Microsoft® Technology Courses

Introduction

Microsoft .NET Framework
Microsoft Visual Studio
C# Programming

Microsoft

Hello

- Instructor: Ranjan Bhatnagar
- Microsoft Certified Trainer
- Freelancer
- Client-driven training professional offering 20+ years of progressive experience in training.
- More than 30 corporates & 30000+ Trainees
- Known for outstanding ability to create dynamic, eye-opening, highly interactive, and fun educational experiences for clients that exceed their highest expectations.

Day6 – Session 1

Theoretical Background
SQL Server Architecture
Creating Tables
Defining Data Integrity

Evolution of Database System

- Data is important to us and must be stored so that we can retrieve it latter when we need it.
- There are two ways to store data in computer system:
  - Using file system
  - Using database
- Note here we are talking about how data is organized when stored.

9

Disadvantages with file

- For data entry and retrieval large special programs need to be written.
- Programs must make sure that there is no redundant data.
- Programs must ensure that data is consistent when different user manipulates the same data concurrently or if system crashes.
- Retrieval of particular data from a large volume will require complex coding.
- Providing security mechanism for different subset of data again will require complex design.
- Different users in different system may store the data on different files. This leads to data isolation or islands of data. Writing programs to merge the data, retrieve data etc. becomes extremely difficult.
- No way to check the integrity of data. That is no way to check if some inappropriate values (like negative salary) is entered in the file.

10

Database system to rescue!

- Database is a collection of meaningful and related data.
- The software that manages this data is database management system (DBMS) or database system.
- For a given database there is a structural description of the type of facts held in the database which is called Schema. The schema describes the items or objects that represent the database. Based on the schema there are different database models… but before that …

11

Advantages of database system

- Reduced application development time
- Easy storage and retrieval
- Data integrity
- Controlled Redundancy
- Data Consistency
- Security
- Transaction and Concurrency
- Crash Recovery
- Administration

12

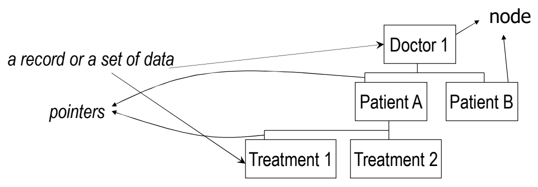
Data Models

- Hierarchical Model
- Network Model
- Relational Model
- Object Model

13

Hierarchical Model

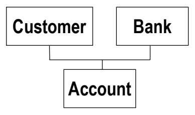
- Data is organized into tree-like structure.
- Used by the older mainframe database systems like IMS by IBM.
- Parent-child relationship between the data. 1:N mapping.



14

Network Model

- Network model is very much like hierarchical model with the exception that it allows a record to have more than one parent
- N:M mapping (many to many relationship)



A customer has accounts in many banks.
A bank has many customers who have accounts.

15

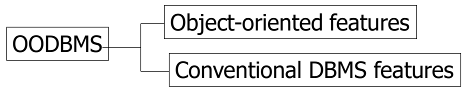
Relational Model

- Most database system use this model.
- The schema for relational model is a table (relation) which has a name, attributes or column or field and a type for each field.
- Relationship is maintained by the common attributes in the tables. Can be one-one or one to many or many to many.
- An example of representation of a relation Student is:
  - Student(studid: integer, name: string)
  - Marks(studid: integer, semester:integer, marks:double)

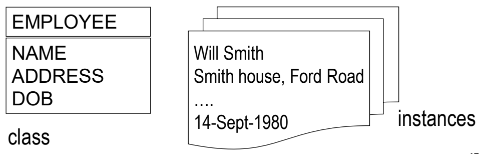
16

Object-Oriented database model

- The new wave of database technologies use Object-oriented data model (OODM).



- Object schema representation

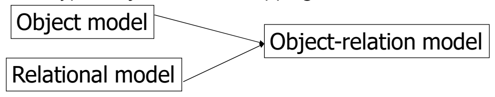


17

Presented by Ranjan Bhatnagar

ORDBMS

- An object-relation database is a relational database that allows developers to integrate their own custom data and types: object-relation mapping.



18

What is RDBMS?

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

19

Normalization

- First proposed by Dr. Codd.
- Normalization is the process of breaking down a relation schema into smaller relations in order to achieve good database design.
- To achieve good design, several step by step decomposition rules are laid out in the form of 'NORMAL FORMS'.
- There are 6 Normal forms.
- For most relation schema, good design is attained at 3 NF itself.
- While normalizing we also pay attention to two important aspects:
- Lossless Join: There should be no loss of information when tables are split.
- Dependency Preservation: The dependency between the attributes must be preserved.

20

1st and 2nd NF

- **First Normal Form**
- There are no duplicated rows in the table.
- Each cell is single-valued (i.e., there are no repeating groups or arrays).
- Entries in a column (attribute, field) are of the same kind
- **Second Normal Form:**
- A table is in 2NF if it is in 1NF and if all non-key attributes are fully functionally dependent on all of the key attributes (or prime Attribute).
- Key attribute or Prime attribute is an attribute that is a part of any candidate key.
- Functional Dependency: Let A and B be two attributes or set of attributes. B is functionally dependent on A if B's value is determined by A's value.
- A is the determinant.

21

3 NF

- **Third Normal Form**
- A table is in 3NF if it is in 2NF and if it has no transitive dependencies of a non-key attribute on the primary key.
- Transitive dependencies (A→B, B→C):
- Let A, B and C be three attributes or set of attributes. If B is functionally dependent on A and C functionally dependent on B (in other words C is indirectly dependent on A) then C is transitively dependent on A.
- Let us look at the Student relation that resulted from 2NF
- Student (StudID, Student Name, Course No, Course Name)
- Transitive dependency:
- StudID→ Course No
- Course No→ Course Name
- Both the attributes are non-key

22

Data integrity

- Data integrity refers to complete, current, accurate and valid data in columns of the table in the database.
- Referential integrity ensures the right relationships between tables. For a relationship to be valid, data in any column of a table should point to existing rows in another table.
- SQL server database objects that is used to maintain data integrity
  - Constraints
  - Rules
  - Defaults
  - DML Triggers

23

Presented by Ranjan Bhatnagar

What Is a Database?

- The term database is used in many different ways, but for our purposes a database is a collection of data stored in some organized fashion.
- The simplest way to think of it is to imagine a database as a filing cabinet.
- The filing cabinet is simply a physical location to store data, regardless of what that data is or how it is organized.
- A container (usually a file or set of files) to store organized data.

24

Tables

- A structured list of data of a specific type.
- When you store information in your filing cabinet, you don't just toss it in a drawer.
- Rather, you create files within the filing cabinet, and then you file related data in specific files.
- In the database world, that file is called a table.
- A table is a structured file that can store data of a specific type.
- A table might contain a list of customers, a product catalog, or any other list of information.

25

Table Names

- Every table in a database has a name that identifies it.
- That name is always unique, meaning no other table in that database can have the same name.
- What makes a table name unique is actually a combination of several things, including the database name and table name.
- This means that although you cannot use the same table name twice in the same database, you definitely can reuse table names in different databases.

26

Schema

- Tables have characteristics and properties that define how data is stored in them.
- These include information about what data may be stored, how it is broken up, how individual pieces of information are named, and much more.
- This set of information that describes a table is known as a schema, and schemas are used to describe specific tables within a database, as well as entire databases and the relationship between tables in them, if any.
- Information about database and table layout and properties.

27

Columns

- All tables are made up of one or more columns.
- A column contains a particular piece of information within a table.
- Aka A single field in a table.
- The best way to understand this is to envision database tables as grids, somewhat like spreadsheets.
- Each column in the grid contains a particular piece of information.
- In a customer table, for example, one column contains the customer number, another contains the customer name, and the address, city, state, and ZIP Code are all stored in their own columns.

