHERE xs.name = N'system_health'
                                        AND xst.target_name = N'ring_buffer' ) AS x
```

To avoid this effect, you can:

- Use a file-based target (event file target or ETW target).
- Reduce the size of the MAX_MEMORY setting for the ring buffer to reduce the likelihood that the formatted data will exceed 4 MB. No single value is guaranteed to work; you may have to try a setting and be prepared to adjust it to minimize the truncation effect while still collecting a useful volume of data in the ring buffer.

Note: This effect is not strictly limited to the ring buffer target; it can occur on any target that stores output in memory buffers (ring buffer target, histogram target, event pairing target, and event counter target). However, it is most likely to affect the ring buffer target because it stores unaggregated raw data. All the other targets using memory buffers contain aggregated data, and are therefore less likely to exceed 4 MB when formatted.

Consider the Performance Impact of Collecting Query Execution Plans

Three events can be used to collect query execution plans as part of an Extended Events session:

- **query_post_compilation_showplan**. Returns the estimated query execution plan when a query is compiled.
- **query_pre_execution_showplan**. Returns the estimated query execution plan when a query is executed.
- **query_post_execution_showplan**. Returns the actual query execution plan when a query is executed.

When using any of these events you should consider that adding them to a session, even when predicates are used to limit the events captured, can have a significant impact on the performance of the database engine instance. This effect is most marked with the **query_post_execution_showplan** event. You should limit your use of these events to troubleshooting specific issues; they should not be included in an Extended Events session that is always running.

Understand the Deadlock Graph Format

Deadlock graphs collected by the **xml_deadlock_report** and **database_xml_deadlock_report** events are in a different format from the deadlock graphs produced by SQL Server Profiler; with these, you can use deadlock graphs captured by Extended Events to represent complex deadlock scenarios involving more than two processes. If saved as an .xdl file, both formats of deadlock graph can be opened by SSMS.

Demonstration: Tracking Session-Level Waits

In this demonstration, you will see how to report on wait types by session using Extended Events.

Demonstration Steps

1. In SSMS, in Solution Explorer, double-click **Demo 2 - track waits by session.sql**.
2. In Object Explorer, expand **Management**, expand **Extended Events**, right-click **Sessions**, and then click **New Session**.
3. In the **New Session** dialog box, on the **General** page, in the **Session name** box, type **Waits by Session**.
4. On the **Events** page, in the **Event library** box, type **wait**, and then, in the list below, double-click **wait_info** to add it to the **Selected events** list.
5. Click **Configure** to display the **Event configuration options** list.
6. In the **Event configuration options** list, on the **Global Fields (Actions)** tab, select the **session_id** check box.

7. On the **Filter (Predicate)** tab, click **Click here to add a clause**. In the **Field** list, click **sqlserver.session_id**, in the **Operator** list, click **>**, and then in the **Value** box, type **50**. This filter will exclude most system sessions from the session.
8. On the **Data Storage** page, click **Click here to add a target**. In the **Type** list, click **event_file**, in the **File name on server** box, type **D:\Demofiles\Mod09\waitbysession**, in the first **Maximum file size** box, type **5**, in the second **Maximum file size** box, click **MB**, and then click **OK**.
9. In Object Explorer, expand **Sessions**, right-click **Waits by Session**, and then click **Start Session**.
10. In File Explorer, in the **D:\Demofiles\Mod09** folder, right-click **start_load_1.ps1**, and then click **Run with PowerShell**. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press **ENTER**. Leave the workload to run for a minute or so before proceeding.
11. In SSMS, in the Demo 2 - track waits by session.sql pane, select the code under the comment that begins **--Step 14**, click **Execute**, and then review the results.
12. Select the code under the comment that begins **-- Step 15**, and then click **Execute** to stop and drop the session, and to stop the workload.
13. In File Explorer, in the **D:\Demofiles\Mod09** folder, note that one (or more) files with a name matching **waitbysession*.xel** have been created.
14. Close File Explorer, close SSMS without saving changes, and then in the Windows PowerShell® window, press **ENTER** twice to close the window.

Categorize Activity

Place each Extended Events target type into the appropriate category. Indicate your answer by writing the category number to the right of each item.

Items	
1	Ring buffer target
2	Event file target
3	Histogram target
4	Event tracking for Windows target
5	Event pairing target
6	Event counter target

Category 1		Category 2
Written to Memory Buffers		Written to File on Disk

Lab: Extended Events

Scenario

While investigating why the new SQL Server instance was so slow, you came across deadlock occurrences and excessive fragmentation in indexes caused by page split operations. In this lab, you will review deadlock occurrences using the default session and implement a new Extended Event session to identify workloads that cause huge page splits.

Objectives

After completing this lab, you will be able to:

- Access data captured by the System Health Extended Events session.
- Create a custom Extended Events session.

Estimated Time: 90 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Using the system_health Extended Events Session

Scenario

While investigating why the new SQL Server instance was so slow, you were informed that users frequently report deadlock error messages in application logs. In this exercise, you will review Extended Events default system_health session and analyze the output of a deadlock event.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Run a Workload

3. Query the system_health Extended Events Session

4. Extract Deadlock Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

2. Run **Setup.cmd** in the **D:\Labfiles\Lab09\Starter folder** as Administrator.

▶ Task 2: Run a Workload

1. In the **D:\Labfiles\Lab09\Starter** folder, run **start_load_1.ps1** with Windows PowerShell. If a message is displayed asking you to confirm a change in execution policy, type **Y**.

2. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the system_health Extended Events Session

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** instance, then **D:\Labfiles\Lab09\Starter\Project\Project.ssmssln** and **Exercise 01 - system_health.sql**.
2. Under the comment that begins **-- Task 2**, edit and execute the query to return data from the system_health session, using the **sys.fn_xe_file_target_read_file** DMF to extract data from the session's event file target.
3. Hint: you can examine the definition of the system_health session to find the file name used by the event file target.

▶ Task 4: Extract Deadlock Data

1. Under the comment that begins **-- Task 3**, edit and execute the query to extract the events with a name attribute of **xml_deadlock_report** from the XML version of the **event_data** column. Include the event time and the **/event/data/value/deadlock** element in your output.
2. Click on any of the row values in the **deadlock_data** column to view the deadlock XML in detail.

Results: After completing this exercise, you will have extracted deadlock data from the SQL Server.

Exercise 2: Tracking Page Splits Using Extended Events

Scenario

While investigating why the new SQL Server instance was so slow, you came across excessive fragmentation in indexes caused by page split operations. In this exercise, you will implement Extended Events to identify those workloads that cause huge page splits.

 **Note:** Although a **page_split** event is available, it doesn't provide enough information for you to discriminate between expected page splits (which occur when a table or index is extended, referred to as *end page splits*) and page splits which can harm performance (which occur when data must be inserted in the middle of a page, referred to as *mid-page splits*). You can detect mid-page splits by analyzing the **transaction_log** event.

The main tasks for this exercise are as follows:

1. Create an Extended Events Session to Track Page Splits
2. Run a Workload
3. Query the Session
4. Extract alloc_unit_id and Count Values
5. Return Object Names
6. Delete the Session

▶ Task 1: Create an Extended Events Session to Track Page Splits

1. In Solution Explorer, open **Exercise 02 – page splits.sql**.
2. Create a new Extended Events session on the **MIA-SQL** instance with the following properties:
   - Session name: **track page splits**
   - Event(s) included: **sqlserver.transaction_log**

- Event filter(s):
  - **operation** = **LOP_DELETE_SPLIT**
  - **database_name** = **AdventureWorks**
- Session target: **Histogram**
  - Filtering target: **sqlserver.transaction_log**
  - Source: **alloc_unit_id**
  - Source type: **event**

▶ Task 2: Run a Workload

1. In the **D:\Labfiles\Lab09\Starter** folder, execute **start_load_2.ps1** with PowerShell.
2. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the Session

- In SSMS, in the query window for **Exercise 02 – page splits.sql,** under the comment that begins **-- Task 3**, edit and execute the query to extract data from the histogram target of the **track page splits** session. Use the **sys.dm_xe_session_targets** DMV to extract data from the session's histogram target. For this task, include only the **target_data** column in your output result set and cast the results to XML.

▶ Task 4: Extract alloc_unit_id and Count Values

- Under the comment that begins **-- Task 4**, edit and execute the query so that it returns the **count** attribute for each **HistogramTarget/Slot** node, and the **value** child node for each **HistogramTarget/Slot** node.

▶ Task 5: Return Object Names

- Under the comment that begins **-- Task 5**, edit and execute the query to join to **sys.allocation_units**, **sys.partitions** and **sys.indexes** to find the names of objects affected by page splits.

▶ Task 6: Delete the Session

1. Delete the **track page splits** session.
2. Close any open applications and windows.

Results: After completing this exercise, you will have extracted page split data from SQL Server.

Question: If an Extended Events session has no targets defined, how would you view the data generated by the session?

Module Review and Takeaways

In this module, you have learned about the Extended Events system and its components. You have learned about the objects that comprise an Extended Events session, and how to query the metadata about each object type to understand the information it provides.

You have learned how to create, amend and drop Extended Events sessions using either SSMS or Transact-SQL, in addition to learning about various methods for extracting data from session targets.

Review Question(s)

Check Your Knowledge

Question	
Which of the following sources does not contain detailed information about Extended Events event definitions?	
Select the correct answer.	
	SQL Server Management Studio Extended Events GUI.
	The DMV sys.dm_xe_objects.
	Microsoft Docs.

Module 10

Monitoring, Tracing, and Baselines

Contents:

Module Overview

This module describes tools and techniques you can use to monitor and trace Microsoft® SQL Server® performance data, and to create baselines to assess future changes in performance. It focuses on data collection strategy and techniques to analyze the collected data.

Objectives

After completing this module, you will be able to:

- Monitor and trace SQL Server performance data.
- Create baselines and benchmarks for SQL Server performance.

Lesson 1
Monitoring and Tracing

This lesson describes the tools that you can use to monitor and trace SQL Server diagnostic data. Monitoring SQL Server enables you to detect performance problems as and when they happen or before they happen, so you can resolve issues proactively. You can also use the trace and monitoring data to establish a baseline and create a benchmark for SQL Server performance.

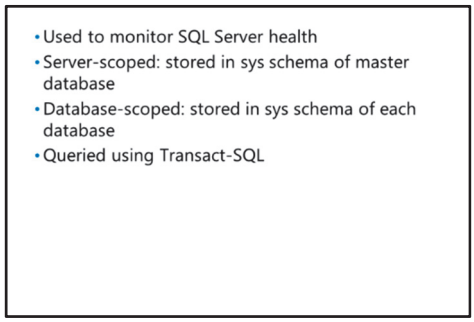
Lesson Objectives

After completing this lesson, you will be able to:

- Use dynamic management objects (DMOs) to monitor SQL Server performance.
- Use Windows® Performance Monitor to monitor SQL Server performance.
- Use SQL Server Activity Monitor to monitor SQL Server Performance.
- Use Extended Events to trace SQL Server events.
- Use SQL Server Profiler to trace SQL Server events.
- Use SQL Trace to trace SQL Server events.
- Describe the default trace.
- Analyze the trace data.
- Replay the trace data.

Dynamic Management Objects

DMOs are a collection of dynamic management views (DMVs) and dynamic management functions (DMFs) that you can use to monitor the health of a SQL Server instance or database.



You can query a DMV by using Transact-SQL in the same way that you can query other views. A DMF is a table-valued function that accepts one or more parameters.

Server-scoped DMOs are stored in the sys schema of the master database and provide information at the instance level. Querying a server-scoped DMO requires the VIEW SERVER STATE permission and the SELECT permission on the appropriate object.

Database-scoped DMOs are stored in the sys schema of each database and provide information at the database level. Querying a database-scoped DMO requires the VIEW DATABASE STATE permission and the SELECT permission on the appropriate object.

There are more than 150 DMOs covering many categories; they are an important and useful tool that you can use to monitor and tune SQL Server. You can identify the category of a DMV by its prefix, as shown in the following table:

Prefix	Category
dm_db_	Database and index
dm_exec_	Query
dm_io_	Disk subsystem
dm_os_	Hardware usage

The following DMVs are particularly valuable for gathering performance information:

sys.dm_db_index_physical_stats. This provides size and fragmentation information for all indexes.

sys.dm_exec_rquests. This provides information about currently running queries including start time, resource utilization, and estimated completion time. You can use it to identify blocking issues.

sys.dm_io_virtual_file_stats. This returns information about database file reads and writes. You can use it to identify contention issues.

sys_dm_os_wait_stats. This returns the total wait time for each wait type since instance startup or when the wait statistics were last cleared. You can clear wait statistics by using the **DBCC SQLPERF(WAITSTATS, CLEAR)** command.

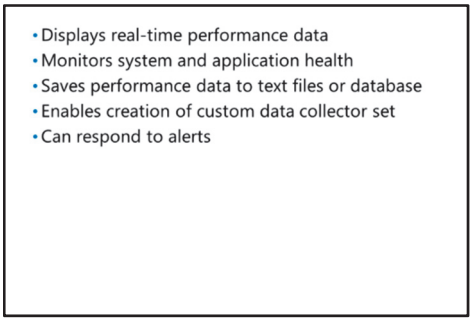
For more information about DMVs, see the topic *Dynamic Management Views and Functions (Transact-SQL)* in Microsoft Docs:

Dynamic Management Views and Functions (Transact-SQL)

http://aka.ms/Yds84a

Windows Performance Monitor

Windows Performance Monitor is a graphical tool for monitoring system performance. It is preinstalled with Windows and provides insight into current application and hardware performance by using built-in Windows performance counters. You can use Performance Monitor to:



- Display real-time system performance data in three formats: line graph, histogram, and report.
- Monitor overall system health, SQL Server health, or the health of other applications by selecting counters from the available performance objects.
- Record current performance counter values in text files and databases to analyze later. You can analyze the performance data by using file manipulation techniques or Transact-SQL queries against the appropriate database.

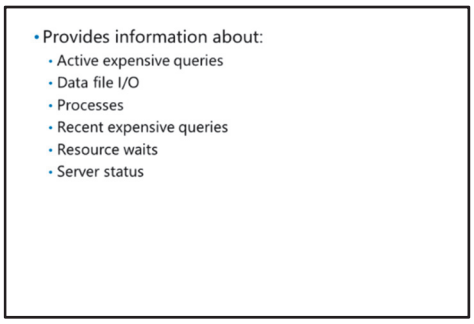
- Create custom sets of performance counters known as a data collector set that you can then schedule to run as appropriate.
- Configure and respond to alerts. For example, when a specified threshold is reached, start a particular data collector set or a particular program.

You can use Performance Monitor for real-time monitoring and to establish a baseline for SQL Server performance. You can collect performance data over time and analyze it to calculate workload characteristics such as peak and off-peak hours, average CPU usage, memory usage, and more. It is often useful to demonstrate performance gains or losses following a system change; by using Performance Monitor you can do so easily by comparing counter values before and after implementation.

Performance Monitor is very lightweight and therefore has minimal performance overhead for sampling intervals of greater than one second. The default or optimal sampling interval is 15 seconds. The amount of I/O that Performance Monitor generates will depend on the number of counters, the sampling interval, and the underlying storage. If there is an I/O issue, consider saving the performance data on separate storage and only enable the counters that you need.

Activity Monitor

Activity Monitor is a lightweight monitoring tool that is built into SQL Server Management Studio. Activity Monitor displays the current SQL Server processes and their effect on the instance. Activity Monitor provides information about the following areas:



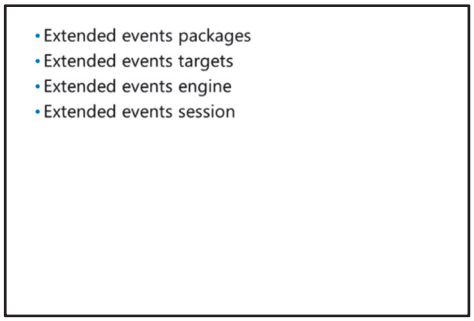
- **Active expensive queries**. This shows running queries with high resource usage in terms of CPU, I/O, and memory.
- **Data file I/O**. This displays usage information for the database files.
- **Processes**. This shows diagnostic information about running processes in SQL Server in a tabular format that is easy to analyze.
- **Recent expensive queries**. This shows recent queries with high resource usage in terms of CPU, I/O, and memory.
- **Resource waits**. This displays real-time wait statistics.
- **Overview**. This provides an overview of the current health of the system.

SQL Server Activity Monitor presents information in a document window that consists of a number of collapsible panes. You can customize these by rearranging or sorting columns, or by adding filters.

When you expand a tab in Activity Monitor, queries are sent to the database engine to retrieve the required data. To ensure that unnecessary load is not placed on the server, querying stops when you collapse a tab.

To open Activity Monitor from within SQL Server Management Studio, in Object Explorer, right-click a database instance, and then click **Activity Monitor**. You cannot export or record data from Activity Monitor.

Extended Events



Extended Events is a lightweight, highly scalable, and configurable performance monitoring system. You can configure Extended Events by using Transact-SQL or a graphical interface. Extended Events, first provided in SQL Server 2008, is based on the following concepts:

- **Extended Events packages**. There are three types of Extended Events packages, which contain the following Extended Events objects:
  - **package0**. This is the default package that contains Extended Events system objects.
  - **Sqlserver**. This contains objects that are related to SQL Server.
  - **Sqlos**. This contains objects that are related to the operating system.

A package can contain the following objects:

  - Events
  - Targets
  - Actions
  - Types
  - Predicates
  - Maps

- **Extended Events targets**. These event consumers receive data during an event session. The following targets are available:
  - Event counter
  - Event file
  - Event pairing
  - Event Tracing for Windows (ETW)
  - Histogram
  - Ring buffer
- **Extended Events engine**. This manages and implements the Extended Events session.
- **Extended Events session**. This describes the Extended Events session. A session can have one of the following states:
  - **Create**. The session is created by executing the CREATE EVENT SESSION command. The session definition and permission level are checked, and the metadata is stored in the master database. The session is not active at this point.
  - **Start**. The session is started by executing the ALTER EVENT SESSION STATE = START command. The process reads the metadata, validates the session definition and permissions, loads the session objects such as events and targets, and activates the event handling.

- **Stop**. The session is stopped by executing the ALTER EVENT SESSION STATE = STOP command. The session metadata is retained.
- **Drop**. The session is closed if active, and metadata is deleted by executing the DROP EVENT SESSION command.

When to Use Extended Events

Extended Events replaces SQL Server Profiler and SQL Trace. You can therefore use it to perform regular monitoring, such as the following:

- Troubleshoot blocking and deadlocks.
- Find resource-intensive queries.
- Capture dynamic-link library (DDL) statements.
- Capture missing column statistics.
- Capture hash and sort warnings.
- Capture wait statistics.

You can also use Extended Events to collect information that you were not able to monitor with SQL Trace. You can now:

- Observe page splits.
- Monitor session-level wait statistics.
- Monitor stored procedures that exceed the last CPU, I/O, and execution times.
- Observe the proportional fill algorithm that is used to fill data in data files.

Extended Events is easy to set up and provides a more lightweight monitoring system than SQL Server Profiler.

The following code example captures blocking occurrences:

Extended Events Session