28

Datatypes

- Every table column has an associated datatype that restricts (or allows) specific data in that column.
- Each column in a database has an associated datatype.
- A datatype defines what type of data the column can contain.
- For example, if the column is to contain a number the datatype would be numeric.
- If the column were to contain dates, text, notes, currency amounts, and so on, the appropriate datatype would be used to specify this.
- Datatypes restrict the type of data that can be stored in a column.
- Datatypes also help sort data correctly, and they play an important role in optimizing disk usage.

29

Rows

- Data in a table is stored in rows; each record saved is stored in its own row.
- Again, envisioning a table as a spreadsheet-style grid, the vertical columns in the grid are the table columns, and the horizontal rows are the table rows.
- For example, a customers table might store one customer per row.
- The number of rows in the table is the number of records in it.
- You might hear users refer to database records when referring to rows. For the most part, the two terms are used interchangeably, but row is technically the correct term.

30

Constraints

- Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
- Constraints could be column level or table level. Column level constraints are applied only to one column where as table level constraints are applied to the whole table.
- Following are commonly used constraints available in SQL:
  - Primary Key
  - Unique Key
  - Not Null
  - Foreign Key
  - Check Constraint
  - Default

31

Constrains Types

- PRIMARY Key: Uniquely identified each rows/records in a database table.
- UNIQUE Constraint: Ensures that all values in a column are different.
- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- FOREIGN Key: Uniquely identified a rows/records in any another database table.
- CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
- DEFAULT Constraint: Provides a default value for a column when none is specified.

32

Always Define Primary Keys

- Although primary keys are not actually required, most database designers ensure that every table they create has a primary key so future data manipulation is possible and manageable.
- Primary keys are usually defined on a single column within a table.
- But multiple columns may be used together as a primary key.
- When multiple columns are used the values of all columns together must be unique (individual columns need not have unique values).

33

Primary Key Best Practices

- In addition to the rules that SQL Server enforces, several universally accepted best practices should be adhered to:
  - Don't update values in primary key columns.
  - Don't reuse values in primary key columns.
  - Don't use values that might change in primary key columns.
    - For example, when you use a name as a primary key to identify a supplier, you would have to change the primary key when the supplier merges and changes its name.

34

What Is SQL?

- SQL (pronounced as the letters S-Q-L or as sequel) is an abbreviation for Structured Query Language.
- SQL is a language designed specifically for communicating with databases.
- Unlike other languages (spoken languages, such as English, or programming languages, such as Java or Visual Basic), SQL is made up of very few words.
- SQL is designed to do one thing and do it well: provide you with a simple and efficient way to read and write data from a database.

35

Presented by
Ranjan Bhatnagar

What are the advantages of SQL?

- SQL is not a proprietary language used by specific database vendors.
- Almost every major DBMS supports SQL, so learning this one language enables you to interact with just about every database you'll run into.
- SQL is easy to learn. The statements are all made up of descriptive English words, and there aren't that many of them.
- Despite its apparent simplicity, SQL is actually a very powerful language, and by cleverly using its language elements you can perform very complex and sophisticated database operations.

36

Introducing SQL Server

- SQL Server is the foundation of Microsoft's data platform, delivering mission-critical performance with in-memory technologies and faster insights on any data, whether on-premises or in the cloud.
- It is the database software (DBMS or Database Management System) that actually does all the work of storing, retrieving, managing, and manipulating data.
- SQL Server is a DBMS; that is, it is database software.
- SQL Server has been around for a long time and is in use at millions of installations worldwide.

37

Reasons to Use SQL Server

- **Performance:** SQL Server is fast (make that very fast).
- **Trusted:** SQL Server is used by some of the most important and prestigious organizations and sites, all of whom entrust it with their critical data.
- **Integration:** SQL Server is tightly integrated with other Microsoft offerings.
- **Simplicity:** SQL Server is one of the easiest DBMSs to install and get up and running, and includes administrative tools that make management of the server painless and simple.

38

Client Server Software

- DBMSs fall into two categories:
  - shared file–based
  - and client/server.
- The former (which include products such as Microsoft Access and FileMaker) are designed for desktop use and are generally not intended for use on higher-end or more critical applications.
- Databases such as SQL Server, Oracle, and MySQL are client/server–based databases.
- The client and server software may be installed on two computers or on one computer. Regardless, the client software communicates with the server software for all database interaction, be it on the same machine or not.

39

Server

- Client/server applications are split into two distinct parts.
  - The server portion is a piece of software that is responsible for all data access and manipulation.
  - This software runs on a computer called the database server.
  - Only the server software interacts with the data files.
  - All requests for data, data additions and deletions, and data updates are funneled through the server software.
  - These requests or changes come from computers running client software.

40

Client

- The client is the piece of software with which the user interacts.
- If you request an alphabetical list of products, for example, the client software submits that request over the network to the server software.
- The server software processes the request; filters, discards, and sorts data as necessary; and sends the results back to your client software.
- All this action occurs transparently to you, the user.
- The fact that data is stored elsewhere or that a database server is even performing all this processing for you is hidden.
- You never need to access the data files directly.

41

Why no access to the data?

- In fact, most networks are set up so that users have no access to the data, or even the drives on which it is stored. Why is this significant?
- Because to work with SQL Server, you'll need access to both a computer running the SQL Server software and client software with which to issue commands to SQL Server:
  - The server software is the SQL Server DBMS. You can run a locally installed copy, or you can connect to a copy running on a remote server to which you have access.
  - The client can be SQL Server–included tools, scripting languages (such as Perl), web application development languages (such as ASP, ASP.NET, ColdFusion, JSP, and PHP), programming languages (such as Visual Basic, VB.NET, C, C++, C#, and Java), and more.

42

SQL Server Versions

- The history of Microsoft SQL Server begins with the first Microsoft SQL Server product - SQL Server 1.0, a 16-bit server for the OS/2 operating system in 1989 - and extends to the current day.
- As of December 2016 the following versions are supported by Microsoft:
  - SQL Server 2008
  - SQL Server 2008 R2
  - SQL Server 2012
  - SQL Server 2014
  - SQL Server 2016
- The current version is Microsoft SQL Server 2016, released June 1, 2016.

43

SQL Server Tools

- SQL Server is a client/server DBMS, so to use SQL Server you'll need a client.
- There are lots of client application options, but when learning SQL Server (and indeed, when writing and testing SQL Server scripts) you are best off using a utility designed for simple script execution.
- The ideal tool depends on the version of SQL Server being used.
- SQL Server features a sophisticated client tool called **Microsoft SQL Server Management Studio**.
- This tool can be used to create and manage databases and tables, control database access and security, run wizards to optimize and fine-tune DBMS performance, and, of course, run SQL statements.

44

How to Use MSSMS

- There are many ways to use Microsoft SQL Server Management Studio, but here are the basic steps needed to enter and test SQL statements:
- The New Query button at the top left of the screen opens a window where SQL statements are entered.
- As T-SQL statements are typed, Microsoft SQL Server Management Studio automatically color-codes the statements and text (this is an invaluable troubleshooting tool because it lets you quickly spot typos or missing quotes and so on).
- To execute (run) a statement, click the Execute button (the one with the red exclamation point on it). You can also press F5 or Ctrl+E to execute a statement.

45

Cont…

- To verify that a SQL statement is syntactically correct (without executing it), click the Parse button (the one with the blue check mark on it).
- Microsoft SQL Server Management Studio displays statement results at the bottom of the screen.
- Results may be displayed in a grid (the default behavior), as plain text, or saved to a file.
- You can switch between these modes by clicking the appropriate toolbar buttons.
- In addition to displaying statement results, Microsoft SQL Server Management Studio also displays status messages (the number of rows returned, for example) in a second tab labeled Messages.

46

Executing Saved Scripts

- Microsoft SQL Server Management Studio can also be used to execute saved scripts
- SQL statements can be saved in files, such as the table-creation command script or it could be any query to populate data from table.
- In fact, all scripts are saved using plain-text format.
- You can save scripts with .SQL extension.
- You can load these script files and run them in Microsoft SQL SQL Server Management Studio.

47

Presented by
Ranjan Bhatnagar

Working with SQL Server

- Making the Connection
  - SQL Server, like all client/server DBMSs, requires that you log into the DBMS before being able to issue commands.
  - SQL Server can authenticate users and logins using its own user list (SQL Server Authentication), or using the Windows user list i.e the logins used to start using Windows (Windows Authentication).
  - To connect to SQL Server, you need the following pieces of information:
    - The Hostname (Computer/Server)
    - A valid User Name
    - A Password (If Required)

48

Selecting a Database

- When you first connect to SQL Server, a default database is opened for you.
- This will usually be a database named master
- Which as a rule you should never play with
- Before you perform any database operations, you need to select the appropriate database.
- To do this, you use the **USE** keyword as

  USE MyDatabase;
- The USE statement does not return any results.
- In SQL Server Management Studio you may select a database from the drop-down list in the toolbar to use it.

49

T-SQL

- Transact-SQL or T-SQL, the Microsoft implementation of the SQL standard. Transact-SQL is central to using SQL Server.
- All applications that communicate with an instance of SQL Server do so by sending Transact-SQL statements to the server, regardless of the user interface of the application.
- New users of databases will usually find it easier to work with SQL Server by using SQL Server Management Studio, instead of writing Transact-SQL statements.

50

What we will Learn?

- Here we will learn,
  - How to create a database
  - Create a table in the database
  - Insert data into the table
  - Update the data
  - Read the data
  - Delete the data and then delete the table.
  - We will create views and stored procedures and configure a user to the database and the data.

51

Create Database

- The CREATE DATABASE statement has a required parameter: the name of the database.
- CREATE DATABASE also has many optional parameters, such as the disk location where you want to put the database files.
- When you execute CREATE DATABASE without the optional parameters, SQL Server uses default values for many of these parameters.
- In a Query Editor window type

  CREATE DATABASE Sample
- press F5 to execute the statement and create a database named Sample

52

Creating a Table

- T-SQL can be used to perform all database and table operations, including the creation and manipulation of tables
- There are generally two ways to create database tables:
  - Using an administration tool to create and manage database tables interactively
  - Using T-SQL statements to manipulate tables directly
- To create tables programmatically, you use the CREATE TABLE SQL statement.
- When you use interactive tools, you are actually using T-SQL statements.

53

Create Table Command

- To create a table, you must provide a name for the table, and the names and data types of each column in the table.
- It is also a good practice to indicate whether null values are allowed in each column.
- To create a table, you must have the CREATE TABLE permission
- Enter the following code into a Query Editor window.

```
CREATE TABLE dbo.Products
(ProductID int PRIMARY KEY NOT NULL,
ProductName varchar(25) NOT NULL,
Price money NULL,
ProductDescription text NULL)
```

54

dbo

- SQL Server ships with few pre-defined schemas that have the same names as the built-in database users and roles.
- The schema is the database object that owns the table.
- The dbo schema is the default schema for a newly created database.
- The dbo schema is owned by the dbo user account.
- By default, users created with the CREATE USER Transact-SQL command have dbo as their default schema.
- If you are an administrator, dbo is the default schema. dbo stands for database owner.

55

T-SQL Datatypes

- Datatypes are basically rules that define what data may be stored in a column and how that data is actually stored.
- Datatypes are used for several reasons:
  - Datatypes enable you to restrict the type of data that can be stored in a column.
  - Datatypes allow for more efficient storage, internally.
  - Datatypes allow for alternate sorting orders.
- Using the wrong datatype can seriously impact your application.
- Here we will discuss the major T-SQL datatype types and what they are used for.

56

String Datatypes

- **CHAR:** Fixed-length string from 1 to 8,000 characters long. Its size must be specified at create time; otherwise, SQL Server assumes CHAR(1).
- **NCHAR:** Fixed-length Unicode string from 1 to 4,000 chars long. Its size must be specified at create time; otherwise, SQL Server assumes NCHAR(1).
- **NTEXT:** Variable-length Unicode text with a maximum size of 1,073,741,823 characters.
- **NVARCHAR:** Variable-length Unicode text with a maximum size of 4,000 characters.
- **TEXT:** Variable-length text with a maximum size of 2,147,483,647 characters.
- **VARCHAR:** Variable-length text with a maximum size of 8,000characters.

57

Numeric Datatypes

- **BIT:** A bit field, with a possible value of 0 or 1.
- **BIGINT:** Integer value. Supports numbers from -9,223,372,036,854,775,808 to -9,223,372,036,854,775,807.
- **DECIMAL (or DEC or NUMERIC):** Floating-point value with varying levels of precision.
- **FLOAT:** Variable-length byte floating-point value.
- **INT (or INTEGER):** Integer value. Supports numbers from -2,147,483,648 to 2,147,483,647.
- **MONEY:** Currency with four decimal places. Supports numbers from -922,337,203,685,477.5808 to 922,337,203,685,477.5807.
- **REAL:** Four-byte floating-point value.
- **SMALLINT:** Integer value. Supports numbers from -32,768 to 32,767.
- **SMALLMONEY:** Currency with four decimal places. Supports numbers from -214,748.3648 to 214,748.3647.
- **TINYINT:** Integer value. Supports numbers from 0 to 255.

58

Date and Time Datatypes

- SQL Server uses special datatypes for the storage of date and time values, as listed
- **TIME:** Defines a time of a day. Supports 00:00:00.0000000 through 23:59:59.9999999
- **DATE:** Defines a date in SQL Server. Supports 0001-01-01 through 9999-12-31
- **DATETIME:** Stores dates from January 1, 1753 through December 31, 9999.
- **SMALLDATETIME:** Stores dates from January 1, 1900 through June 6, 2079.

59

Presented by
Ranjan Bhatnagar

Binary Datatypes

- Binary datatypes are used to store all sorts of data (even binary information), such as graphic images, multimedia, and word processing documents
- **BINARY:** Fixed-length binary data of up to 8,000 characters.
- **VARBINARY:** Variable-length binary data of up to 8,000 characters.
- **VARBINARY(max):** Variable-length binary data exceeding 8,000 characters.

60

Other Datatypes

- In addition to the datatypes listed thus far, SQL Server supports several special-purpose datatypes
- **CURSOR:** Contains a reference to a cursor
- **TABLE:** A temporary table
- **UNIQUEIDENTIFIER:** A unique identifier in 16-byte GUID format
- **XML:** Well-formed XML data

61

Another Example

```
CREATE TABLE customers
(
        cust_id INT NOT NULL IDENTITY(1,1),
        cust_name NCHAR(50) NOT NULL ,
        cust_address NCHAR(50) NULL ,
        cust_city NCHAR(50) NULL ,
        cust_state NCHAR(5) NULL ,
        cust_zip NCHAR(10) NULL ,
        cust_country NCHAR(50) NULL ,
        cust_contact NCHAR(50) NULL ,
        cust_email NCHAR(255) NULL ,
        PRIMARY KEY (cust_id)
);
```

62

Understanding NULL

- Every table column is either a NULL column or a NOT NULL column, and that state is specified in the table definition at creation time.
- Don't confuse NULL values with empty strings.
- A NULL value is the lack of a value; it is not an empty string.
- If you were to specify '' (two single quotes with nothing in between them), that would be allowed in a NOT NULL column.
- An empty string is a valid value; it is not no value.
- NULL values are specified with the keyword NULL, not with an empty string.