```
CREATE EVENT SESSION [Monitor_blocking] ON SERVER
ADD EVENT sqlserver.blocked_process_report(
    ACTION(sqlserver.client_app_name,
           sqlserver.client_hostname,
           sqlserver.database_name))
ADD TARGET package0.asynchronous_file_target
(SET filename = N'c:\blocked_process.xel',
     metadatafile = N'c:\blocked_process.xem',
     max_file_size=(65536),
     max_rollover_files=2)
WITH (MAX_DISPATCH_LATENCY = 5SECONDS)
GO
```

The Monitor_blocking session writes the blocked process report to the file target every five seconds. To start monitoring, execute the following query:

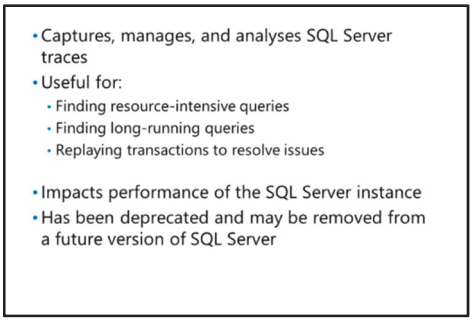
ALTER EVENT SESSION [Monitor_blocking] ON SERVER STATE=START

To stop the Extended Events session, execute the following query:

ALTER EVENT SESSION [Monitor_blocking] ON SERVER STATE=STOP

SQL Server Profiler

SQL Server Profiler is an application for capturing, managing, and analyzing traces. The trace can be saved to a trace file or a database table. SQL Server Profiler has been deprecated and may be removed in future versions of SQL Server.



You can use SQL Server Profiler to capture relevant data points for selected events that you can use to identify the causes of performance problems. You must understand the following terms before you work with SQL Server Profiler:

- **Event**. An event is an action that occurs within a SQL Server instance. For example, the start and end of a SQL batch, the start and end of a stored procedure, login connections, failures, and disconnections.
- **Event class**. An event class is all of the event data that can be traced. For example, SQL:BatchCompleted, Audit Login, and Audit Logout.
- **Event category**. Events are grouped into event categories based on their relevance. For example, lock events are grouped into a lock event category.
- **Data column**. A data column is an element of an event class. For example, SQL:BatchCompleted has a data column duration that captures the execution time of the Transact-SQL batch that has completed.
- **Trace**. A trace records the data based on the events that are selected in SQL Server Profiler. For example, to trace audit logout, select the audit logout event under the Security event class.
- **Filter**. Data in a trace can be filtered on data columns. For example, to find queries that are taking longer than 10 seconds, place a filter on the duration data column.

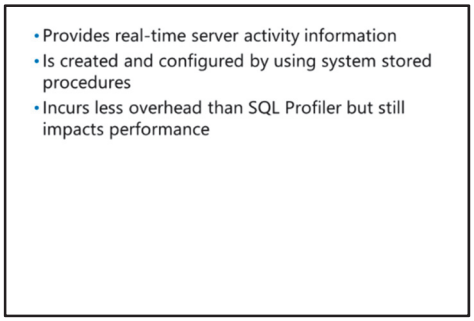
You can use SQL Server Profiler to:

- Find resource-intensive queries.
- Find long-running queries.
- Resolve problems by replaying a recorded trace in a test environment.
- Correlate Performance Monitor output to diagnose problems.
- Capture blocking and deadlock graphs.
- Find missing statistics, and hash and sort warnings.

SQL Server Profiler is a good tool to diagnose and fix performance problems. However, before you use it, you should be aware that SQL Server Profiler has a negative impact on the performance of a SQL Server instance, particularly if you run more than one trace at a time. Events in SQL Server Profiler generate all data columns, even if you have not selected all columns for recording.

SQL Trace

SQL Trace, or server-side trace, provides real-time insights into the activity of SQL Server. You can use it to:



- Understand the duration, frequency, and resource usage of queries.
- Gather a baseline or create a benchmark for a system's activity.
- Troubleshoot application errors and performance problems.
- Audit user activity.

SQL Trace works as follows:

1. The trace controller inside the database engine maintains a bitmap of events that are being collected by an active trace.
2. The event provider checks whether its event is active in the bitmap. If it is, then it provides a copy of the event data to the trace controller.
3. The trace controller queues the event data and provides the event data to all active traces that are collecting the event.
4. The individual traces filter the event data, removing any columns that are not required and discarding events that do not match trace filters.
5. The remaining event data is written to a file locally on the server or buffered to the row-set provider for consumption by external applications, such as SMO and SQL Server Profiler.

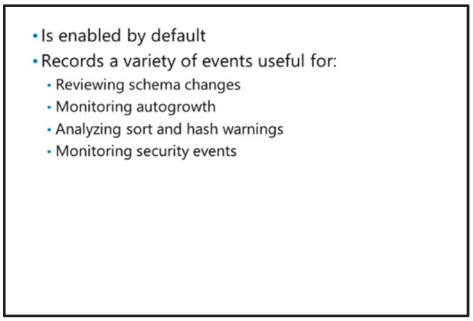
SQL Trace is created by using the following system stored procedures:

- **sp_trace_create**. This creates a trace with the provided configuration and returns the trace_id for the new trace.
- **sp_trace_setevent**. This adds or removes an event or column to an existing trace.
- **sp_trace_setfilter**. This applies a filter to an existing trace.
- **sp_trace_setstatus**. This modifies the status of a trace (0-stop, 1-start, 2-delete).

SQL Trace incurs less overhead than SQL Server Profiler. However, it does negatively affect the performance of the instance that you trace. Some of the best practices to follow when you work with SQL Trace include:

- Use SQL Trace to establish baselines and create benchmarks only when there is not an alternate way to analyze a problem.
- Trace only relevant events and the data columns with appropriate filters, because collecting all data columns and events is more likely to cause performance issues.

The Default Trace



The default trace is a server-side trace that comes as default with SQL Server. The default trace is enabled by default, and you can disable it by using the **sp_configure** system stored procedure. The following query checks the status of the default trace:

SELECT * FROM sys.configurations WHERE name = 'default trace enabled'

Default trace details are obtained by querying the **sys.traces** system view, as shown in the following code:

SELECT * FROM sys.traces WHERE is_default=1

The default trace is a lightweight trace that captures the following database and server events:

- Database:
  - Data file auto grow
  - Data file auto shrink
  - Database mirroring status change
  - Log file auto grow
  - Log file auto shrink
- Errors and Warnings:
  - Errorlog
  - Hash warning
  - Missing Column Statistics
  - Missing Join Predicate
  - Sort Warning
- Full-Text:
  - FT Crawl Aborted
  - FT Crawl Started
  - FT Crawl Stopped
- Objects:
  - Object Altered
  - Object Created
  - Object Deleted

- Security Audit:
  - Audit Add DB user event
  - Audit Add login to server role event
  - Audit Add Member to DB role event
  - Audit Add Role event
  - Audit Add login event
  - Audit Backup/Restore event
  - Audit Change Database owner
  - Audit DBCC event
  - Audit Database Scope GDR event (Grant, Deny, Revoke)
  - Audit Login Change Property event
  - Audit Login Failed
  - Audit Login GDR event
  - Audit Schema Object GDR event
  - Audit Schema Object Take Ownership
  - Audit Server Starts and Stops
- Server:
  - Server Memory Change

The default trace can be useful when you want to retrieve schema changes history, monitor auto growth to proactively avoid performance issues, optimize query performance by analyzing sort warning, hash warning, missing column statistics and missing join predicate, and monitor security events, server start and stop, failed logins.

You can query the default trace by using Transact-SQL.

Querying the Default Trace

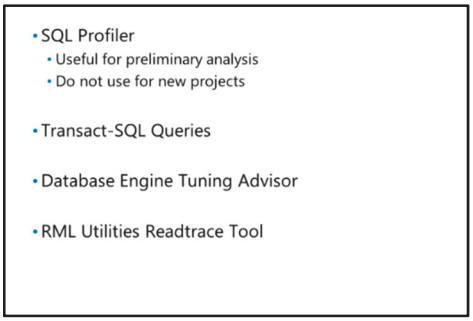
```
SELECT *
FROM ::fn_trace_gettable('C:\Program Files\Microsoft SQL
Server\MSSQL.13\MSSQL\LOG\log.trc',0)
INNER JOIN sys.trace_events e
ON eventclass = trace_event_id
```

The path of the default trace file might be different. The preceding query selects all data columns from the default trace.

Analyzing Trace Data

The trace data that you can collect by using SQL Trace or SQL Server Profiler is only useful if you analyze it. SQL Trace provides comprehensive information about the SQL Server's workload. You can use the following methods to analyze SQL Trace data:

- SQL Server Profiler.
- Transact-SQL queries.
- The Database Engine Tuning Advisor.
- Replay Markup Language (RML) Utilities Readtrace Tool.



SQL Server Profiler

SQL Server Profiler is useful for preliminary analysis; however, where possible, you should use other methods to analyze a SQL Trace. You can filter trace data on events or data columns to isolate problematic queries. You can group trace data into data columns to get an overview of the trace. You can save filtered trace data to a table to perform further analysis by using Transact-SQL.

Note: SQL Server Profiler has been marked for deprecation since SQL Server 2012. Microsoft intends to remove the tool in a future version of SQL Server. Extended Events is now the activity tracing tool that Microsoft recommends.

Filter Trace Data

To filter trace data, follow these steps:

1. In SQL Server Profiler, open the saved trace (trace file or table).
2. On the **File** menu, click **Properties**. This brings up the **Trace File** properties window.
3. Select or clear events to filter events; similarly, select or clear data columns to filter data columns.
4. To filter data column values, at the bottom of the **Trace File** properties window, click **Column Filters**. This opens the **Edit Filter** dialog box.
5. In the **Edit Filter** dialog box, in the leftmost pane, select a data column; in the rightmost pane, expand either **Like** or **Not like**, and then create the appropriate filter.

Group Trace Data

You can group trace data based on data columns for analysis. For example, you can use a group based on the **CPU** column to help to isolate CPU-intensive queries. Similarly, you can use a group based on the **Duration** column to help to isolate long-running queries. To group trace data, follow these steps:

1. In SQL Server Profiler, open the trace.
2. On the **File** menu, click **Properties**. The **Trace File** properties dialog box appears.
3. In the **Trace File** Properties dialog box, click **Organize Columns**.
4. In the **Organize Columns** dialog box, to group on a particular column, click the **Up** button to move the column to the **Groups** section. To ungroup, move it to the **Columns** section.
5. Click **OK** to show the grouped trace data.

Grouping can help you to identify Transact-SQL queries that require optimization.

Transact-SQL Queries

Trace data that is saved in a file or table can be queried by using Transact-SQL statements. This means you can use the full power of the SQL language to analyze trace data.

To query a trace data file, you use the **sys.fn_trace_gettable** function. The function takes the following two parameters:

- **Filename**. This is the fully qualified path of the trace file from which data is to be read. This is a mandatory parameter.
- **Rollover files**. This is the number of files to process in case of multiple trace files. The default is to process all files.

The following Transact-SQL query demonstrates how you would read all **SQL:BatchCompleted** events from the trace file that is stored in C:\tracefile.trc.

Reading Events from a Trace File

```
SELECT        TextData,
              SPID,
              Duration,
              Reads,
              Writes,
              CPU
FROM sys.fn_trace_gettable ('C:\tracefile.trc', 1)
WHERE EventClass = 12
```

The following Transact-SQL query demonstrates the aggregation of data from the same trace file. The result will be a list of Transact-SQL statements with their total execution time, reads, writes, and CPU time. You can use this information to identify long-running, CPU-intensive, or I/O-intensive queries.

Aggregating Data