63

Primary Keys Revisited

- A primary key values must be unique.
- That is, every row in a table must have a unique primary key value.
- If a single column is used for the primary key, it must be unique.
- If multiple columns are used, the combination of them must be unique.
- The CREATE TABLE examples seen thus far use a single column as the primary key.
- The primary key is thus defined using a statement such as PRIMARY KEY (vend_id)

64

Multiple Column Primary Key

- To create a primary key made up of multiple columns, simply specify the column names as a comma-delimited list, as shown in this example:

```
CREATE TABLE orderitems
(
     order_num INT NOT NULL ,
     order_item INT NOT NULL ,
     prod_id NCHAR(10) NOT NULL ,
     quantity INT NOT NULL ,
     item_price MONEY NOT NULL ,
     PRIMARY KEY (order_num, order_item)
);
```

65

Using IDENTITY

- Considering Customers and Orders tables -
  - Customers in the customers table are uniquely identified by column cust_id, a unique number for each and every customer.
  - orders in the orders table each have a unique order number stored in the column order_num.
- IDENTITY tells SQL Server that this column is to be automatically incremented each time a row is added.
- Each time an INSERT operation is performed, SQL Server automatically increments the column, assigning it the next available value.
- This way, each row is assigned a unique cust_id, which is then used as the primary key value.

66

More on Identity

- IDENTITY needs to know the number to start from known as the seed.
- The increment to be used each time a new value is generated.
- IDENTITY(1,1) means "start with a seed of 1 and increment by 1 to generate each new number."
- To start with a seed of 100 and increment by 10, you could use IDENTITY(100,10).
- If the seed and increment values are not provided, (1,1) is used by default.
- Only one IDENTITY column is allowed per table, and IDENTITY columns are usually used as primary keys.

67

Determining the IDENTITY Value

- One downside of having SQL Server generate (via IDENTITY) primary keys for you is that you don't know what those values are.
- Then how could you obtain this value when an IDENTITY column is used?
- By referring to the special @@IDENTITY system function, like this:

```
SELECT @@IDENTITY AS newId;
```

- This returns the last generated IDENTITY value as newId, which you can then use in subsequent T-SQL statements.

68

Specifying Default Values

- SQL Server enables you to specify default values to be used if no value is specified when a row is inserted.
- Default values are specified using the DEFAULT keyword in the column definitions in the CREATE TABLE statement. For Example:

```
CREATE TABLE orderitems
(
    order_num INT NOT NULL ,
    order_item INT NOT NULL ,
    prod_id NCHAR(10) NOT NULL ,
    quantity INT NOT NULL DEFAULT 1,
    item_price MONEY NOT NULL ,
    PRIMARY KEY (order_num, order_item)
);
```

69

T-SQL functions as default values

- The previous example used a fixed value as the specified DEFAULT.
- You can also use T-SQL functions as default values, as shown in this example:

```
CREATE TABLE orders
(
    order_num INT NOT NULL IDENTITY(1,1),
    order_date DATETIME NOT NULL DEFAULT GetDate(),
    cust_id INT NOT NULL ,
    PRIMARY KEY (order_num)
);
```

- The default value for this column is GetDate(), a T-SQL function that returns the current system date.

70

Updating Tables

- To update table definitions, you use the ALTER TABLE statement. But, ideally, tables should never be altered after they contain data.
- To change a table using ALTER TABLE, you must specify the following information:
  - The name of the table to be altered after the keywords ALTER TABLE.
  - The list of changes to be made.
- Example

```
ALTER TABLE vendors
ADD vend_phone CHAR(20);
```

71

Presented by Ranjan Bhatnagar

Updating Tables

- To remove this newly added column, you can do the following:

```
ALTER TABLE vendors
DROP COLUMN vend_phone;
```

- One common use for ALTER TABLE is to define foreign keys. The following is the code used to define the foreign keys.

```
ALTER TABLE orderitems
ADD CONSTRAINT fk_orderitems_orders
FOREIGN KEY (order_num)
REFERENCES orders (order_num);
```

72

Deleting Tables

- Deleting tables (actually removing the entire table, not just the contents) is very easy, arguably too easy.
- Tables are deleted using the DROP TABLE statement:

```
DROP TABLE customers2;
```

- Beware, above statement deletes the customers2 table (assuming it exists).
- There is no confirmation, nor is there an undo: Executing the statement will permanently remove the table.

73

Renaming Tables

- There is no T-SQL statement for renaming tables, but a SQL Server–provided stored procedure named sp_rename can be used to accomplish this:

```
EXEC sp_rename 'customers2', 'customers';
```

- The sp_rename can be used to rename all sorts of objects, including tables.
- The above example renames table customers2 to customers.

74

Inserting Data

- As its name suggests, INSERT is used to insert (add) rows to a database table.
- INSERT can be used in several ways:
  - To insert a single complete row
  - To insert a single partial row
  - To insert multiple rows
  - To insert the results of a query
- Use of the INSERT statement can be disabled per table or per user using SQL Server security
- Unlike some other DBMSs, SQL Server does not support multiple VALUES clauses for a single INSERT statement.

75

Inserting Complete Rows

- The simplest way to insert data into a table is to use the basic INSERT syntax, which requires that you specify the table name and the values to be inserted into the row.

```
INSERT INTO Customers VALUES(10006, 'Pep E. LaPew',
'100 Main Street',  'Los Angeles', 'CA', '90046', 'USA', NULL, NULL);
```

- INSERT statements usually generate no output.
- The data to be stored in each table column is specified in the VALUES clause, and a value must be provided for every column.
- If a column has no value (for example, the cust_contact and cust_email columns), the NULL value should be used.
- The columns must be populated in the order in which they appear in the table definition.

76

Manually Specifying Identity Field Values

- By default, identity fields do not allow you to explicitly specify values; SQL Server gets to do this itself.
- If you really do need to manually specify the value of an identity field, you can do so if you first issue the statement

```
SET IDENTITY_INSERT = ON.
```

- Assuming that the value specified is valid (and not already in use), subsequent inserts will auto-increment using this new value.

77

Better be Safe

- The safer (and, unfortunately, more cumbersome) way to write the INSERT statement is as follows:
  INSERT INTO customers(cust_name, cust_address, cust_city, cust_state, cust_zip, cust_country, cust_contact, cust_email) VALUES('Pep E. LaPew', '100 Main Street', 'Los Angeles', 'CA', '90046', 'USA', NULL, NULL);
- When the row is inserted, SQL Server will match each item in the columns list with the appropriate value in the VALUES list.
- The first entry in VALUES corresponds to the first specified column name. The second value corresponds to the second column name, and so on.

78

Advantages

- Because column names are provided, the VALUES must match the specified column names in the order in which they are specified, and not necessarily in the order the columns appear in the actual table.
- The advantage of this is that, even if the table layout changes, the INSERT statement will still work correctly.
- You'll notice that cust_id was not specified; unneeded columns can simply be omitted from both the columns list and the VALUES list.

79

Always Use a Columns List

- As a rule, never use INSERT without explicitly specifying the column list.
- This will greatly increase the probability that your SQL will continue to function in the event that table changes occur.
- Regardless of the INSERT syntax being used, the correct number of VALUES must be specified.
- If no column names are provided, a value must be present for every table column.
- If columns names are provided, a value must be present for each listed column.
- If none is present, an error message will be generated, and the row will not be inserted.

80

Omitting Columns

- You may omit columns from an INSERT operation if the table definition so allows.
- One of the following conditions must exist:
  - The column is defined as allowing NULL values (no value at all).
  - A default value is specified in the table definition. This means the default value will be used if no value is specified.
- If you omit a value from a table that does not allow NULL values and does not have a default, SQL Server generates an error message, and the row is not inserted.

81

Inserting Multiple Rows

- The basic INSERT statement only inserts a single row at a time, so you would need to use multiple INSERT statements.
- You could possibly submit them all at once, each terminated by a semicolon.

INSERT INTO customers(cust_name, cust_address, cust_city, cust_state, cust_zip, cust_country) VALUES('Pep E. LaPew', '100 Main Street', 'Los Angeles', 'CA', '90046', 'USA');
INSERT INTO customers(cust_name, cust_address, cust_city, cust_state, cust_zip, cust_country) VALUES('M. Martian', '42 Galaxy Way', 'New York', 'NY', '11213', 'USA');

- Unlike some other DBMSs, SQL Server does not support multiple VALUES clauses for a single INSERT statement.

82

Inserting Retrieved Data

- INSERT is generally used to add a row to a table using specified values.
- Another form of INSERT can be used to insert the result of a SELECT statement into a table.
- This is known as INSERT SELECT, and, as its name suggests, it is made up of an INSERT statement and a SELECT statement.

INSERT INTO customers(cust_contact, cust_email, cust_name, cust_address, cust_city, cust_state, cust_zip, cust_country) SELECT cust_contact, cust_email, cust_name, cust_address, cust_city, cust_state, cust_zip, cust_country FROM custnew;

- To try this example, create and populate the custnew table first.

83

Presented by Ranjan Bhatnagar

Caution

- The previous slide example imports data from a table named custnew into the customers table.
- The format of the custnew table should be the same as the customers table.
- When populating custnew, be sure not to use cust_id values that were already used in customers.
- The subsequent INSERT operation will fail if primary key values are duplicated.
- Alternatively, just omit that column and have SQL Server generate new values during the import process.

84

Column Names in INSERT SELECT

- This example uses the same column names in both the INSERT and SELECT statements for simplicity's sake.
- But there is no requirement that the column names match. In fact, SQL Server does not even pay attention to the column names returned by the SELECT.
- Rather, the column position is used, so the first column in the SELECT will be used to populate the first specified table column, and so on.
- This is very useful when you are importing data from tables that use different column names.

85

Another Way

- Another way to insert retrieved data is to use SELECT INTO. This variation to SELECT allows you to specify a destination table that will be populated with the results of a SELECT statement.

```
SELECT cust_contact, cust_email, cust_name, cust_address,
cust_city, cust_state, cust_zip, cust_country INTO customersExport
FROM customers;
```

- The SELECT statement used in an INSERT SELECT can include a WHERE clause to filter the data to be inserted.
- When used, the INTO clause must come after the column list and before the FROM clause. INTO specifies the name of a table to be created, and this table name must not already exist.
- SELECT INTO is very useful when you're trying to create a single table that contains rows retrieved from multiple tables.

86

Updating Data

- To update (modify) data in a table, you use the UPDATE statement.
- UPDATE can be used in two ways:
  - To update specific rows in a table
  - To update all rows in a table
- Special care must be exercised when using UPDATE because it is all too easy to mistakenly update every row in your table.
- Be sure to read this entire section on UPDATE before using this statement.

87

Update Command

- The basic format of an UPDATE statement is made up of three parts:
  - The table to be updated
  - The column names and their new values
  - The filter condition that determines which rows should be updated
- Customer 10005 now has an email address, and so his record needs updating. The following statement performs this update:

```
UPDATE customers SET cust_email = 'elmer@fudd.com'
WHERE cust_id = 10005;
```

88

More on Update

- The UPDATE statement always begins with the name of the table being updated. In this example, it is the **customers** table.
- The SET command is then used to assign the new value to a column. As used here, the SET clause sets the cust_email column to the specified value:

SET cust_email = 'elmer@fudd.com'

- The UPDATE statement finishes with a WHERE clause that tells SQL Server which row to update. Without a WHERE clause, SQL Server would update all the rows in the customers table with this new email address, definitely not the desired effect.

89

Updating Multiple Columns

- When updating multiple columns, you use only a single SET command and separate each column = value pair by a comma.
- In this example, the columns cust_name and cust_email will both be updated for customer 10005.

```
UPDATE customers
SET cust_name = ‘The Fudds’,
cust_email = ‘elmer@fudd.com’
WHERE cust_id = 10005;
```

- No comma is specified after the last column.

90

Using Subqueries in an UPDATE Statement

- Subqueries may be used in UPDATE statements, enabling you to update columns with data retrieved with a SELECT statement.
- To delete a column’s value, you can set it to NULL (assuming the table is defined to allow NULL values). You can do this as follows:

```
UPDATE customers
SET cust_email = NULL
WHERE cust_id = 10005;
```

- Here, the NULL keyword is used to save no value to the cust_email column.

91

Deleting Data

- To delete (remove) data from a table, you use the DELETE statement.
- DELETE can be used in two ways:
  - To delete specific rows from a table
  - To delete all rows from a table
- Special care must be exercised when using DELETE because it is all too easy to mistakenly delete every row from your table.
- The following statement deletes a single row from the customers table:

```
DELETE FROM customers WHERE cust_id = 10006;
```

92

More on Delete

- While using Delete Don’t Omit the WHERE Clause.
- DELETE takes no column names or wildcard characters.
- DELETE deletes entire rows, not columns.
- To delete specific columns, use an UPDATE statement
- The DELETE statement deletes rows from tables, even all rows from tables. But DELETE never deletes the table itself.

93

Best Practices

- Never execute an UPDATE or a DELETE without a WHERE clause, unless you really do intend to update and delete every row.
- Make sure every table has a primary key and use it as the WHERE clause whenever possible.
- Before you use a WHERE clause with an UPDATE or a DELETE, first test it with a SELECT to make sure it is filtering the right records; it is far too easy to write incorrect WHERE clauses.
- Use database-enforced referential integrity so that SQL Server will not allow the deletion of rows that have data in other tables related to them.

94

Retrieving Data with Select

- The SELECT statement is used to retrieve information from one or more tables.
- To use SELECT to retrieve table data, at a minimum you must specify two pieces of information:
  - What you want to select
  - From where you want to select it.
- A simple SQL SELECT statement

```
SELECT prod_name FROM products;
```

- Use above statement to retrieve a single column called prod_name from the products table.

95

Presented by Ranjan Bhatnagar

Unsorted Data

- The desired column name is specified right after the SELECT keyword.
- The FROM keyword specifies the name of the table from which to retrieve the data.
- If query results are not explicitly sorted, data will be returned in no order of any significance.
- A simple SELECT statement like the one just shown returns all the rows in a table.
- Data is not filtered (so as to retrieve a subset of the results), nor is it sorted.

96

White Space & Terminating Statements

- All extra white space within a SQL statement is ignored when that statement is processed.
- SQL statements can be specified on one long line or broken up over many lines.
- Most SQL developers find that breaking up statements over multiple lines makes them easier to read and debug.
- Multiple SQL statements must be separated by semicolons (the ; character).
- SQL Server (like most DBMSs) does not require that a semicolon be specified after single statements.
- Of course, you can always add a semicolon if you wish. It'll do no harm, even if it isn't needed.

97

SQL Statements and Case

- It is important to note that SQL statements are not case sensitive.
- Therefore, SELECT is the same as select, which is the same as Select.
- Many SQL developers find that using uppercase for all SQL keywords and lowercase for column and table names makes code easier to read and debug.
- As a best practice, pick a case convention and use it consistently.

98

Retrieving Multiple Columns

- To retrieve multiple columns from a table, the same SELECT statement is used.
- The only difference is that multiple column names must be specified after the SELECT keyword, and each column must be separated by a comma, but not after the last column name.
- The following SELECT statement retrieves three columns from the products table:

```
SELECT prod_id, prod_name, prod_price FROM products;
```

- SQL statements typically return raw, unformatted data.