```
SELECT
              CAST(TextData AS NVARCHAR(MAX)) AS TextData,
              SUM(Duration) AS Duration,
              SUM(Reads) AS Reads,
              SUM(Writes) AS Writes ,
              SUM(CPU) AS CPU
FROM sys.fn_trace_gettable('C:\tracefile.trc',1)
WHERE EventClass = 12
GROUP BY CAST(TextData AS NVARCHAR(MAX))
ORDER BY CPU DESC
```

You can also use Transact-SQL statements to get security audit, blocking, and deadlock details and warnings such as sort warnings, hash warnings, and missing column statistics from trace data.

Database Engine Tuning Advisor

The Database Engine Tuning Advisor is a graphical and command-line tool that analyzes how the SQL Server Database Engine processes queries and makes recommendations for improving the performance of queries through the creation of indexes, partitions, indexed views, and statistics. It is provided with SQL Server and can consume a trace file, analyze the queries that are captured, and then provide details of:

- New indexes and statistics that can increase query performance.
- Suggest partition strategy.
- The usage of existing indexes.

To analyze a trace file by using Database Engine Tuning Advisor, follow these steps:

1. Open the Database Engine Tuning Advisor:
    a. In SQL Server Management Studio, on the **Tools** menu, click **Database Engine Tuning Advisor**.
    b. On the **Start** screen, open **SQL Server Database Engine Tuning Advisor**.
2. On startup, a new session is created in the leftmost pane. In the rightmost pane, on the **General** tab, perform the following steps:
    a. Change the session name, if required.
    b. In the **Workload** section, select the trace file.
    c. Select the database for workload analysis.
    d. Select the database and tables to tune.
3. On the **Tuning Options** tab, select the type of analysis to perform.
4. Click **Start Analysis** to start the analysis.
5. When the analysis is complete, the Database Engine Tuning Advisor provides recommendations for the creation of indexes and statistics that are useful for tuning. It also produces reports that provide valuable insights into the workload.

 **Note:** The recommendations that the Database Engine Tuning Advisor provides are based on the provided workload. To obtain the maximum benefit from the Database Engine Tuning Advisor, ensure that the workload you submit closely represents the entire application workload.

Always check any recommendations from the Database Engine Tuning Advisor before you implement them, because some suggestions may be very similar and you can combine them for efficiency. When you implement indexes, remember that although an index may speed up some queries, it will adversely affect others; this is particularly relevant when dealing with transactional systems.

RML Utilities Readtrace Tool

RML utilities are free SQL Server tools that are provided by Microsoft. RML utilities are not installed with SQL Server, but you can download them.

 **RML Utilities for SQL Server (x64) CU4**

http://aka.ms/m65uz0

The SQL Server support team uses RML utilities to diagnose and resolve customer issues, but database administrators can use the Readtrace tool, included with RML utilities, to analyze a SQL Trace file. To analyze trace files by using Readtrace, install the RML utilities and then execute the following command in the Command Prompt window:

```
Readtrace –IC:\tracedemo.trc
```

The trace file path may be different in your case. The Readtrace tool will process the trace and will launch Reporter.exe, which is an external application. The Reporter application contains a set of SQL Server reports that give insights into the trace data and provide a cumulative resource usage chart.

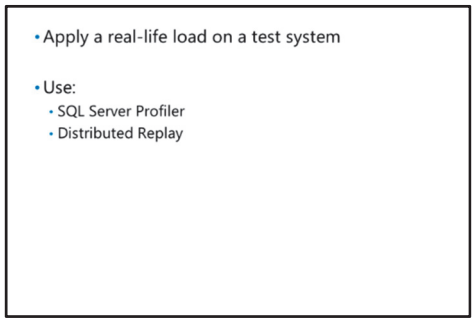
The cumulative resource usage chart provides time-based aggregated analysis that you can investigate. Other than the cumulative resource usage chart, Readtrace displays resource consumption based on the following parameters:

- Application name
- Unique batches
- Databases
- Unique statements
- Login name

Readtrace also provides reports on interesting events and data lineage.

Trace Replay

Trace Replay helps you to assess the impact of hardware and software changes by putting a real-life load on a test system. You can perform a Trace Replay by using SQL Server Profiler or the Microsoft SQL Server Distributed Replay feature.



Note: Although SQL Server Profiler is still available and is provided with Microsoft SQL Server, try to use alternative tools where possible, because SQL Server Profiler is deprecated and may be removed in a future version of SQL Server.

You can use Trace Replay to replay a production workload on a test server to help to identify performance issues or to test the impact of software patches, bug fixes, or configuration changes. For example, you can replay the trace to check the effect of the newly created indexes on the test server.

Note: Trace Replay replays transactions in the order in which they occurred but not with the same timings. For example, if a captured trace file contained two queries that were issued 10 minutes apart, they would be replayed sequentially without the 10-minute gap.

SQL Server Profiler

SQL Server Profiler is a wrapper over the SQL Trace commands and provides a graphical interface to start and stop traces, choose events to trace, and apply trace filters. You can also use SQL Server Profiler to play back the trace events from a trace in the sequence in which they occurred.

SQL Server Profiler provides a T-SQL_Replay template that is used to capture data for trace replay. To replay a trace, start the trace by using the TSQL_Replay template and save the trace data. To set up trace replay, follow these steps:

1. Open SQL Server Profiler, and then start a new trace with the TSQL_Replay trace template.
2. Record the trace. The EventSequence column is used to track the order in which the events occur. The EventSequence value is incremented globally as traces are recorded. This is used when events are replayed in the order in which they occur.
3. After the data collection is complete, save and close the trace window.
4. Open the saved trace. On the top menu, a replay option appears. Click Replay, and then click Start.
5. A connection window appears. You can replay the trace on the server from which it was captured or on a different server. Connect to the relevant server.
6. A replay configuration option window appears. You can save the replay results to a file or a table. Select the desired option, and then click OK to continue.
7. The trace is replayed.

SQL Server Distributed Replay

SQL Server Distributed Replay is a more scalable solution than SQL Server Profiler and provides increased functionality, including the ability to replay trace data from multiple servers on a single SQL Server instance. SQL Server Distributed Replay is particularly suitable when the concurrency in a trace file is high and replaying it through SQL Server Profiler causes bottlenecks. SQL Server Distributed Replay is also useful for stress testing.

A Distributed Replay environment consists of the following components:

- **Distributed Replay administration tool (DReplay.exe)**. This is an application that is used to control the distributed replay. DReplay.exe is console-based.
- **Distributed Replay clients**. This is one or more computers that combine to replay transactions from the trace file on the target SQL Server instance.
- **Distributed Replay controller**. This manages the activities of the Distributed Replay clients. Only one Distributed Replay controller is active within a Distributed Replay environment.
- **Target SQL Server instance**. This is the SQL Server instance that accepts the transactions from the Distributed Replay clients. This is normally a test server.

You can install all components on the same computer or on different computers—either physical or virtual.

To replay a trace by using SQL Server Distributed Replay, follow these steps:

1. **Configure preprocess configuration settings**. Preprocess configuration settings are modified in the xml file, DReplay.exe.preprocess.config, which is located in the administration tool install folder.
2. **Preprocess trace data**. Before a trace file can be used with Distributed Replay, it must be preprocessed. This is done with the Distributed Replay administration tool (DReplay.exe).

3. **Configure replay settings**. Replay configuration settings are modified in the xml file DReplay.exe.replay.config located in the administration tool install folder.
4. **Start the replay**. Start the replay using the Distributed Replay administration tool (DReplay.exe) specifying the options required at the command prompt.

For more information about Distributed Replay, see the topic *SQL Server Distributed Replay* in Microsoft Docs:

SQL Server Distributed Replay

http://aka.ms/Bgvihr

Demonstration: Analyzing Performance Monitor Data

In this demonstration, you will see how to analyze Performance Monitor data.

Demonstration Steps

1. Ensure the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Demofiles\Mod10** folder, right-click **Setup.cmd**, and then click **Run as Administrator**.
3. In the **User Account Control** dialog box, click **Yes**, wait for the script to finish, and then press Enter.
4. Start **SQL Server Management Studio**, and then connect to the **MIA-SQL** database engine instance by using Windows authentication.
5. In SQL Server Management Studio, in the **D:\Demofiles\Mod10\Demo01** folder, open the **CollectPerfMonData.sql** file, and then execute the script to create the **CollectPerfMonData** job.
6. In Windows Explorer, navigate to the **D:\Demofiles\Mod10\Demo01** folder, right-click **RunWorkload.cmd** and then click **Run as Administrator**.
7. In the **User Account Control** dialog box, click **Yes**, and wait for the script to finish.
8. In SQL Server Management Studio, in the **D:\Demofiles\Mod10\Demo01** folder, open the **AnalysisQueries.sql** file.
9. Select the code under **Step 1 - CPU Usage Trend**, click **Execute**, and then observe the **CPU usage %** counter value that is collected during the workload execution.
10. Select the code under **Step 2 - Memory usage trend**, click **Execute**, and then observe how the **Total Server Memory (KB)** increases over time, and then remains consistent.
11. Select the code under **Step 3 - Transaction throughput**, click **Execute**, and then observe the trend in the **Transactions/sec** counter value.
12. Select the code under **Step 4 - Transaction reads per second**, click **Execute**, and then observe the trend in the **Page reads/sec** counter value.
13. Select the code under **Step 5 - Cleanup job and database**, and then click **Execute** to clean up the database and agent job.
14. Close SQL Server Management Studio without saving any changes.

Check Your Knowledge

Question	
Which tool or feature might you use instead of SQL Server Profiler to find resource-intensive queries on a SQL Server instance?	
Select the correct answer.	
	SQL Trace
	Performance Monitor
	Extended Events
	SQL Server Agent

Lesson 2
Baselining and Benchmarking

This lesson focuses on how to establish baselines and create benchmarks for SQL Server. You must establish a SQL Server baseline because baselines give insights into trends that occur in a SQL Server environment. This simplifies troubleshooting.

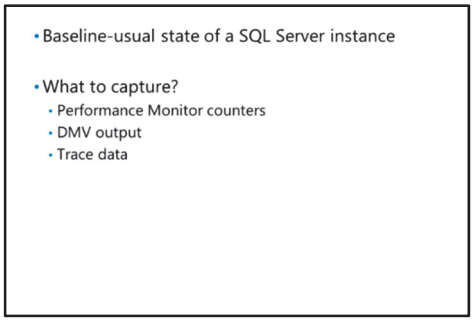
Lesson Objectives

After completing this lesson, you will be able to:

- Understand the methodology for creating baselines.
- Use stress-testing tools.
- Collect data by using DMVs.
- Collect data by using Performance Monitor.
- Analyze collected data.
- Use the Database Engine Tuning Advisor.

Methodology for Creating Baselines

A baseline is the normal or usual state of a SQL Server environment. A SQL Server baseline informs you how a system typically performs on any given day. Some of the main benefits of a baseline include the following:



- It provides insights into trends that occur in an environment, in addition to data.
- You can compare baselines with current system state to find out what has changed and proactively tune or resolve problems.
- It provides better capacity planning.
- It makes troubleshooting swift and easy.

SQL Server diagnostic data is captured over time and averaged to get a baseline. In a SQL Server environment, the resource usage may vary from time to time. For example, resource usage can be high during business peak hours and can be lower during off-peak hours. It can be high during weekends when weekly reports are generated. Therefore, multiple baselines may be needed for each of these situations.

A benchmark is a standard point of reference against which subsequent metrics can be compared. It is different from a baseline. For example, the benchmark for a stored procedure execution time can be 2 seconds. The baseline will give the usual or normal execution time of a given stored procedure. If it is up to the baseline then it is considered good; otherwise, the stored procedure needs to be optimized.

Although it is important for you to establish a baseline for SQL Server, it is not easy. Merely collecting data will not help. The sole purpose of establishing a baseline is to get the benefits it offers. Establishing a baseline for a SQL instance involves more than just writing the scripts to collect data. Before you write scripts, you should know what to capture, the frequency, and the storage. These topics are discussed in detail in the following sections.

What to Capture?

Many diagnostic data points are available to be collected from one or more sources. Too much data can be overwhelming and difficult to analyze. It will only add to the storage cost instead of being useful. Capture relevant data points—the data that can help you diagnose and foresee performance problems. To start with, consider collecting the following data points:

- **System usage**. System usage is largely described in terms of CPU, I/O, and memory consumption. You can quickly check these basic Performance Monitor counters against the current values in case of sudden performance degradation. These values can also be used to define a system usage trend, which will further help in capacity planning.
- **SQL Server configuration**. These are instance-level or database-level configuration settings, such as max server memory, degree of parallelism, or auto shrink. You can change these configuration settings, but changing them without having advanced knowledge of SQL Server can create problems.
- **Database size information**. The system will come to a halt when storage runs out of space. This makes it necessary to capture database and file size information. It will help you to ensure that the system is up and running and proactively reacts to space issues, thereby preventing system downtime.
- **Wait statistics**. Wait statistics is the first place to look when you troubleshoot SQL Server performance issues. It gives deeper insights into the root cause of a system and is very helpful when optimizing a slow system. You can also compare the wait statistics current values to figure out what went wrong and fix it.

Data Capture Frequency

After you decide what to capture, the next step is to decide the frequency of the data capture. The frequency governs the amount of data that is captured. Too much data will result in high storage costs; too little data will not give better understanding of the system. The frequency depends on the type of data that you capture. Performance Monitor data can be captured every 15 seconds or so. However, it is not ideal to capture SQL instance configuration data every 15 seconds.

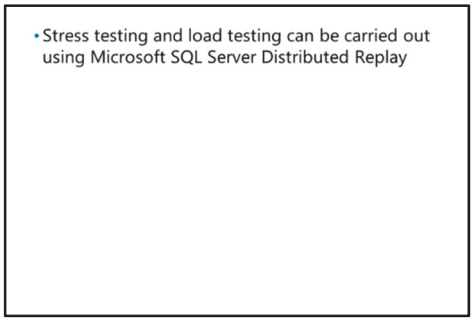
It is also necessary to have a data retention strategy. It is not advisable to keep a year or six months' worth of baseline data as this data is not likely to be of use. In addition, capturing data during off-peak hours will only add additional data to the storage without being useful. The best practice is to capture data during business peak hours and keep it for three or four months.

Data Storage

The data collected might be small, but the frequency and duration of collection might mean that it requires a large amount of storage. The recommendation is to store the baseline data in a separate database on a separate SQL Server instance. The separate database can be used to store baseline database from more than one server. You should also optimize the baseline database to get the relevant information on time. The aggregate queries may run slowly if the tables are not properly indexed.

Stress-Testing Tools

When you migrate to a new SQL Server instance, there is often a requirement to perform load tests and stress tests. Load testing measures the ability of a system to cope with a set load, and stress testing measures the ability of the system to cope with abnormal loads.

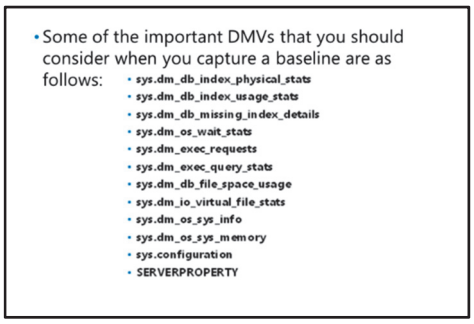


You can use SQL Server Distributed Replay to perform load testing and stress testing of a Microsoft SQL Server instance.

Stress testing is carried out in the same way as a normal trace replay by using Microsoft SQL Distributed Replay in Stress Mode, which is its default setting.

Data Collection Using DMVs

DMVs provide insights into SQL Server internal workings. The data that DMVs expose will help you to understand SQL Server internal workings and optimize the performance of your SQL Server instance. For example, the **sys.dm_db_index_physical_stats** DMV is used to get the index fragmentation level and defragment as appropriate.



The data provided can be easily captured by using Transact-SQL queries—no additional utilities are required. You can save the Transact-SQL scripts to use when you diagnose performance. For example, a script to find blocking can be saved and executed as and when necessary.

DMVs are useful in capturing a baseline because data such as wait statistics, SQLOS information, and cached query plans cannot be obtained through other sources. Data collection through certain DMVs may incur overhead on the SQL Server instance. For example, obtaining index fragmentation details using the **sys.dm_db_index_physical_stats** DMV for all indexes in a large database might take time to return and can negatively affect SQL Server performance. Some of the important DMVs that you should consider when you capture a baseline are as follows:

- **sys.dm_db_index_physical_stats**. This returns index size and fragmentation information and the forwarded record count for heaps. It is essential to track the rate of index fragmentation to decide a proper strategy for index maintenance. Using this DMV can negatively affect performance.

- **sys.dm_db_index_usage_stats**. This returns cumulative seeks, scans, lookups, and updates for an index. The information can be helpful to identify the following:
  - Indexes with high scan count and low seek count.
  - Unused indexes, which are not listed under this DMV.
  - Indexes with low seeks and high updates.
  - Frequently used indexes.
  - The last time the index was used.
- **sys.dm_db_missing_index_details**. This returns missing index details, excluding spatial indexes. You should test the index suggestions on a test server before you deploy to production.

The following example query collects index physical statistics:

Index Physical Statistics