99

Presentation of Data

- Data formatting is a presentation issue, not a retrieval issue.
- Therefore, presentation (for example, alignment and displaying price values as currency amounts with the currency symbol and commas) is typically specified in the application that displays the data.
- Actual raw retrieved data (without application-provided formatting) is rarely displayed as is.

100

Retrieving All Columns

- You can use SELECT statements to request all columns without having to list them individually.
- This is done using the asterisk (*) wildcard character in lieu of actual column names, as follows:

```
SELECT * FROM products;
```

- When a wildcard (*) is specified, all the columns in the table are returned.
- The columns are in the order in which they appear in the table definition.
- However, this cannot be relied on because changes to table schemas (adding and removing columns, for example) could cause ordering changes.

101

Using Wildcards

- As a rule, you are better off not using the * wildcard unless you really do need every column in the table.
- Even though use of wildcards might save you the time and effort needed to list the desired columns explicitly, retrieving unnecessary columns usually slows down the performance of your retrieval and your application.
- There is one big advantage to using wildcards. Because you do not explicitly specify column names, it is possible to retrieve columns whose names are unknown.

102

Retrieving Distinct Rows

- As you have seen, SELECT returns all matched rows. But what if you did not want every occurrence of every value? For example, suppose you want the vendor ID of all vendors with products in your products table:

  SELECT vend_id FROM products;
- Suppose the SELECT statement returned 15 rows (even though only four vendors are in that list) because there are 15 products listed in the products table.
- So how could you retrieve a list of distinct values?
- The solution is to use the DISTINCT keyword, which, as its name implies, instructs SQL Server to only return distinct values:

  SELECT DISTINCT vend_id FROM products;

103

Can't Be Partially DISTINCT

- The DISTINCT keyword must be placed directly in front of the column names.
- The DISTINCT keyword applies to all columns, not just the one it precedes.
- If you were to specify SELECT DISTINCT vend_id, prod_price, all rows would be retrieved unless both of the specified columns were distinct.
- So, DISTINCT works on rows retrieved not on the data stored.

104

Limiting Results

- SELECT statements return all matched rows, possibly every row in the specified table.
- To return just the first row or rows, use the TOP keyword as in example:

  SELECT TOP(5) prod_name FROM products;
- The above statement uses the SELECT statement to retrieve a single column. TOP(5) instructs SQL Server to return no more than five rows.
- You can also use TOP to get a percentage of rows by adding the keyword PERCENT. Here is an example:

  SELECT TOP(25) PERCENT prod_name FROM products;
- TOP(25) PERCENT instructs SQL Server to return the first 25% of rows in the products table.

105

Using Fully Qualified Table Names

- The SQL examples used thus far have referred to columns by just the column names.
- It is also possible to refer to columns using fully qualified names (using both the table and column names).

  SELECT products.prod_name FROM products;
- Table names, too, may be fully qualified, as shown here:

  SELECT products.prod_name FROM Inventory.dbo.products;
- A fully qualified table name is made up of a database name, the table owner name, and the table name.

106

Sorting Retrieved Data

- As you learnt, the SQL SELECT statement returns data appears to be displayed in no particular order at all.
- To explicitly sort data retrieved using a SELECT statement, you use the ORDER BY clause.
- ORDER BY takes the name of one or more columns by which to sort the output.

  SELECT prod_name FROM products ORDER BY prod_name;
- The ORDER BY clause instructing SQL Server to sort the data alphabetically by the prod_name column.

107

Presented by Ranjan Bhatnagar

Sorting by Multiple Columns

- It is often necessary to sort data by more than one column. For example, if you are displaying an employee list, you might want to display it sorted by last name and first name (first sort by last name, and then within each last name sort by first name). This would be useful if multiple employees have the same last name.
- To sort by multiple columns, simply specify the column names separated by commas.
- The following code retrieves three columns and sorts the results by two of them, first by price and then by name.

SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price, prod_name;

108

Specifying Sort Direction

- Data sorting is not limited to ascending sort orders (from A to Z).
- Although this is the default sort order, the ORDER BY clause can also be used to sort in descending order (from Z to A).
- To sort by descending order, you must specify the keyword DESC.
- The following example sorts the products by price in descending order.

SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price DESC;

109

Another Example

SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price DESC, prod_name;

- The DESC keyword only applies to the column name that directly precedes it.
- In the above example, DESC was specified for the prod_price column, but not for the prod_name column.
- Therefore, the prod_price column is sorted in descending order, but the prod_name column (within each price) is still sorted in standard ascending order.
- If you want to sort in descending order on multiple columns, be sure each column has its own DESC keyword.

110

ASC (Ascending)

- The opposite of DESC is ASC (for ascending), which may be specified to sort in ascending order.
- In practice, however, ASC is not generally used because ascending order is the default sequence (and is assumed if neither ASC nor DESC is specified).

111

ORDER BY and TOP

- Using a combination of ORDER BY and TOP, it is possible to find the highest or lowest value in a column.
- The following example demonstrates how to find the value of the most expensive item:

SELECT TOP(1) prod_price FROM products ORDER BY prod_price DESC;

- prod_price DESC ensures that rows are retrieved from most to least expensive, and TOP(1) tells SQL Server to just return one row.
- When specifying an ORDER BY clause, be sure that it is after the FROM clause. Using clauses out of order will generate an error message.

112

Filtering Data

- Database tables usually contain large amounts of data, and you seldom need to retrieve all the rows in a table.
- More often than not, you'll want to extract a subset of the table's data as needed for specific operations or reports.
- Retrieving just the data you want involves specifying search criteria, also known as a filter condition.
- Within a SELECT statement, you filter data by specifying search criteria in the WHERE clause.
- The WHERE clause is specified right after the table name (using the FROM clause)

113

Using the WHERE Clause

SELECT prod_name, prod_price FROM products
WHERE prod_price = 2.50;

- This statement retrieves two columns from the products table, but instead of returning all rows, it returns only rows with a prod_price value of 2.50.
- This example uses a simple equality test: It checks to see if a column has a specified value, and it filters the data accordingly.
- However, T-SQL enables you to do more than just test for equality.

114

The WHERE Clause Operators

- The first WHERE clause we looked at tests for equality, determining if a column contains a specific value. T-SQL supports a whole range of comparison operators, some of which are listed

Operator	Description
=	Equality
<>	Nonequality
!=	Nonequality
<	Less Than
<=	Less Than or Equal to
!<	Not Less Than
>	Greater Than
>=	Greater Than or Equal to
!>	Not Greater Than
BETWEEN	Between two specified values
IS NULL	Is a NULL value

115

Few Examples

- SELECT prod_name, prod_price FROM products WHERE prod_name = 'fuses';
- SELECT prod_name, prod_price FROM products WHERE prod_price < 10;
- SELECT prod_name, prod_price FROM products WHERE prod_price <= 10;
- SELECT vend_id, prod_name FROM products WHERE vend_id <> 1003;
- SELECT vend_id, prod_name FROM products WHERE vend_id != 1003;
- SELECT prod_name, prod_price FROM products WHERE prod_price BETWEEN 5 AND 10;
- SELECT prod_name FROM products WHERE prod_price IS NULL;
- SELECT cust_id FROM customers WHERE cust_email IS NULL;

116

Using the AND, OR Operator

- To filter by more than one column, you use the AND operator to append conditions to your WHERE clause.

  SELECT prod_id, prod_price, prod_name FROM products WHERE vend_id = 1003 AND prod_price <= 10;
- The OR operator instructs SQL Server to retrieve rows that match either condition, so if one matches and one does not, the row would still be retrieved.