```
SELECT
              @@SERVERNAME AS ServerName,
               ips.database_id,
               DB_NAME(ips.database_id)   AS DatabaseName,
               OBJECT_NAME(ips.object_id) AS ObjectName,
               ind.NAME                    AS IndexName,
               ips.index_id,
               ips.index_type_desc,
               ips.avg_fragmentation_in_percent,
               ips.fragment_count,
        ips.record_count,
        ips.forwarded_record_count,
               ips.page_count,
               ind.fill_factor,
              GETDATE() As DataCollectionDate
FROM   sys.Dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, N'LIMITED') AS
               ips
               INNER JOIN sys.indexes AS ind WITH (nolock)
               ON ips.[object_id] = ind.[object_id]
                AND ips.index_id = ind.index_id
-- remove this filter to collect for all databases
WHERE  ips.database_id = Db_id()
```

The information collected can be used to:

- Decide whether to rebuild or reorganize an index.
- Establish trends in index fragmentation. This can be helpful in a very busy environment, where an index that is frequently fragmented can be reorganized more often to control the fragmentation level.
- **sys.dm_os_wait_stats**. This provides aggregated wait statistics information for an instance. The waits are always accumulating even in an optimized environment. Often, wait statistics are the first place to check to diagnose performance issues. For more information about wait statistics, see Module 1, *SQL Server Architecture, Scheduling, and Waits*.
- **sys.dm_exec_requests**. This lists queries that are currently executing. It can be used to find current resource-intensive queries, long-running queries, or blocking information. The query text can be retrieved by using the **sys.dm_exec_sql_text** DMV.

The following query shows currently running workloads along with their resource consumption:

Current Executing Queries

```
SELECT
              @@SERVERNAME AS ServerName,
              DB_NAME(er.database_id) AS DatabaseName,
              er.session_id,
              (SELECT SUBSTRING(qt.[text],er.statement_start_offset/2,
    (CASE WHEN er.statement_end_offset = -1
    THEN LEN(CONVERT(nvarchar(max), qt.[text])) * 2
    ELSE er.statement_end_offset END - er.statement_start_offset)/2))
             AS QueryText,
             er.start_time,
             er.status,
             er.Command,
             er.last_wait_type,
             er.wait_type AS current_wait_type,
             er.wait_time,
             er.cpu_time,
             er.total_elapsed_time,
             er.reads,
             er.writes,
             er.logical_reads
FROM sys.dm_exec_requests er
cross apply sys.dm_exec_sql_text(er.sql_handle) qt
ORDER BY total_elapsed_time DESC
```

- **sys.dm_exec_query_stats**. This returns aggregate performance statistics for cached query plans in a SQL Server instance. The information includes query execution count, last execution time, reads, writes, CPU time, and the total number of rows that are returned. The information can be used to find resource-intensive plans. Queries with a high execution count that consume considerable amounts of resources are the ones you should check. Queries with low execution count and high resource usage—for example, weekly reports—can be ignored. You must capture this information because it is removed after the query plan is removed from the plan cache. The **sys.dm_exec_procedure_stats** DMV gives similar information about cached procedures.
- **sys.dm_db_file_space_usage**. This returns space usage information for each file in a database. You can compare the collected values with current values to identify any abrupt increase in database size. You can also use it to identify trends in database growth and predict future storage requirements.
- **sys.dm_io_virtual_file_stats**. This returns reads, writes, latency, and current size for every database file. This is an important DMV for investigating I/O bottlenecks and disk performance.

The following query shows I/O usage for all database files in a SQL Server instance:

I/O Usage

```
SELECT
DB_NAME(VFS.Database_id) AS DataBaseName,
mf.file_id,
mf.name As FileName,
vfs.Sample_Ms,
vfs.Num_Of_Reads,
vfs.Num_Of_Bytes_Read,
vfs.IO_Stall_Read_ms,
vfs.Num_Of_Writes,
vfs.Num_Of_Bytes_Written,
vfs.IO_Stall_Write_ms,
vfs.IO_Stall,
GETDATE() AS DataCollectionDate
 FROM sys.Dm_io_virtual_file_stats(NULL, NULL) vfs
 INNER JOIN sys.master_files mf ON mf.FILE_ID = vfs.FILE_ID  AND
 mf.DataBase_ID = vfs.DataBase_ID
```

- **sys.dm_os_sys_info**. This returns miscellaneous system information, such as computer start time, number of CPUs, cpu_ticks, SQL Server start time, total physical memory, and buffer pool committed memory. The information is useful to track the changes in a virtual environment.
- **sys.dm_os_sys_memory**. This returns operating system memory information, such as total physical memory available, available page file size, and system memory state.

In addition to the DMVs already mentioned, the following system views are useful:

- **sys.configuration**. This contains server-wide configuration options and their current values, such as max server memory, min server memory, and backup compression default. You can compare collected data with current values to identify problems that are caused by any changes in server configuration option. For example, a decrease in max server memory is more likely to reduce the performance. You can query collected data to identify this change and act accordingly.
- **SERVERPROPERTY**. This returns server property information, such as product version, edition, and servername. The information is needed for reporting purposes.

The following query returns SQL Server version information:

Returning SQL Server Version Information

```
SELECT
SERVERPROPERTY('ProductVersion') AS ProductVersion,
SERVERPROPERTY('ProductLevel') AS ProductLevel,
SERVERPROPERTY('Edition') AS Edition,
SERVERPROPERTY('EngineEdition') AS EngineEdition;
```

Backupset

The **backupset** table in the msdb system database stores backup history. You can use the backup history to establish trends in database size over time and predict future database growth.

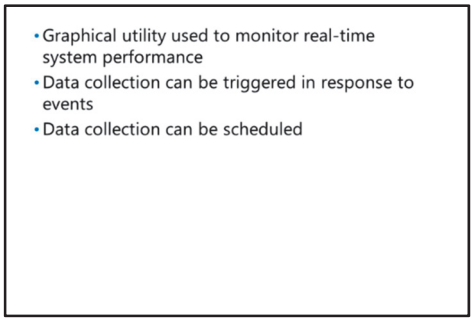
sysjobhistory

The **msdb.dbo.sysjobhistory** table stores SQL Agent job history. You can query the job history to get the long-running jobs and tune them accordingly. Additionally, you can use it to track the successful completion of the jobs.

You can query DMVs and system objects, and save relevant information in intermediate tables. You can then use the intermediate tables for analysis or to establish a baseline.

Data Collection Using Perfmon Counters

Performance Monitor is a Windows graphical utility to monitor real-time performance data or capture data over a period. You can create a baseline or compare the current system performance against an established baseline by using the captured data.



You cannot use the real-time performance data for historical analysis. It can only be used to monitor current system state and compare against the established baseline. To monitor real-time performance using Performance Monitor, follow these steps:

1. To start Performance Monitor, in the Run command window, type **perfmon**.
2. In the leftmost pane, expand **Monitoring Tools**, and then click **Performance Monitor**. This will open the Performance Monitor window in the rightmost pane.
3. To add the counters to monitor, click the **Plus Sign**.
4. When you have added the required counters, click **OK** to view performance information.

By default, Performance Monitor shows the last 100 seconds of data. This value can be changed from the Performance Monitor properties window that is opened by right-clicking on Performance Monitor and selecting properties.

There are three different types of graph available in Performance Monitor: Line, Histogram, and Report.

Performance Monitor provides data collector sets to automate collection of selected counters. There are two types of data collector sets; system and user defined. The system set include OS and network specific counters but does not include SQL Server counters. The data collector sets can also be triggered to start when a specific event or alerts occurs. For example, a data collector set can be started when the available memory is less than 100 MB. To set up and schedule a data collector set to collect performance counters, follow these steps:

1. To start Performance Monitor, in the Run command window, type **perfmon**.
2. In the **Performance Monitor** window, in the leftmost pane, expand **Data Collector Sets**, right-click **User Defined**, click **New**, and then click **Data Collector Set**. The **Create new Data Collector Set** dialog box will appear.
3. In the **Create new Data Collector Set** dialog box, type a name, click **Create manually**, and then click **Next**.
4. In the **Create data logs** section, select **Performance counter**, and then click **Next**.
5. Choose the appropriate performance counters.
6. Specify a sampling interval, and then click **Finish**. You can now schedule the data collector or execute manually as required.

To configure the data collector set:

1. Right-click the data collector set you created, and then click **Properties**.
2. On the **Schedule** tab, click **Add**, and in the **Folder Action** dialog box, schedule the collection.
3. On the **Stop Condition** tab, specify a duration or maximum size for the set.

The data collector set will use your schedule to start collecting data or you can manually start it by using the **Action** menu.

This section describes some of the important performance counters to collect.

CPU usage:

- Processor:
  - %Processor Time
  - %Privileged Time
- Process (sqlservr.exe):
  - %Processor Time
  - %Privileged Time

The %Processor Time counter gives information about the total CPU usage and should be monitored for each available CPU. The Process (sqlservr.exe)/% Processor Time counter details how much CPU the SQL Server instance is using. If high CPU usage is a result of another application, you should investigate options for tuning that application. Occasional CPU spikes may occur and should not be a matter of concern, but you should investigate prolonged values of greater than 80 percent.

Memory Usage:

- Memory
  - Available Mbytes
- SQL Server:Buffer Manager
  - Lazy writes/sec
  - Buffer cache hit ratio
  - Page life expectancy
  - Page reads/sec
  - Page writes/sec
- SQL Server:Memory Manager
  - Total Server Memory (KB)
  - Target Server Memory (KB)

The Memory/Available Mbytes counter shows the amount of physical memory, in megabytes, that is immediately available for allocation to a process or for system use. This should be ideally above 300 MB. When it drops below 64 MB, on most servers, Windows will display low memory notifications. The SQLOS reacts to these notifications by reducing its memory usage.

The page life expectancy counter shows the amount of time that data pages stay in the buffer pool. Ideally, this should be above 300 seconds. If page life expectancy is below 300 seconds, investigate other buffer manager counters to get to the root cause. If the Lazy writes/sec counter is consistently non-zero, along with low page life expectancy and high page reads/sec and page writes/sec counters, there is a buffer pool contention. The buffer cache hit ratio counter shows how often SQL Server gets a page from the buffer rather than the disk. This should ideally be close to 100 percent.

The total server memory is the current amount of memory that an instance of SQL Server is using. The target server memory is the amount of memory that is allocated to a SQL Server instance. Ideally, total server memory is equal to target server memory on a stable system. If total server memory is less than the target server memory, it means that SQL Server is still populating the cache and loading the data pages into memory. A sudden decrease in total server memory indicates a problem and needs further investigation.

Disk usage:

- Physical Disk
  - Avg. Disk sec/Read
  - Avg. Disk Bytes/Read
  - Avg. Disk sec/Write
  - Avg. Disk Bytes/Write
- Paging File
  - %Usage
- SQL Server: Access Methods
  - Forwarded Records/sec
  - Full Scans/sec
  - Index Searches/sec
  - Page splits/sec

The Avg. Disk sec/Read counter shows the average time taken, in seconds, to read data from disk. Similarly, the Avg. Disk sec/write counter shows the average time taken, in seconds, to write data to disk. High values of these counters may not indicate hardware issues. Poorly tuned queries and missing or unused indexes may result in high I/O usage.

The forwarded records/sec value should be less than 10 per 100 batch requests/sec. Consider creating a clustered index if this counter is consistently high. A high value for the Full Scans/sec counter may cause high CPU usage. Full scans on smaller tables are fine; however, a large number of scans on big tables should be investigated. The counter page splits/sec value should ideally be less than 20 per 100 batch requests/sec. A high number of page splits may result in blocking, high I/O, or memory pressure. Set an appropriate fill factor value to balance out page splits.

SQL Server statistics:

- SQL Server: SQL Statistics
  - Batch requests/sec
  - SQL compilations/sec
  - SQL recompilations/sec
- SQL Server: General Statistics
  - User connections
  - Logins/sec
  - Logouts/sec

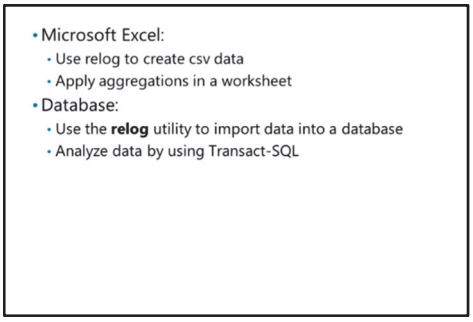
The batch requests/sec value is the number of Transact-SQL batch requests that SQL Server receives per second. The SQL compilations/sec counter shows the number of times per second that SQL compilations have occurred. A high number of SQL compilations and recompilations may cause CPU bottleneck. The SQL compilations/sec value should ideally be less than 10 percent of the number of batch requests/sec, and SQL recompilations should ideally be less than 10 percent of the total number of SQL compilations/sec.

The user connections counter shows the number of users who are currently connected to a SQL Server instance. The logins/sec and logouts/sec values should ideally be less than two. If the value is consistently greater than two, it means that the connection pooling is not being correctly used by the application.

This is not an exhaustive list of counters to be considered but these are good starting points when baselining SQL Server.

Analyzing Collected Data

The data you collect will not be useful until you can analyze it to get meaningful information. You can analyze Performance Monitor data by using Microsoft Excel® or by importing the performance logs into a database:



- **Microsoft Excel**. The performance data, in csv format, can be manually analyzed in Excel. If the data is collected in binary format, the binary log file can be converted to csv format with the **relog** command-line utility. The **relog** utility ships with Windows and does not require a separate installation. The following example shows a typical command to convert a binary log file to csv format by using the **relog** utility:

```
relog <binary file path> -f csv –o <csv file path>
```

Analyzing data in Excel can be a tedious task. The column headers need to be formatted, the performance data requires formatting, and then aggregate columns need to be added to get the maximum and minimum standard deviation for the counter values. This becomes even more tedious when more than one file is to be analyzed.

- **Database**. The performance data can be imported into SQL Server and analyzed by using Transact-SQL statements. The performance data can be imported into a database manually, loaded by using SQL Server Integration Services, or loaded by using the **relog** utility. The simplest method is probably the use of the **relog** command-line utility. The following example shows the syntax to import a performance log into a database by using the **relog** utility:

```
relog <binary file path> -f SQL -o SQL:<ODBC Connection>!<display string>
```

The **relog** utility accepts a binary log file and inserts it into the database that the Open Database Connectivity (ODBC) connection specifies. The display string identifies the binary log file or the data collector set within the database. The **relog** utility imports data into three different tables, as follows:

- **DisplayToID**. This lists each data collector set that is imported into the database. A unique identifier uniquely identifies each data collector set. The data collector is identified by the display string value that is specified when importing the data, as shown in the preceding **relog** command syntax. The table also contains the number of records that are imported and the log start and log stop time.

- **CounterDetails**. This contains one row for each counter that is present in the performance log file. Every counter is uniquely identified by a unique counter id. Each counter has an associated machine name. This is helpful in identifying counter values from different computers.
- **CounterData**. The **CounterData** table stores the actual counter values. The important columns are **GUID**, **counterID**, **counter value**, and **counterdatetime**. The **GUID** columns link to the **DisplayToID GUID** column. The **counterID** columns link to the **CounterDetails counterID** column. The **counter value** column contains the actual counter value, and the **counterdatetime** column contains the time that the value was recorded.

These three tables can be queried to get different counter values, as shown in the following example:

Counter Values