  SELECT prod_name, prod_price FROM products WHERE vend_id = 1002 OR vend_id = 1003;

117

Understanding Order of Evaluation

- WHERE clauses can contain any number of AND and OR operators.
- However, combining AND and OR operators presents an interesting problem.
- To demonstrate this, let's look at an example. Suppose you need a list of all products costing 10 or more made by vendors 1002 and 1003.
- The following SELECT statement uses a combination of AND and OR operators to build a WHERE clause:

  SELECT prod_name, prod_price FROM products WHERE vend_id = 1002 OR vend_id = 1003 AND prod_price >= 10;

118

Analysis

- Look at the listed results.
- Two of the rows returned have prices less than 10; so, obviously, the rows were not filtered as intended.
- Why did this happen?

```
Output Asumed
prod_name            prod_price
----------------     ----------
Detonator            13.00
Bird seed            10.00
Fuses                3.42
Oil can              8.99
Safe                 50.00
TNT (5 sticks)       10.00
```

- The answer is the order of evaluation. T-SQL processes AND operators before OR operators.
- When SQL Server sees the preceding WHERE clause, it reads products made by vendor 1002 regardless of price, and any products costing 10 or more made by vendor 1003.
- In other words, because AND ranks higher in the order of evaluation, the wrong operators were joined together.

119

Presented by
Ranjan Bhatnagar

Solution

- The solution to this problem is to use parentheses to explicitly group related operators.
- Take a look at the following SELECT statement and output:

```
SELECT prod_name, prod_price FROM products WHERE
(vend_id = 1002 OR vend_id = 1003) AND prod_price >= 10;
```

```
       Output  Asumed
prod_name        prod_price
---------------- ----------
Detonator        13.00
Bird seed        10.00
Safe             50.00
TNT (5 sticks)   10.00
```

- Because parentheses have a higher order of evaluation than either AND or OR operators, SQL Server first filters the OR condition within those parentheses.

120

Using the IN Operator

- Parentheses have another very different use in WHERE clauses.
- The IN operator is used to specify a range of conditions, any of which can be matched.
- IN takes a comma-delimited list of valid values, all enclosed within parentheses.

```
SELECT prod_name, prod_price FROM products
WHERE vend_id IN (1002,1003) ORDER BY prod_name;
```

- The IN operator accomplishes the same goal as OR, same result can be obtained using

```
SELECT prod_name, prod_price FROM products WHERE
vend_id = 1002 OR vend_id = 1003 ORDER BY prod_name;
```

121

Why use the IN operator?

- When you are working with long lists of valid options, the IN operator syntax is far cleaner and easier to read.
- The order of evaluation is easier to manage when IN is used (because fewer operators are used).
- IN operators almost always execute more quickly than lists of OR operators.
- The biggest advantage of IN is that the IN operator can contain another SELECT statement, enabling you to build highly dynamic WHERE clauses.

122

Using the NOT Operator

- The WHERE clause's NOT operator has one function and one function only: NOT negates whatever condition comes next.
- The following example demonstrates the use of NOT. To list the products made by all vendors except vendors 1002 and 1003, you can use the following:

```
SELECT prod_name, prod_price
FROM products
WHERE vend_id NOT IN (1002,1003)
ORDER BY prod_name;
```

123

Using the LIKE Operator

- How could you search for all products that contain the text anvil within the product name?
- That cannot be done with simple comparison operators; that's a job for wildcard searching.
- Using wildcards, you can create search patterns that can be compared against your data.
- The wildcards themselves are actually characters that have special meanings within SQL WHERE clauses, and SQL supports several wildcard types.
- To use wildcards in search clauses, you must use the LIKE operator. LIKE instructs SQL Server that the following search pattern is to be compared using a wildcard match rather than a straight equality match.

124

The Percent Sign (%) Wildcard

- The most frequently used wildcard is the percent sign (%).
- Within a search string, % means match any number of occurrences of any character.
- For example, to find all products that start with the word jet, you can issue the following SELECT statement:

```
SELECT prod_id, prod_name
FROM products
WHERE prod_name LIKE 'jet%';
```

- The % tells SQL Server to accept any characters after the word jet, regardless of how many characters there are.

125

Presented by
Ranjan Bhatnagar

More on %

- Wildcards can also be used in the middle of a search pattern, although that is rarely useful.
- The following example finds all products that begin with an s and end with an e:
  SELECT prod_name FROM products
  WHERE prod_name LIKE 's%e';
- It is important to note that, in addition to matching one or more characters, % also matches zero characters.
- % represents zero, one, or more characters at the specified location in the search pattern.

126

The Underscore (_) Wildcard

- Another useful wildcard is the underscore (_).
- The underscore is used just like %, but instead of matching multiple characters, the underscore matches just a single character.
  SELECT prod_id, prod_name FROM products
  WHERE prod_name LIKE '_ ton anvil%';
- The search pattern matched a single character, not more but the following statement uses the % wildcard and returns matching products.
  SELECT prod_id, prod_name
  FROM products
  WHERE prod_name LIKE '% ton anvil%';

127

The Brackets ([]) Wildcard

- The brackets ([]) wildcard is used to specify a set of characters, any one of which must match a character in the specified position (the location of the wildcard).
- To find all contacts whose names begin with the letter J or the letter M, you can do the following:
  SELECT cust_contact FROM customers
  WHERE cust_contact LIKE '[EJ]%' ORDER BY cust_contact;
- This wildcard can be negated by prefixing the characters with ^.
- The following matches any contact name that does not begin with the letter E or the letter J.
  SELECT cust_contact FROM customers
  WHERE cust_contact LIKE '[^EJ]%' ORDER BY cust_contact;

128

Tips for Using Wildcards

- Wildcard searches typically take far longer to process than any other search types discussed previously.
- Here are some tips to keep in mind when using wildcards:
  - Don't overuse wildcards. If another search operator will do, use it instead.
  - When you do use wildcards, try not to use them at the beginning of the search pattern unless absolutely necessary. Search patterns that begin with wildcards are the slowest to process.
  - Pay careful attention to the placement of the wildcard symbols. If they are misplaced, you might not return the data you intended.

129

Creating Calculated Fields

- Data stored within a database's tables is often not available in the exact format needed by your applications. Here are some examples:
  - You need to display a field containing the name of a company along with the company's location, but that information is stored in separated table columns.
  - City, state, and ZIP Code are stored in separate columns (as they should be), but your mailing label printing program needs them retrieved as one correctly formatted field.
  - Column data is in mixed upper- and lowercase, and your report needs all data presented in uppercase.
  - An order items table stores item price and quantity but not the expanded price (price multiplied by quantity) of each item. To print invoices, you need that expanded price.
  - You need total, averages, or other calculations based on table data.

130

Concatenating Fields

- Joining values together (by appending them to each other) to form a single long value.
- To demonstrate working with calculated fields, let's start with a simple example, creating a title made up of two columns.
  SELECT vend_name + ' (' + vend_country + ')'
  FROM vendors ORDER BY vend_name;
- The + operator concatenates strings, appending them to each other to create one bigger string. The previous SELECT statements concatenate four elements.

131

Presented by
Ranjan Bhatnagar

Trimming

- The new calculated field contains extraneous spaces and is not exactly what we were looking for.
- There is a need to trim data so as to remove any trailing spaces.
- This can be done using the T-SQL RTrim() function, as follows:

  SELECT RTrim(vend_name) + ' (' + RTrim(vend_country) + ')' FROM vendors ORDER BY vend_name;
- The RTrim() function trims all spaces from the right of a value. By using RTrim(), you can trim the individual columns properly.

132

The LTrim() Function

- In addition to RTrim() (which, as you've just seen, trims the right side of a string), T-SQL also supports the use of LTrim() (which trims the left side of a string).
- To trim both the right and left sides of a string, use both functions, as in

  RTrim(LTrim(vend_name)).