```
SELECT
            dd.DisplayString,
            cd.CounterDateTime,
            cdt.ObjectName,
            cdt.CounterName,
            cd.CounterValue
FROM dbo.CounterData cd
JOIN dbo.DisplayToID dd ON cd.GUID=dd.GUID
JOIN dbo.CounterDetails cdt on cd.CounterID=cdt.CounterID
WHERE did.DisplayString='SQLPerf'
ORDER BY cdt.ObjectName, cdt.CounterName,cd.RecordIndex
```

The preceding query will list the counter values for the display string "SQLPerf." To get the data from all display strings, remove the filter on the **DisplayString** column in the preceding query.

You can aggregate data to form a baseline, and you can import data from multiple data collector sets with different display strings, and then compare them to diagnose issues.

A sample query to get aggregate data from a particular display string or data collector set.

Aggregating Data for a Single Display String or Data Collector Set

```
SELECT CONVERT(VARCHAR(10), cd.CounterDateTime, 101) AS CounterDateTime
            ,RTRIM(cdt.ObjectName) AS ObjectName
            ,RTRIM(cdt.CounterName) AS CounterName
            ,MIN(cd.CounterValue) AS "Minimum value"
            ,MAX(cd.CounterValue) AS "Maximum value"
            ,Avg(cd.CounterValue) AS "Average value"
FROM dbo.CounterData cd
INNER JOIN dbo.DisplayToID did ON cd.GUID = did.GUID
INNER JOIN dbo.CounterDetails cdt ON cd.CounterID = cdt.CounterID
WHERE did.DisplayString = 'SQLPerf'
GROUP BY CONVERT(VARCHAR(10), cd.CounterDateTime, 101)
            ,RTRIM(cdt.ObjectName)
            ,RTRIM(cdt.CounterName)
ORDER BY RTRIM(cdt.ObjectName)
            ,RTRIM(cdt.CounterName)
```

A sample query to display aggregate data from more than one display string.

Aggregate Data for More Than One Display String

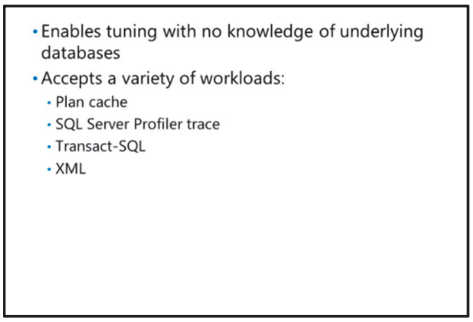
```
SELECT did.DisplayString AS DataCollectorSet
             ,RTRIM(cdt.ObjectName) AS ObjectName
             ,RTRIM(cdt.CounterName) AS CounterName
             ,MIN(cd.CounterValue) AS "Minimum value"
             ,MAX(cd.CounterValue) AS "Maximum value"
             ,Avg(cd.CounterValue) AS "Average  value"
FROM dbo.CounterData cd
INNER JOIN dbo.DisplayToID did ON cd.GUID = did.GUID
INNER JOIN dbo.CounterDetails cdt ON cd.CounterID = cdt.CounterID
GROUP BY did.DisplayString
             ,RTRIM(cdt.ObjectName)
             ,RTRIM(cdt.CounterName)
ORDER BY RTRIM(cdt.ObjectName)
             ,RTRIM(cdt.CounterName)
```

You can use the preceding query to compare counter values from different data collectors. Query results can be written to tables or flat files for further analysis.

Analyzing data manually is relatively straightforward and gives a lot of flexibility with the type of analysis you can perform.

Database Engine Tuning Advisor

The Database Engine Tuning Advisor has an advantage over more traditional tuning methods because it does not require the user to have any knowledge of the underlying database or the internal workings of SQL Server.



Before the Database Engine Tuning Advisor is used for the first time, a user with the **sysadmin** permission must initialize the tool. Initialization is carried out by connecting to a database instance from the Database Engine Tuning Advisor. Initialization creates a number of tables in the msdb database that are used for tuning workloads.

For more information about the Database Engine Tuning Advisor, see the topic *Start and Use the Database Engine Tuning Advisor* in Microsoft Docs.

Start and Use the Database Engine Tuning Advisor.

http://aka.ms/Mdpzxr

The Database Engine Tuning Advisor can accept any of the following as workloads:

- Plan cache
- SQL Server Profiler trace file or fable
- Transact-SQL script
- XML file

Demonstration: Collecting Performance Data using DMVs

In this demonstration, you will see how to collect performance data by using DMVs.

Demonstration Steps

1. Start SQL Server Management Studio, and then connect to the **MIA_SQL** database engine instance by using Windows authentication.
2. In the **D:\Demofiles\Mod10\Demo02** folder, open the **DMVDataCollection.sql** file.
3. Select the code under **Step 1 - Collect physical index stats**, and then click **Execute**. Note the index details, particularly the **avg_fragementation_in_percent** column.
4. Select the code under **Step 2 - Return current executing queries**, and then click **Execute**. Note that the query returns all queries that are currently executing on the instance.
5. Select the code under **Step 3 - Return I/O usage**, and then click **Execute**. Note that the query returns I/O stats for all database files.
6. Close SQL Server Management Studio without saving changes.

Verify the correctness of the statement by placing a mark in the column to the right.

Statement	Answer
True or false? Performance Monitor is useful for capturing baseline data, such as wait statistics, SQLOS information, and cached query plans that cannot be obtained through other sources.	

Lab: Monitoring, Tracing, and Baselining

Scenario

You are investigating why a new SQL Server instance is so slow; users frequently complain that their workloads run very slowly during peak hours of business. In addition, to troubleshoot performance issues in future and take more informed corrective measures, you decide to establish a baseline for SQL Server performance. In this lab, you will set up data collection for analyzing workload during peak business hours and implement a baseline methodology to collect performance data at frequent intervals, so that comparisons can be made with the baseline.

Objectives

After completing this lab, you will be able to:

- Collect and analyze performance data by using Extended Events.
- Implement a methodology to establish a baseline.

Estimated Time: 60 minutes

Virtual machine: **10987C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Collecting and Analyzing Data Using Extended Events

Scenario

You have been asked to prepare a reusable Extended Events session to collect and analyze workload.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Set up an Extended Events Session

3. Execute Workload

4. Analyze Collected Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 10987C-MIA-DC and 10987C-MIA-SQL virtual machines are running, and then log on to the 10987C-MIA-SQL machine as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab10\Starter** folder, run **Setup.cmd** as an Administrator.

▶ Task 2: Set up an Extended Events Session

1. Create an Extended Events session to capture the **sqlserver.error_reported**, **sqlserver.module_end**, **sqlserver.sp_statement_completed**, and **sqlserver.sql_batch_completed** events with a **ring_buffer** target. In the **D:\Labfiles\Lab10\Starter** folder, the **SetupExtendedEvent.sql** file has a possible solution script.
2. Watch Live Data for the Extended Events session.

▶ Task 3: Execute Workload

1. In the **D:\Labfiles\Lab10\Starter** folder, in the **RunWorkload.cmd** file, run the workload multiple times to generate event data for the **Extended Event** session.
2. In the **AnalyzeSQLEE** session live data window, stop the feed data, and then add the **duration**, **query_hash**, and **statement** columns to the view.

▶ Task 4: Analyze Collected Data

1. In the AnalyzeSQLEE Extended Events live data window, group the data on **query_hash** data, and then aggregate the data on average of **duration**. Sort the data in descending order of duration so that statements that take the highest average time are at the top.
2. Review the data in one of the query hash rows.
3. Drop the AnalyzeSQLEE Extended Events session.

Results: After completing this exercise, you will have set up an Extended Events session that collects performance data for a workload and analyzed the data.

Exercise 2: Implementing Baseline Methodology

Scenario

You are asked to set up a baseline methodology to collect data that can be used as baseline for comparison if the instance develops performance issues.

The main tasks for this exercise are as follows:

1. Set up Data Collection Scripts
2. Execute Workload
3. Analyze Data

▶ Task 1: Set up Data Collection Scripts

- Create a database named **baseline** by using default settings, and then clear the wait statistics for the database. In the D:\Labfiles\Lab10\Starter\10987-10 folder, the **PrepareScript.sql** Transact-SQL file has a sample solution script.

▶ Task 2: Execute Workload

1. Create a job from the **WaitsCollectorJob.sql** Transact-SQL file in the **D:\Labfiles\Lab10\Starter** folder.
2. Run the **waits_collections** job to collect statistics before and after running the **RunWorkload.cmd** file multiple times.

▶ Task 3: Analyze Data

1. Using the collected waits data, write and execute a query to find the waits for the workload. In the **D:\Labfiles\Lab10\Starter\10987-10** folder, the **WaitBaselineDelta.sql** file has a sample solution script.
2. Using the collected waits data, write and execute a query to find the percentage of waits. In the **D:\Labfiles\Lab10\Starter\10987-10** folder, the **WaitBaselinePercentage.sql** file has a sample solution script.

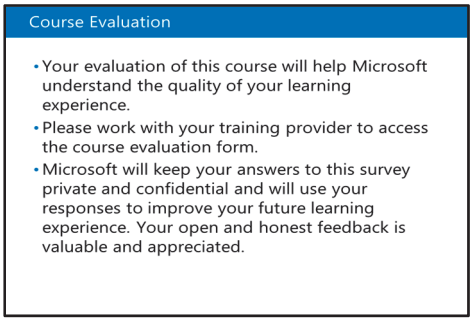
3. Using the collected waits data, write and execute a query to find the top 10 waits. In the **D:\Labfiles\Lab10\Starter\10987-10** folder, the **WaitBaselineTop10.sql** file has a sample solution script.
4. Close SQL Server Management Studio without saving any changes.
5. Close File Explorer.

Results: After completing this exercise, you will have implemented a baseline for a workload.

Module Review and Takeaways

Constant monitoring and tracing is the key to identifying performance issues that are happening in their environment. Benchmarks and baselines are the key to implanting a robust performance troubleshooting methodology.

Course Evaluation



Your evaluation of this course will help Microsoft understand the quality of your learning experience.

Please work with your training provider to access the course evaluation form.

Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.

Module 1: SQL Server Architecture, Scheduling, and Waits

Lab: SQL Server Architecture, Scheduling, and Waits

Exercise 1: Recording CPU and NUMA Configuration

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. In the D:\Labfiles\Lab01\Starter folder, right-click **Setup.cmd**, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Record CPU Configuration

1. Start SQL Server Management Studio, and then connect to the **MIA-SQL** database engine by using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the Open Project window, open the **D:\Labfiles\Lab01\Starter\Project\Project.ssmssln** project.
4. In Solution Explorer, double-click the **Lab Exercise 01 - CPU and NUMA.sql** query. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**, or press Ctrl+Alt+L on the keyboard.)
5. Under the heading for Task 1, type the following:

```
SELECT cpu_count, hyperthread_ratio
FROM sys.dm_os_sys_info;
```

6. Highlight the query that you have typed, and then click **Execute** (or press F5 or Ctrl+E).

▶ Task 3: Record CPU-Related Configuration Settings

1. In SQL Server Management Studio, in the query pane, under the heading for Task 2, edit the query so that it reads as follows:

```
SELECT *
FROM sys.configurations
WHERE name IN
  ('affinity mask',
   'affinity64 mask',
   'cost threshold for parallelism',
   'lightweight pooling',
   'max degree of parallelism',
   'max worker threads',
   'priority boost');
```

2. Highlight the query that you have typed, and then click **Execute**.

▶ Task 4: Record NUMA Configuration

1. In SQL Server Management Studio, in the query pane, under the heading for Task 3, type the following:

```
SELECT * FROM sys.dm_os_nodes;
```

2. Highlight the query that you have typed, and then click **Execute**.

▶ Task 5: Record Distribution of Schedulers Across NUMA Nodes

1. In SQL Server Management Studio, in the query pane, edit the query under the heading for Task 4 so that it reads as follows:

```
SELECT OSS.scheduler_id, OSS.status, OSS.parent_node_id, OSN.node_state_desc
FROM sys.dm_os_schedulers
AS OSS
JOIN sys.dm_os_nodes AS OSN
ON OSS.parent_node_id = OSN.node_id;
```

2. Highlight the query, and then click **Execute**.

Results: At the end of this exercise, you will be able to:

Record CPU configuration.

Record NUMA configuration.

Exercise 2: Monitoring Schedulers and User Requests

▶ Task 1: Start the Workload

1. Open Windows Explorer, and then navigate to D:\Labfiles\Lab01\Starter.
2. Right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**. Leave the script running, and continue with the lab.

▶ Task 2: Monitor Workload Pressure on Schedulers

1. In Solution Explorer, double-click the **Lab Exercise 02 - Monitor Schedulers.sql** query.
2. When the query window opens, in the query pane, type the following query after the Task 2 description:

```
SELECT *
FROM sys.dm_os_schedulers
WHERE status = 'VISIBLE ONLINE';
```

3. Highlight the query, and then click **Execute**.
4. Execute this query several times and notice how the column values change. The value of runnable_tasks_count gives an indication of the length of the runnable queue, and therefore of CPU pressure.

▶ Task 3: Monitor Task Status for User Requests

1. In the query pane, type the following under the heading for Task 3:

```
SELECT *
FROM sys.dm_exec_requests
WHERE session_id > 50;
```

2. Highlight the code that you have typed, and then click **Execute**.
3. The workload sessions will be those with a command of **SELECT** and a non-NULL **sql_handle**. The workload sessions are likely to be waiting for the CXPACKET wait type.
4. The CXPACKET wait type indicates that parallel tasks are waiting for other tasks that are part of the same request to finish working.

▶ Task 4: Stop the Workload

- In the query pane, highlight the code under the heading for Task 4, and then click **Execute**. This will stop the workload.

Results: At the end of this exercise, you will be able to:

Monitor workload pressure on schedulers.

Monitor thread status for user requests.

Exercise 3: Monitoring Waiting Tasks and Recording Wait Statistics

▶ Task 1: Clear Wait Statistics

1. In Solution Explorer, double-click the **Lab Exercise 03 - Waits.sql** query.
2. When the query window opens, in the query pane, type the following query after the Task 1 description:

```
DBCC SQLPERF('sys.dm_os_wait_stats', CLEAR);
```

3. Highlight the query, and then click **Execute**.

▶ Task 2: Check Current Wait Statistics

1. In the query pane, type the following under the heading for Task 2:

```
SELECT * FROM sys.dm_os_wait_stats;
```

2. Highlight the code that you have typed, and then click **Execute**.
3. Notice that most of the column values contain zero.

▶ Task 3: Start the Workload

1. Open Windows Explorer, and then navigate to D:\Labfiles\Lab01\Starter.
2. Right-click **start_load_exercise_03.ps1**, and then click **Run with PowerShell**. If prompted press **y** and then **Enter**. Leave the script running, and continue with the lab.

▶ Task 4: Monitor Waiting Tasks While the Workload Is Running

1. In SQL Server Management Studio, in the query pane, type the following under the heading for Task 4:

```
SELECT * FROM sys.dm_os_waiting_tasks WHERE session_id > 50;
```

2. Highlight the code that you have typed, and then click **Execute**.
3. You will see LCK_M_S waits in the **wait_type** column.

▶ Task 5: Record Wait Statistics for Analysis

1. Under the heading for Task 5, highlight the first query, and then click **Execute**.
2. Edit the second query under Task 5 so that it reads as follows:

```
SELECT ws.*
FROM #wait_stats_snapshot AS snap
JOIN sys.dm_os_wait_stats AS ws
ON ws.wait_type = snap.wait_type
WHERE ws.wait_time_ms - snap.wait_time_ms > 0
ORDER BY ws.wait_time_ms - snap.wait_time_ms DESC;
```

3. Highlight the code that you have amended, and then click **Execute**.

▶ Task 6: Stop the Workload

- In the query pane, highlight the code under the heading for Task 6, and then click **Execute**. This will stop the workload.

Results: At the end of this exercise, you will be able to:

Monitor the waiting tasks list.

Capture and review wait statistics.

Module 2: SQL Server I/O

Lab: Testing Storage Performance

Exercise 1: Configuring and Executing Diskspd

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

▶ Task 2: Execute Diskspd

1. On 10987C-MIA-SQL, right-click the Start button, and then click **Windows PowerShell (Admin)**.
2. In the **User Account Control** dialog box, click **Yes**.
3. In the **Administrator: Windows PowerShell** window, type the following code, and then press Enter:

```
Cd D:\Labfiles\Lab02\Diskspd-v2.0.15\amd64fre
```

4. Type the following code, and then press Enter:

```
.\diskspd.exe -d180 -c2G -r -t4 -w40 -o32 -b64K -L D:\Labfiles\Lab02\test.dat;
```

5. After a few minutes, review the output of the test. Notice that the output includes:
   - CPU activity during the test, for each CPU.
   - Total I/O, read I/O, and write I/O statistics for each thread.
   - Total speed, read speed, and write speed by percentile.
6. When you have finished your review, delete the test file, by typing the following code, and then press Enter:

```
del D:\Labfiles\Lab02\test.dat
```

7. Close Windows PowerShell® when you have finished.

Results: At the end of this exercise, you will have configured and run Diskspd to test I/O subsystem performance.

Module 3: Database Structures

Lab: Database Structures

Exercise 1: Exploring Page Allocation Structure

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab03\Starter** folder, right-click **Setup.cmd** and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and wait for the script to finish.

▶ Task 2: Explore Page Structure

1. On the taskbar, click **Microsoft SQL Server Management Studio 17**.
2. In the **Connect to Server** dialog box, click **Connect**.
3. Click **New Query**, type the following Transact-SQL, and then click **Execute**:

```
USE AdventureWorks;
GO
SELECT db_name(database_id) Database_Name, object_name([object_id]) Table_Name,
allocation_unit_type, allocation_unit_type_desc
  allocated_page_file_id, allocated_page_page_id, page_type, page_type_desc FROM
sys.dm_db_database_page_allocations(db_id('AdventureWorks'),object_id('Person.Contact
Type'),NULL,NUll,'DETAILED');
GO
```

4. Examine the query results, noting that there are four pages: two IAM pages, an index page, and a data page. Note the value in the **allocated_page_page_id** column for the row with the value **DATA_PAGE** in the **page_type_desc** column.

▶ Task 3: Explore Record Structure

1. In the query window, type the following Transact-SQL to enable trace flag 3604, highlight the code, and then click **Execute**:

```
DBCC TRACEON(3604);
GO
```

2. Type the following Transact-SQL, replacing the characters **XXX** with the page id you noted in the previous lab task; highlight the code, and then click **Execute**:

```
DBCC PAGE(11,1,XXX,2)
GO
```

3. Examine the query results, noting the page type, allocation status and other page information.

Results: After completing this exercise, you will have explored data page and record structure.

Exercise 2: Configuring Instant File Initialization

▶ Task 1: Reset Security Policy

1. Click **Start**, type **secpol.msc**, and then click **secpol.msc**.
2. In the **Local Security Policy** management console, in the left pane, under **Security Settings**, expand **Local Policies**, and then click **User Rights Assignment**.
3. In the right pane, under **Policy**, double-click **Perform volume maintenance tasks**.
4. In the **Perform volume maintenance tasks Properties** dialog box, on the **Local Security Setting** tab, click **Administrators**, click **Remove**, and then click **OK**.
5. Leave the **Local Security Policy** management console open for use later in the lab.

▶ Task 2: Record Workload Execution Time

1. In SQL Server Management Studio, in Object Explorer, right-click the **MIA-SQL** database instance, and click **Restart**.
2. In the **User Account Control** dialog box, click **Yes**.
3. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes**.
4. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes** to confirm you want to restart dependent services.
5. When the services have restarted, on the **File** menu, point to **Open**, and then click **File**.
6. In the **Open File** dialog box, navigate to the **D:\Labfiles\Lab03\Starter** folder, and then double-click **InstantFileInit.sql**.
7. Click **Execute** and note how long the script takes to run.
8. Leave SQL Server Management Studio open.

▶ Task 3: Enable Instant File Initialization and Compare Run Time

1. In the Local Security Policy management console, in the right pane, under **Policy**, double-click **Perform volume maintenance tasks**.
2. In the **Perform volume maintenance tasks Properties** dialog box, on the **Local Security Setting** tab, click **Add User or Group**.
3. In the **Select Users, Computers, Service Accounts, or Groups** dialog box, in the **Enter the object names to select** box, type **Administrators**, and then click **OK**.
4. In the **Perform volume maintenance tasks Properties** dialog box, click **OK**.
5. In SQL Server Management Studio, in Object Explorer, right-click the **MIA-SQL** database instance, and then click **Restart**.
6. In the **User Account Control** dialog box click **Yes**.
7. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes**.
8. In the **Microsoft SQL Server Management Studio** dialog box, click **Yes** to confirm you want to restart dependent services.
9. When SQL Server services have restarted, click **Execute** to run the code in the query window again.
10. Compare the run time of the script with and without instant file initialization enabled.
11. Close SQL Server Management Studio, without saving any changes.

Results: At the end of this exercise, you will have enabled instant file initialization.

Exercise 3: Reconfiguring tempdb Data Files

▶ Task 1: Execute Workload and Record Latch Contention Metrics

1. In the **D:\Labfiles\Lab03\Starter** folder, right-click the **tempdbLoad.cmd** file, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. Wait until all the command windows have closed.
4. On the taskbar, click **Microsoft SQL Server Management Studio 17**.
5. In the **Connect to Server** dialog box, click **Connect**.
6. Click **New Query**, type the following Transact-SQL, and then click **Execute**:

```
SELECT *
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'PAGELATCH%'
```

7. Note the wait stats for the SQL Server instance.

▶ Task 2: Add Additional Data Files to tempdb

1. On the **File** menu, point to **Open**, and then click **File**.
2. In the **Open File** dialog box, in the **D:\Labfiles\Lab03\Starter** folder, double-click **addTempdbFiles.sql**.
3. Click **Execute** to run the code.

▶ Task 3: Measure Performance Improvement

1. In the **D:\Labfiles\Lab03\Starter** folder, right-click the **tempdbLoad.cmd** file, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. Wait until all the command windows have closed.
4. In SQL Server Management Studio, click **New Query**, type the following Transact-SQL, and then click **Execute**:

```
select *
from sys.dm_os_wait_stats
where wait_type like 'PAGELATCH%'
```

5. Note the wait stats for the SQL server instance.
6. Compare the previous wait stats figures with those from before the additional **tempdb** files were added, noting the reduced waits with more **tempdb** files.
7. Close SQL Server Management Studio without saving changes.

Results: After completing this lab, **tempdb** will be using multiple data files.

Module 4: SQL Server Memory

Lab: SQL Server Memory

Exercise 1: Reconfigure SQL Server Memory

▶ Task 1: Execute Workload and Record Memory Wait

1. Ensure that the **10987C-MIA-DC** and **10987C-MIA-SQL** virtual machines are both running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Navigate to the folder D:\Labfiles\Lab04\Starter, right-click the file **Setup.cmd**, and then click **Run as administrator**. In the **User Account Control** dialog box, click **Yes**.
3. Start SQL Server Management Studio and connect to the **MIA-SQL** SQL Server instance using Windows Authentication.
4. In File Explorer, navigate to the D:\Labfiles\Lab04\Starter folder, right-click **loadscript.ps1**, and then click **Run with PowerShell**.
5. If a message is displayed asking you to confirm a change in execution policy, type **Y** and then press ENTER.
6. When the script completes, return to SQL Server Management Studio.
7. Click **New Query**, in the query pane, type the following query, and then click **Execute**:

```
SELECT *
 FROM sys.dm_os_wait_stats
WHERE wait_type = 'MEMORY_ALLOCATION_EXT'
```

   Record the results of the query.

8. Leave SQL Server Management Studio open for the next task.

▶ Task 2: Set Min and Max Memory Appropriately

1. In SQL Server Management Studio, click **New Query**, type the following code, and then click **Execute**:

```
EXEC sp_configure N'Min Server Memory','512';
EXEC sp_configure N'Max Server Memory','4096';
```

2. In Object Explorer, right-click the **MIA-SQL** instance, and then click **Restart**.
3. In the **User Account Control** dialog, click **Yes**.
4. In the **Microsoft SQL Server Management Studio** dialog click **Yes** to confirm you wish to restart the service.
5. In the **Microsoft SQL Server Management Studio** dialog, click **Yes** to confirm you want to restart dependent services.

▶ Task 3: Execute Workload, Record Memory Wait and Measure Performance Improvement

1. Navigate to the folder D:\Labfiles\Lab04\Starter, right-click the script **loadscript.ps1**, and then click **Run with PowerShell**.
2. When the script completes, return to SQL Server Management Studio, click **New Query**, type the following code, and then click **Execute**:

```
select *
from sys.dm_os_wait_stats
 where wait_type = 'MEMORY_ALLOCATION_EXT'
```

3. Record the results of the query.
4. Compare the results you obtained before and after you changed Min Server Memory and Max Server Memory.

Results: After this lab, the SQL Server memory settings will be reconfigured.

Module 5: SQL Server Concurrency

Lab: Concurrency and Transactions

Exercise 1: Implement Snapshot Isolation

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. In the **D:\Labfiles\Lab05\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Clear Wait Statistics

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab05\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click the query **Lab Exercise 01 - snapshot isolation.sql**. (If Solution Explorer is not visible, select **Solution Explorer** on the **View** menu or press Ctrl+Alt+L on the keyboard.)
5. To clear wait statistics, select the query under the comment that begins **Task 1**, and then click **Execute**.

▶ Task 3: Run the Workload

1. Open Windows Explorer and navigate to the **D:\Labfiles\Lab05\Starter** folder.
2. Right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. If a message is displayed asking you to confirm a change in execution policy, type **Y** and then press ENTER.
4. Wait for the workload to complete and then press ENTER to close the window.

▶ Task 4: Capture Lock Wait Statistics

1. In SQL Server Management Studio, in the query pane, edit the query under the comment that begins **Task 3** so that it reads:

```
SELECT wait_type, waiting_tasks_count, wait_time_ms,
max_wait_time_ms, signal_wait_time_ms
INTO #task3
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'LCK%'
AND wait_time_ms > 0
ORDER BY wait_time_ms DESC;
```

2. Select the query you have amended and click **Execute**.

▶ Task 5: Enable SNAPSHOT Isolation

1. In SQL Server Management Studio Object Explorer, under MIA-SQL, expand **Databases**.
2. Right-click **AdventureWorks** and then click **Properties**.
3. In the **Database Properties – AdventureWorks** dialog box, on the **Options** page, in the **Miscellaneous** section, change the value of the **Allow Snapshot Isolation** setting to **True**, and then click **OK**.

▶ Task 6: Implement Snapshot Isolation

1. In Solution Explorer, double-click the query **Lab Exercise 01 – stored procedure.sql**.
2. Amend the stored procedure definition in the file so that it reads:

```
USE AdventureWorks;
GO
ALTER PROC Proseware.up_Campaign_Report
AS
   SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
   SELECT TOP 10 * FROM Sales.SalesTerritory AS T
   JOIN (
          SELECT CampaignTerritoryID,
          DATEPART(MONTH, CampaignStartDate) as start_month_number,
          DATEPART(MONTH, CampaignEndDate) as end_month_number,
          COUNT(*) AS campaign_count
          FROM Proseware.Campaign
          GROUP BY CampaignTerritoryID, DATEPART(MONTH,
CampaignStartDate),DATEPART(MONTH, CampaignEndDate)
   ) AS x
   ON x.CampaignTerritoryID = T.TerritoryID
   ORDER BY campaign_count;
GO
```

3. Click **Execute**.

▶ Task 7: Rerun the Workload

1. In the SQL Server Management Studio query window for **Lab Exercise 01 - snapshot isolation.sql**, select the query under the comment that begins **Task 1**, and then click **Execute**.
2. Switch to File Explorer and in the **D:\Labfiles\Lab05\Starter** folder, right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. Wait for the workload to complete.

▶ Task 8: Capture New Lock Wait Statistics

1. In SQL Server Management Studio, in the query window for **Lab Exercise 01 - snapshot isolation.sql**, amend the query under the comment that begins **Task 8** so that it reads:

```
SELECT wait_type, waiting_tasks_count, wait_time_ms,
max_wait_time_ms, signal_wait_time_ms
INTO #task8
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'LCK%'
AND wait_time_ms > 0
ORDER BY wait_time_ms DESC;
```

2. Select the query you have amended and click **Execute**.

▶ Task 9: Compare Overall Lock Wait Time

1. In SQL Server Management Studio, in the query pane, select the query under the comment that begins **Task 9** and click **Execute**. Compare the total **wait_time_ms** you have captured between the **#task3** and **#task8** temporary tables. Note that the wait time in the **#task8** table—after SNAPSHOT isolation was implemented—is lower.
2. Close the query windows for **Lab Exercise 01 - snapshot isolation.sql** and **Lab Exercise 01 – stored procedure.sql** without saving changes, but leave SQL Server Management Studio open for the next exercise.

Results: After this exercise, the **AdventureWorks** database will be configured to use the SNAPSHOT isolation level.

Exercise 2: Implement Partition Level Locking

▶ Task 1: Open Activity Monitor

1. In SQL Server Management Studio Object Explorer, right-click **MIA-SQL** and click **Activity Monitor**.
2. In Activity Monitor, click **Resource Waits** to expand the section.

▶ Task 2: Clear Wait Statistics

1. In Solution Explorer, double-click **Lab Exercise 02 - partition isolation.sql**.
2. Select the code under the comment that begins **Task 2**, and click **Execute**.

▶ Task 3: View Lock Waits in Activity Monitor

1. Switch to File Explorer and in the **D:\Labfiles\Lab05\Starter** folder, right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**.
2. Wait for the workload to complete (it will take a few minutes).
3. Switch to SQL Server Management Studio and to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
4. Switch to the PowerShell workload window and press Enter to close it.

▶ Task 4: Enable Partition Level Locking

1. Return to the SQL Server Management Studio query window where **Lab Exercise 02 - partition isolation.sql** is open.
2. Under the comment that begins **Task 5**, type:

```
USE AdventureWorks;
GO
ALTER TABLE Proseware.CampaignResponsePartitioned SET (LOCK_ESCALATION = AUTO);
GO
```

3. Select the query you have typed and click **Execute**.
4. Select the query under the comment that begins **Task 2**, then click **Execute**.

▶ Task 5: Rerun the Workload

1. Switch to File Explorer and in the **D:\Labfiles\Lab05\Starter** folder, right-click **start_load_exercise_02.ps1** and then click **Run with PowerShell**.
2. Wait for the workload to complete (it will take a few minutes).
3. Return to the **MIA-SQL - Activity Monitor** tab. In the **Resource Waits** section, note the value of **Cumulative Wait Time (sec)** for the **Lock** wait type.
4. Compare this value to the value you noted earlier in the exercise; the wait time will be considerably lower after you implemented partition level locking.
5. Switch to the PowerShell workload window and press Enter to close it.

Results: After this exercise, the **AdventureWorks** database will use partition level locking.

Module 6: Statistics and Index Internals

Lab: Statistics and Index Internals

Exercise 1: Fixing Cardinality Estimation Errors

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Using Windows Explorer, navigate to the **D:\Labfiles\Lab06\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Run the Workload

1. Open File Explorer and navigate to **D:\Labfiles\Lab06\Starter**.
2. Right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**. If a message is displayed asking you to confirm a change in execution policy, type **Y** and then press ENTER.
3. Wait a few minutes for the workload to complete.
4. Note the elapsed time reported by the script, and then press ENTER to close the PowerShell window.

▶ Task 3: List Statistics Objects

1. Start **SQL Server Management Studio (SSMS)** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab06\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click **Lab Exercise 01 - Cardinality.sql**. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**.)
5. Select the query under the comment that begins **task 2** and click **Execute**.
6. The following statistics objects look like they might be out of date:
   - **Proseware.CampaignAdvert.IX_CampaignAdvert_AdvertMedia**
   - **Proseware.WebResponse.IX_WebResponse_CampaignAdvertID**

▶ Task 4: Examine Statistics in Detail

1. Edit the first query under the comment that begins **Task 3** so that it reads:

```
DBCC SHOW_STATISTICS
('Proseware.WebResponse','IX_WebResponse_CampaignAdvertID');
```

2. Select the query you have amended and click **Execute**. Note the value of the **rows** column in the first result set.
3. Select the second query under the comment that begins **Task 3** and click **Execute**. Compare the value returned by the query to the value you noted in step 2.
4. The values are substantially different: 1,000 compared to 1 million. This degree of error in the statistics is likely to cause a cardinality estimation problem.

▶ Task 5: Update Statistics

1. Edit the first query under the comment that begins **Task 4**, so that it reads:

```
UPDATE STATISTICS Proseware.WebResponse WITH FULLSCAN;
```

2. Select the query that you have amended and click **Execute**.
3. Select the second query under the comment that begins **Task 4** and click **Execute**. The query has the following text:

```
DBCC SHOW_STATISTICS ('Proseware.WebResponse','IX_WebResponse_CampaignAdvertID');
```

4. Edit the third query under the comment that begins **Task 4** so that it reads:

```
UPDATE STATISTICS Proseware.CampaignAdvert WITH SAMPLE 50 PERCENT;
```

5. Select the query that you have amended and click **Execute**.

▶ Task 6: Rerun the Workload

1. In File Explorer, navigate to **D:\Labfiles\Lab06\Starter**.
2. Right-click **start_load_exercise_01.ps1** and then click **Run with PowerShell**.
3. Wait for the workload to complete.
4. Note the elapsed time reported by the script, and then press ENTER to close the PowerShell window.
5. With updated statistics, the execution of the workload is considerably faster.

Results: At the end of this lab, statistics in the **AdventureWorks** database will be updated.

Exercise 2: Improve Indexing

▶ Task 1: Execute the Workload

1. In SQL Server Management Studio, in Solution Explorer, double-click **Lab Exercise 02 - Workload.sql**.
2. Click **Execute**.

▶ Task 2: Examine Existing Indexes

1. In Solution Explorer, double-click **Lab Exercise 02 - Indexing.sql**.
2. Under the comment that begins **Task 2**, type:

```
USE AdventureWorks;
EXEC sp_help 'Proseware.WebResponse';
```

3. Highlight the code you have written, and then click **Execute**.
4. Information about indexes on the table will be found in the sixth result set in the Results pane.

▶ Task 3: Use the Database Engine Tuning Advisor

1. In SQL Server Management Studio, on the **Tools** menu, click **Database Engine Tuning Advisor**.
2. When the tuning advisor starts, in the **Connect to Server** dialog box, connect to the **MIA-SQL** database engine using Windows authentication.
3. In the **Workload** section, click **File**, in the text box, type **D:\Labfiles\Lab06\Starter\Project\Project\Lab Exercise 02 - Workload.sql**, and then in the **Database for workload analysis** box, click **AdventureWorks**.
4. In the **Select databases and tables to tune** list, select **AdventureWorks**. On the row for the **AdventureWorks** database, click in the **Selected Tables** column, click the drop-down arrow, clear the check box next to **Name**, and then select **WebResponse**. Click the drop-down arrow again to close the list.
5. On the toolbar, click **Start Analysis**.
6. When analysis completes, on the **Index Recommendations** tab, click the value in the **Definition** column starting **([log_date] asc** to examine the suggested index definition. Notice that the suggested index is similar to the **IX_WebResponse_log_date_CampaignAdvertID** index already on the table.
7. Close the Database Engine Tuning Advisor.

▶ Task 4: Implement a Covering Index

1. In SQL Server Management Studio, switch to the query pane where the **Lab Exercise 02 - Indexing.sql** file is open.
2. Highlight the first query under the comment that begins **Task 5**, then click **Execute** to drop the existing index.
3. Edit the second query under the comment that begins **Task 5** to read:

```
CREATE INDEX IX_WebResponse_log_date_CampaignAdvertID_browser_name
ON Proseware.WebResponse (log_date, CampaignAdvertID,browser_name)
INCLUDE (page_visit_time_seconds);
```

4. Highlight the query you have amended, and then click **Execute**.

▶ Task 5: Rerun the Workload

1. In SQL Server Management Studio, switch to **Lab Exercise 02 - Workload.sql**.
2. Click **Execute**.
3. Switch to **Lab Exercise 02 - Indexing.sql**. Select the query under the comment that begins **Task 6** and click **Execute**. The results of the query demonstrate that the new index was used.
4. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, the indexing of the **Proseware.WebResponse** table in the **AdventureWorks** database will be improved.

Exercise 3: Using Columnstore Indexes

▶ Task 1: Add a Nonclustered Columnstore Index to a Row-Based Table

1. In SQL Server Management Studio, in Solution Explorer, double-click **Lab Exercise 03 - Columnstore.sql**.
2. Amend the query under the comment that begins **Task 1** so that it reads:

```
USE AdventureWorks;
GO
CREATE COLUMNSTORE INDEX IX_NCI_WebResponse
ON Proseware.WebResponse (log_date, page_url, browser_name, page_visit_time_seconds);
GO
```

3. Select the query you have amended and click **Execute**.

▶ Task 2: Create a Table with a Clustered Columnstore Index

1. Amend the first query under the comment that begins **Task 2** so that it reads:

```
CREATE TABLE Proseware.Demographic
(  DemographicID bigint NOT NULL,
             DemoCode varchar(50),
             Code01 int,
             Code02 int,
             Code03 tinyint,
             Income decimal(18,3),
             MaritalStatus char(1),
             Gender char(1),
             INDEX PK_Proseware_Weblog CLUSTERED COLUMNSTORE
);
```

2. Select the query that you have edited and click **Execute**.
3. Select the second query under the comment that begins **Task 2** and click **Execute**.
4. After the query that you executed in the previous step, type:

```
SELECT * FROM Proseware.Demographic;
```

5. Highlight the query you have typed and click **Execute**. Verify that one row has been inserted.

▶ Task 3: Add a Nonclustered Row-Based Index to a Table with a Clustered Columnstore Index

1. Edit the first query under the comment that begins **Task 3** so that it reads:

```
CREATE UNIQUE NONCLUSTERED INDEX IX_Demographic_DemographicID
ON Proseware.Demographic (DemographicID);
```

2. Select the query you have amended and click **Execute**.
3. Select the second query under the comment that begins **Task 3** and click **Execute**.

   Note that an error is raised; this is expected behavior, because the nonclustered index prevents you from inserting duplicate data.

4. Close SQL Server Management Studio without saving any changes.

Results: At the end of this exercise, the **Proseware.WebResponse** in the **AdventureWorks** database will have a nonclustered columnstore index. A new table—**Proseware.Demographic**—will be created with a clustered columnstore index.

Module 7: Query Execution and Query Plan Analysis

Lab: Query Execution and Query Plan Analysis

Exercise 1: Improving SELECT Performance for Historical Marketing Campaign Data

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQ**L as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab07\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Collect an Actual Execution Plan

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows® authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab07\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click the **Lab Exercise 01 - tuning 1.sql**.
5. Select the query under the comment which begins **Task 1**. On the **Query** menu, click **Include Actual Execution Plan**.
6. Click **Execute**. Note the execution time.
7. In the Results pane, on the **Execution plan** tab, right-click the graphical execution plan, and then click **Save Execution Plan As**.
8. In the **Save As** dialog box, enter the file name as **D:\Labfiles\Lab07\plan1.sqlplan** and then click **Save**.
9. On the **Execution plan** tab, scroll to the far right-hand side of the actual query plan. In the top right-hand corner, position the cursor over the query plan operator, the name of which starts **Clustered Index Scan (Clustered)**. In the pop-up that appears, notice the difference between **Estimated Number of Rows** (approximately 3.74) and **Actual Number of Rows** (more than 1.8 million).

   This type of issue is caused by a poor estimate of cardinality. The table statistics indicate that the tables involved have very few rows, but in reality, the row count is much higher. Updating statistics for the tables involved will improve performance.

▶ Task 3: Rebuild Table Statistics

1. In SQL Server Management Studio, in the query pane, select the query under the comment that begins **Task 2**:

```
ALTER TABLE Proseware.Campaign REBUILD
GO
ALTER TABLE Proseware.CampaignResponse REBUILD;
GO
```

2. Click **Execute**.

▶ Task 4: Compare the New Actual Execution Plan

1. In the query pane, select the query under the comment that begins **Task 1** and click **Execute**. Note the run time of the query.
2. In the Results pane, on the **Execution plan** tab, scroll to the far right of the actual query plan. In the top right, position the cursor over the query plan operator, the name of which starts **Clustered Index Scan (Clustered)...**. In the pop-up that appears, notice the similarity in the **Estimated Number of Rows** (more than 1.8 million) and **Actual Number of Rows** (more than 1.8 million).
3. Note that the estimated and actual row counts now match almost exactly. The query will execute faster than it did previously.
4. The execution plan includes a suggestion for an index that will improve query performance.
5. On the **Execution plan** tab, right-click the graphical execution plan and click **Compare Showplan**. In the **Open** dialog box, select **D:\Labfiles\Lab07\plan1.sqlplan** and click **Open**. The new and old query execution plans will open side-by-side.
6. When you have finished your comparison, close the **Showplan Comparison** tab. Leave SSMS open for the next exercise.

Results: At the end of this exercise, you will have improved the performance of a SELECT query by analyzing the query plan.

Exercise 2: Improving Stored Procedure Performance

▶ Task 1: Collect an Actual Execution Plan

1. In SQL Server Management Studio, in Solution Explorer, double-click **Lab Exercise 02 - tuning 2.sql**.
2. In the query pane, highlight the following text, and then click **Execute**:

```
USE AdventureWorks;
GO
```

3. On the **Query** menu, click **Include Actual Execution Plan**.
4. In the query pane, under the comment that begins **Task 1**, edit the query so that it reads:

```
EXEC Proseware.up_CampaignResponse_Add
 @CampaignName = 1010000,
 @ResponseDate = '2016-03-01',
 @ConvertedToSale = 1,
 @ConvertedSaleValueUSD = 100.00;
```

5. Select the query you have edited then click **Execute**.
6. In the Results pane, on the **Execution plan** tab, right-click the graphical execution plan, and then click **Save Execution Plan As**.
7. In the **Save As** dialog box, use the file name **D:\Labfiles\Lab07\plan2.sqlplan**, then click **Save**.
8. On the **Execution plan** tab, notice that the first query in the batch has 77 percent of the total batch cost. Notice the execution plan warning on the SELECT operator in the first query, and that the selection of data from the **Proseware.Campaign** table uses an index scan.
9. There are two changes that might improve the performance of this query:
   a. You could add a nonclustered index to the **Proseware.Campaign.CampaignName** column.
   b. You could change the data type of the @CampaignName parameter of the stored procedure so that it matches the data type of the **Proseware.Campaign.CampaignName** column.

▶ Task 2: Add a Covering Index

- In the query pane, amend the query under the comment that begins **Task 2** so that it reads:

```
CREATE UNIQUE NONCLUSTERED INDEX ix_Campaign_CampaignName ON Proseware.Campaign
(CampaignName);
```

2. Select the query you have edited and click **Execute**.
3. Select the code under the comment that begins **Task 1** and click **Execute**.
4. In the Results pane, on the **Execution plan** tab, right-click the **graphical execution plan** and click **Save Execution Plan As**.
5. In the **Save As** dialog box, use the file name **D:\Labfiles\Lab07\plan3.sqlplan** and click **Save**.
6. Notice that the relative cost of the first query in the batch has reduced slightly, and that a scan of the new index is being used.
7. An index scan is used because of the data type mismatch between the @CampaignName parameter and the **CampaignName** column. The warning on the SELECT operator is still present.

▶ Task 3: Change the Data Type of the @CampaignName Parameter

1. In the query pane, under the comment that begins **Task 3**, edit the first three lines of the text in the stored procedure definition so that they read as follows:

```
ALTER PROCEDURE Proseware.up_CampaignResponse_Add
(
 @CampaignName varchar(20),
```

2. Select the query under the comment that begins **Task 3**—from ALTER PROCEDURE… to the end of the script—then click **Execute**.
3. Select the query under the comment that begins **Task 1** and click **Execute**.
4. In the Results pane, on the **Execution plan** tab, right-click the **graphical execution plan** and click **Save Execution Plan As**.
5. In the **Save As** dialog box, use the file name **D:\Labfiles\Lab07\plan4.sqlplan** and click **Save**.
6. Notice that the relative cost of the first query in the batch has reduced to approximately 20 percent. Also, notice that a seek of the new index is being used, and that the data type conversion warning no longer appears.
7. Close SSMS without saving any changes.

Results: At the end of this lab, you will have examined a query execution plan for a stored procedure and implemented performance improvements by adding a covering index and eliminating an implicit data type conversion.

Module 8: Plan Caching and Recompilation

Lab: Plan Caching and Recompilation

Exercise 1: Troubleshooting with the Plan Cache

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running.
2. Log on to 10987C-MIA-SQL as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
3. In the **D:\Labfiles\Lab08\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
4. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Start the Workload

1. Open Windows Explorer and browse to **D:\Labfiles\Lab08\Starter**.
2. Right-click **start_load_exercise_01.ps1**, and then click **Run with PowerShell**.
3. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press ENTER.
   Once the workload script is running, continue with the exercise. Do not wait for it to finish.

▶ Task 3: Check for Plan Cache Bloat

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, open the project **D:\Labfiles\Lab08\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, double-click **Lab Exercise 01 - plan cache.sql**. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**.)
5. Select the query under the comment that begins **Task 2**, and then click **Execute**.

   Plan cache bloat is occurring; a single query hash is linked to hundreds of cached plans. Queries that are identical, other than literal values, are assigned the same query hash.

▶ Task 4: Identify the Query Causing Plan Cache Bloat

1. In SQL Server Management Studio, in the query pane, under the comment that begins **Task 3**, edit the query:

```
SELECT TOP(1) [text]
FROM sys.dm_exec_query_stats AS qs
CROSS APPLY sys.dm_exec_sql_text(qs.plan_handle) AS st
WHERE query_hash = <query hash from task 1>
```

   Replace the text "<query hash from task 1>" with the value of the **query_hash** column returned from task 2.

2. Select the query you have amended and click **Execute**.

▶ Task 5: Identify the Stored Procedure Causing Plan Cache Bloat

1. Select the query under the comment that begins **Task 4**, and click **Execute.**
2. The procedure **Proseware.up_CampaignReport** is the only candidate procedure identified by your query.
3. **Proseware.up_CampaignReport** uses a dynamic SQL query; this is the cause of plan cache bloat, because each execution of the dynamic SQL query gets its own query execution plan added to the cache. In this case, the dynamic SQL query is unnecessary and can be removed.

▶ Task 6: Rewrite Proseware.up_CampaignReport to Prevent Plan Cache Bloat

1. In Solution Explorer, double-click the query **Lab Exercise 01a -Proseware.up_CampaignReport.sql**. (If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**.)
2. Amend the stored procedure definition in the file so that it reads:

```
ALTER PROCEDURE Proseware.up_CampaignReport
(@CampaignName varchar(20))
AS
   SELECT  cn.CampaignID,
           cn.CampaignName,
   cn.CampaignStartDate,
   cn.CampaignEndDate,
   st.Name,
   cr.ResponseDate,
   cr.ConvertedToSale,
   cr.ConvertedSaleValueUSD
   FROM Proseware.Campaign AS cn
   JOIN Sales.SalesTerritory AS st
   ON st.TerritoryID = cn.CampaignTerritoryID
   JOIN Proseware.CampaignResponse AS cr
   ON cr.CampaignID = cn.CampaignID
   WHERE CampaignName = @CampaignName;
GO
```

3. Click **Execute.**

▶ Task 7: Verify That the Stored Procedure Is Using a Single Query Plan

1. In the **Lab Exercise 01 - plan cache.sql** pane, select the query under the comment that begins **Task 6** and click **Execute**.
2. Notice that only one row is returned by the query; this indicates that the stored procedure is using only one query plan.

▶ Task 8: Stop the Workload

1. In the query pane, highlight the code under the comment that begins **Task 7** and click **Execute**. This will stop the workload.
2. Press ENTER in the PowerShell workload window to close it.
3. Leave SQL Server Management Studio open for the next exercise.

Results: At the end of this exercise, you will have refactored a stored procedure to reduce plan cache bloat.

Exercise 2: Working with the Query Store

▶ Task 1: Start the Workload

1. Open Windows Explorer and browse to **D:\Labfiles\Lab08\Starter**.
2. Right-click **start_load_exercise_02.ps1**, and then click **Run with PowerShell**. Allow this to run for a few minutes before continuing.

▶ Task 2: Enable the Query Store

1. In SQL Server Management Studio, in the Object Explorer pane, expand **Databases**, right-click **ProseWare**, and then click **Properties**.
2. In the **Database Properties - ProseWare** dialog box, on the **Query Store** page, change the value of the **Operation Mode (Requested)** property to **Read Write**, and then click **OK**.

▶ Task 3: Amend the Query Store Statistics Collection Interval

1. In SQL Server Management Studio, in the Object Explorer pane, expand **Databases**, right-click **ProseWare**, and then click **Properties**.
2. In the **Database Properties - ProseWare** dialog box, on the **Query Store** page, change the value of the **Statistics Collection Interval** property to **1 minute**, and then click **OK**.

▶ Task 4: Check the Top Resource Consuming Queries Report

1. In Object Explorer, expand **ProseWare**, and then expand **Query Store**.
2. Double-click **Top Resource Consuming Queries**.
3. Hover over the largest bar in the histogram in the upper left of the Top Resource Consumers window, and note the value of **query id**. This is the **query id** of the most expensive query.

▶ Task 5: Add a Missing Index

1. In Solution Explorer, double-click **Lab Exercise 02 - Query Store.sql**.
2. Highlight the code under the comment that begins **task 5**, and click **Execute**.

▶ Task 6: Force a Query Plan

1. In Object Explorer, double-click **Tracked Queries**.
2. In the Tracked Queries pane, in the **Tracking Query** box, type the query id you noted in an earlier task, then press ENTER.
3. In the graph in the upper half of the Tracked Queries pane, click on either of the two points.
4. On the toolbar, click **Force Plan**, and then in the **Confirmation** dialog box, click **Yes**.

▶ Task 7: Stop the Workload

1. Return to the query window where **Lab Exercise 02 - Query Store.sql** is open.
2. In the query pane, highlight the code under the comment that begins **Task 7** and click **Execute**. This will stop the workload.
3. Close SQL Server Management Studio without saving any changes.
4. Close the PowerShell workload window.

Results: At the end of this exercise, you will be able to:

Configure the Query Store.

Use the Query Store to investigate statement query execution plans.

Use the Query Store to force a query execution plan.

Module 9: Extended Events

Lab: Extended Events

Exercise 1: Using the system_health Extended Events Session

▶ Task 1: Prepare the Lab Environment

1. Ensure that the **MT17B-WS2016-NAT**, **10987C-MIA-DC**, and **10987C-MIA-SQL** virtual machines are running, and then log on to **10987C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab09\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

▶ Task 2: Run a Workload

1. Open File Explorer and navigate to the **D:\Labfiles\Lab09\Starter** folder.
2. Right-click **start_load_1.ps1** and then click **Run with PowerShell**.
3. If a message is displayed asking you to confirm a change in execution policy, type **Y**, and then press **ENTER**.
4. Wait for the workload to complete—this should take about a minute—and then press **ENTER** to close the Windows PowerShell window.

▶ Task 3: Query the system_health Extended Events Session

1. Start **SQL Server Management Studio** and connect to the **MIA-SQL** database engine using Windows authentication.
2. In SQL Server Management Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab09\Starter\Project** folder, click **Project.ssmssln**, and then click **Open**.
4. In Solution Explorer, double-click **Exercise 01 - system_health.sql**.
5. Edit the code under the comment that begins **-- Task 2** so that it reads as follows:

```
SELECT CAST(event_data AS xml) AS xe_data
FROM sys.fn_xe_file_target_read_file('system_health*.xel', NULL, NULL, NULL);
```

6. Select the query that you edited in the previous step, and then click **Execute**.

▶ Task 4: Extract Deadlock Data

1. Edit the query under the comment that begins **-- Task 3** so that it reads as follows:

```
SELECT xe_event.c.value('@timestamp', 'datetime2(3)') AS event_time,
xe_event.c.query('/event/data/value/deadlock') AS deadlock_data
FROM
(
   SELECT CAST(event_data AS xml) AS xe_data
   FROM sys.fn_xe_file_target_read_file('system_health*.xel', NULL, NULL, NULL)
) AS xe_data
CROSS APPLY xe_data.nodes('/event') AS xe_event(c)
WHERE xe_event.c.value('@name', 'varchar(100)') = 'xml_deadlock_report'
ORDER BY event_time;
```

2. Select the query that you edited in the previous step, and then click **Execute**.
3. In the Results pane, click on any of the row values in the **deadlock_data** column to view the deadlock XML in detail.
4. Leave SSMS open for the next exercise.

Results: After completing this exercise, you will have extracted deadlock data from the SQL Server.

Exercise 2: Tracking Page Splits Using Extended Events

▶ Task 1: Create an Extended Events Session to Track Page Splits

1. In Solution Explorer, double-click **Exercise 02 – page splits.sql**.
2. In Object Explorer, under **MIA-SQL (SQL Server 13.0.1000 - ADVENTUREWORKS\Student)**, expand **Management**, expand **Extended Events**, right-click **Sessions**, and then click **New Session Wizard**.
3. In the **New Session Wizard**, on the **Introduction** page, click **Next**.
4. On the **Set Session Properties** page, in the **Session name** box, type **track page splits**, and then click **Next**.
5. On the **Choose Template** page, click **Next**.
6. On the **Select Events To Capture** page, in the **Event library** section, click the drop-down button in the **Channel** column header (you may have to scroll right), then select **Debug**. In the first **Search Events** box, type **transaction_log,** double-click the **transaction_log** row in the **Event Library** list, which will add it to the **Selected events** list, and then click **Next**.
7. On the **Capture Global Fields** page, click **Next**.
8. On the **Set Session Event Filters** page, click **Click here to add a clause**. In the **Field** drop-down list, click **sqlserver.database_name**, in the **Value** box, type **AdventureWorks**, and then click **Finish**.
9. On the New Session Wizard: **Create Event Session** page, click **Close.**
10. In Object Explorer, expand **Sessions**, right-click **track page splits**, and then click **Properties**.
11. In the **Session Properties** dialog box, on the **Events** page, click **Configure**.
12. In the **Selected events** list, click **transaction_log**, on the **Filter (Predicate)** tab, click **Click here to add a clause**. Ensure that the value of the **And/Or** box is **And**, in the **Field** list, click **operation**, in the **Operator** list, click **=**, and then in the **Value** list, click **LOP_DELETE_SPLIT**.
13. On the **Data Storage** page, click **Click here to add a target**. In the **Type** list, click **histogram**.
14. In the **Event to filter on** list, click **transaction_log**, in the **Base buckets on** section, click **Field**, in the **Field** list, click **alloc_unit_id**, and then click **OK**.
15. In Object Explorer, right-click **track page splits** and click **Start session**.

▶ Task 2: Run a Workload

1. In File Explorer, navigate to the **D:\Labfiles\Lab09\Starter** folder.
2. Right-click **start_load_2.ps1**, and then click **Run with PowerShell**.
3. Wait for the workload to complete. This should take about 60 seconds.

▶ Task 3: Query the Session

1. In SQL Server Management Studio, in the query window for **Exercise 02 – page splits.sql**, edit the code under the comment that begins **-- Task 3** so that it reads as follows:

```
USE AdventureWorks;
GO
SELECT CAST(target_data AS XML) AS target_data
FROM sys.dm_xe_sessions AS xs
JOIN sys.dm_xe_session_targets xt
ON xs.address = xt.event_session_address
WHERE xs.name = 'track page splits'
AND xt.target_name = 'histogram';
```

2. Select the query that you edited in the previous step, and then click **Execute**.
3. In the results pane, click the returned XML to review the data.

▶ Task 4: Extract alloc_unit_id and Count Values

1. In the Exercise 02 – page splits.sql pane, edit the code under the comment that begins **-- Task 4** so that it reads as follows:

```
SELECT xe_node.value('(value)[1]', 'bigint') AS alloc_unit_id,
xe_node.value('(@count)[1]', 'bigint') AS split_count
FROM (     SELECT CAST(target_data AS XML) AS target_data
FROM sys.dm_xe_sessions AS xs
JOIN sys.dm_xe_session_targets xt
ON xs.address = xt.event_session_address
WHERE xs.name = 'track page splits'
AND xt.target_name = 'histogram')
AS xe_data
CROSS APPLY target_data.nodes('HistogramTarget/Slot') AS xe_xml (xe_node);
```

2. Select the query that you edited in the previous step, and then click **Execute**.
3. Review the number of splits in each node.

▶ Task 5: Return Object Names

1. Edit the code under the comment that begins **-- Task 5** so that it reads as follows:

```
SELECT OBJECT_SCHEMA_NAME(sp.object_id) AS object_schema,
          OBJECT_NAME(sp.object_id) AS object_name,
          si.name AS index_name,
          xe.split_count
FROM (    SELECT xe_node.value('(value)[1]', 'bigint') AS alloc_unit_id,
                         xe_node.value('(@count)[1]', 'bigint') AS split_count
          FROM (  SELECT CAST(target_data AS XML) AS target_data
                         FROM sys.dm_xe_sessions AS xs
                         JOIN sys.dm_xe_session_targets xt
                         ON xs.address = xt.event_session_address
                         WHERE xs.name = 'track page splits'
                         AND xt.target_name = 'histogram') AS xe_data
          CROSS APPLY target_data.nodes('HistogramTarget/Slot') AS xe_xml (xe_node))
AS xe
JOIN sys.allocation_units AS sau
ON sau.allocation_unit_id = xe.alloc_unit_id
JOIN sys.partitions AS sp
ON sp.partition_id = sau.container_id
JOIN sys.indexes AS si
ON si.object_id = sp.object_id
AND si.index_id = sp.index_id;
```

2. Select the query that you edited in the previous step, and then click **Execute**.
3. Review the objects affected by page splits.

▶ Task 6: Delete the Session

1. In Object Explorer, under **Sessions**, right-click **track page splits**, and click **Delete**.
2. In the **Delete Object** dialog box, click **OK**.
3. Close SSMS without saving changes.
4. In the Windows PowerShell window, press **ENTER** to close the window.

Results: After completing this exercise, you will have extracted page split data from SQL Server.

Module 10: Monitoring, Tracing, and Baselines

Lab: Monitoring, Tracing, and Baselining

Exercise 1: Collecting and Analyzing Data Using Extended Events

▶ Task 1: Prepare the Lab Environment

1. Ensure that the 10987C-MIA-DC and 10987C-MIA-SQL virtual machines are running, and then log on to the 10987C-MIA-SQL machine as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. In the **D:\Labfiles\Lab10\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. When you are prompted, click **Yes** to confirm that you want to run the command file, and then wait for the script to finish.

▶ Task 2: Set up an Extended Events Session

1. Start SQL Server Management Studio, and then connect to the **MIA-SQL** database engine instance by using Windows authentication.
2. On the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab10\Starter** folder, click **10987-10.ssmssln**, and then click **Open**.
4. In Solution Explorer, under **Queries**, double-click **SetupExtendedEvent.sql**, and then on the toolbar, click **Execute**.
5. In Object Explorer, expand **Management**, expand **Extended Events**, and then expand **Sessions**.
6. Right-click **AnalyzeSQLEE**, and then click **Watch Live Data**.

▶ Task 3: Execute Workload

1. In File Explorer, in the **D:\Labfiles\Lab10\Starter** folder, right-click **RunWorkload.cmd**, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. After execution of the workload completes, repeat steps 1 and 2.
4. In SQL Server Management Studio, on the **Extended Events** menu, click **Stop Data Feed**.
5. In the **AnalyzeSQLEE: Live Data** pane, right-click the **name** column heading, and then click **Choose Columns**.
6. In the **Choose Columns** dialog box, under **Available columns**, click **duration**, click **>**, click **query_hash**, click **>**, click **statement**, click **>**, and then click **OK**.

▶ Task 4: Analyze Collected Data

1. In the **AnalyzeSQLEE: Live Data** pane, right-click the **query_hash** column heading, and then click **Group by this Column**.
2. Right-click the **duration** column heading, point to **Calculate Aggregation**, and then click **AVG**.
3. Right-click the **duration** column heading, and then click **Sort Aggregation Descending**.
4. Expand one of the query hash rows to observe the top statements by duration.

5. In Solution Explorer, double-click **cleanup.sql**, and then click **Execute** to remove the Extended Event.
6. Leave SQL Server Management Studio open for the next exercise.

Results: After completing this exercise, you will have set up an Extended Events session that collects performance data for a workload and analyzed the data.

Exercise 2: Implementing Baseline Methodology

▶ Task 1: Set up Data Collection Scripts

1. In SQL Server Management Studio, in Solution Explorer, double-click **PrepareScript.sql**.
2. Examine the contents of the script, and then click **Execute**. The error can be ignored as this just means the database has already been removed.

▶ Task 2: Execute Workload

1. In Solution Explorer, double-click **WaitsCollectorJob.sql**, and then click **Execute**.
2. In Object Explorer, expand **SQL Server Agent**, expand **Jobs**, right-click **waits_collections**, and then click **Start Job at Step**.
3. Wait for the job to complete, and then click **Close**.
4. In File Explorer, navigate to the **D:\Labfiles\Lab10\Starter** folder, right-click **RunWorkload.cmd**, and then click **Run as administrator**.
5. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.
6. In SQL Server Management Studio, in **Jobs**, right-click **waits_collections**, and then click **Start Job at Step**.
7. Wait for the job to complete, and then click **Close**.

▶ Task 3: Analyze Data

1. In Solution Explorer, double-click **WaitBaselineDelta.sql**, and then click **Execute**.
2. In Solution Explorer, double-click **WaitBaselinePercentage.sql**, and then click **Execute**.
3. In Solution Explorer, double-click **WaitBaselineTop10.sql**, and then click **Execute**.
4. In the Results pane, observe the top 10 waits that were collected during the execution of the workload.
5. Close SQL Server Management Studio without saving any changes.
6. Close File Explorer.

Results: After completing this exercise, you will have implemented a baseline for a workload.