133

Using Aliases

- The SELECT statement used to concatenate the address field works well, as shown in the previous example.
- But what is the name of this new calculated column?
- Well, the truth is, it has no name; it is simply a value.
- Although this can be fine if you are just looking at the results in a SQL query tool, an unnamed column cannot be used within a client application because the client has no way to refer to that column.
- To solve this problem, SQL supports column aliases. An alias is just that, an alternative name for a field or value. Aliases are assigned with the AS keyword.

134

Example

SELECT RTrim(vend_name) + ' (' + RTrim(vend_country) + ')' AS vend_title FROM vendors ORDER BY vend_name;

- **NOTE:** AS Is Optional. Unlike in most other SQL implementations, in T-SQL the AS keyword is actually optional.
- As such, SELECT vend_name AS VendName and SELECT vend_name VendName accomplish the same thing.
- In practice, it is a good idea to always specify the AS keyword (so that you'll
- be used to using it when you find yourself using another DBMS).

135

Performing Mathematical Calculations

- Another frequent use for calculated fields is performing mathematical calculations on retrieved data.
- Let's take a look at an example.

  SELECT prod_id, quantity, item_price FROM orderitems WHERE order_num = 20005;
- The orders table contains all orders received, and the orderitems table contains the individual items within each order.
- The above SQL statement retrieves all the items in order number 20005.

136

T-SQL Mathematical Operators

- Operator
  - + Addition
  - - Subtraction
  - * Multiplication
  - / Division
  - % Modulo
- How to Test Calculations?
- Although SELECT is usually used to retrieve data from a table, the FROM clause may be omitted to simply access and work with expressions.
  - SELECT 3 * 2; would return 6,
  - SELECT Trim(' abc '); would return abc
  - SELECT GetDate() uses the GetDate() function to return the current date and time.

137

Presented by Ranjan Bhatnagar

Understanding Functions

- Like almost any other language, SQL supports the use of functions to manipulate data.
- Functions are operations that are usually performed on data, usually to facilitate conversion and manipulation.
- Most SQL implementations support the following types of functions:
  - Text functions
  - Numeric functions
  - Date and time functions
  - System functions

138

Types of Functions

- Text functions are used to manipulate strings of text (for example, trimming or padding values and converting values to upper and lowercase).
- Numeric functions are used to perform mathematical operations on numeric data (for example, returning absolute numbers and performing algebraic calculations).
- Date and time functions are used to manipulate date and time values and to extract specific components from these values (for example, returning differences between dates and checking date validity).
- System functions return information specific to the DBMS being used (for example, returning user login information or checking version specifics).

139

Commonly used Text Manipulation Functions

- **CharIndex()** Returns the position of a specified character within a string
- **Left()** Returns characters from the left of a string
- **Len()** Returns the length of a string
- **Lower()** Converts string to lowercase
- **LTrim()** Trims white space from the left of a string
- **Replace()** Replaces characters within a string with other specified characters
- **Right()** Returns characters from the right of a string
- **RTrim()** Trims white space from the right of a string
- **Soundex()** Returns a string's SOUNDEX value
- **Str()** Converts a numeric value to a string
- **SubString()** Returns characters from within a string
- **Upper()** Converts string to uppercase

140

Few Examples – Text Functions

- SELECT vend_name, UPPER(vend_name) AS vend_name_upcase FROM vendors ORDER BY vend_name;
- SELECT LEFT(prod_name, 5) FROM Products ORDER BY prod_id;
- SELECT REPLACE('abcdefghicde','cde','xxx');
- SELECT cust_name, cust_contact FROM customers WHERE Soundex(cust_contact) = Soundex('Y Lie');
- SELECT LOWER(SUBSTRING(prod_name, 1, 20)) AS Lower, UPPER(SUBSTRING(prod_name, 1, 20)) AS Upper, LOWER(UPPER(SUBSTRING(prod_name, 1, 20))) As LowerUpper FROM products WHERE prod_price between 11.00 and 20.00;

141

Commonly Used Date and Time Manipulation Functions

- **DateAdd()** Adds to a date (days, weeks, and so on)
- **DateDiff()** Calculates the difference between two dates
- **DateName()** Returns a string representation of date parts
- **DatePart()** Returns parts of a date (day of week, month, year, and so on)
- **Day()** Returns the day portion of a date
- **GetDate()** Returns the current date and time
- **Month()** Returns the month portion of a date
- **Year()** Returns the year portion of a date

142

Supported Date Parts and Abbreviations

- Day → dd or d
- Dayofyear → dy or y
- Hour → hh
- Millisecond → ms
- Minute → mi or n
- Month → m or mm
- Quarter → q or qq
- Second → ss or s
- Week → wk or ww
- Weekday → (DatePart() only) dw
- Year → yy or yyyy

143

Presented by Ranjan Bhatnagar

Few Examples – Date Time Functions

- SELECT order_num, DatePart(weekday, order_date) AS weekday FROM orders;
- SELECT order_num, DateName(weekday, DatePart(weekday, order_date)) AS weekday FROM orders;
- SELECT cust_id, order_num FROM orders WHERE order_date = '2005-09-01';
- SELECT cust_id, order_num FROM orders WHERE DateDiff(day, order_date, '2005-09-01') = 0;
- SELECT cust_id, order_num FROM orders WHERE DateDiff(month, order_date, '2005-09-01') = 0;
- SELECT cust_id, order_num FROM orders WHERE Year(order_date) = 2005 AND Month(order_date) = 9;

144

Commonly Used Numeric Manipulation Functions

- Abs() Returns a number's absolute value
- Cos() Returns the trigonometric cosine of a specified angle
- Exp() Returns the exponential value of a specific number
- Pi() Returns the value of pi
- Rand() Returns a random number
- Round() Returns a number rounded to a specified length or precision
- Sin() Returns the trigonometric sine of a specified angle
- Sqrt() Returns the square root of a specified number
- Square() Returns the square of a specified number
- Tan() Returns the trigonometric tangent of a specified angle

145

Few Examples – Numeric Functions

- SELECT ABS(-1.0), ABS(0.0), ABS(1.0);
- SELECT RAND(100), RAND(), RAND();
- SELECT ROUND(123.9994, 3), ROUND(123.9995, 3), ROUND(123.9995, -1);
- SELECT SQRT(1.00), SQRT(10.00);
- SELECT PI();
- SELECT EXP(LOG(20)), LOG(EXP(20))
- SELECT SIN(45.175643), SIN(90);
- SELECT COS(14.76) AS cosCalc1, COS(-0.1472738) AS cosCalc2;
- SELECT TAN(PI()/2), TAN(.45) ;

146

Commonly Used System Functions

- @@IDENTITY Is a system function that returns the last-inserted identity value.
- HOST_NAME() Returns the workstation name.
- ISNUMERIC() Determines whether an expression is a valid numeric type.
- ISNULL() Replaces NULL with the specified replacement value
- ERROR_MESSAGE() Returns the message text of the error that caused the CATCH block of a TRY…CATCH construct to be run.

147

Few Examples – System Functions

- SELECT @@IDENTITY AS 'Identity';
- SELECT HOST_NAME();
- SELECT vend_name, vend_city FROM vendors WHERE ISNUMERIC(vend_zip)<> 1;
- SELECT AVG(ISNULL(prod_price, 50)) FROM products;
- **BEGIN TRY**

```
BEGIN TRY
    SELECT 1/0;
END TRY
BEGIN CATCH
   SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

148

Using Aggregate Functions

- It is often necessary to summarize data without actually retrieving it all, and SQL Server provides special functions for this purpose.
- Using these functions, T-SQL queries are often used to retrieve data for analysis and reporting purposes.
  - Determining the number of rows in a table (or the number of rows that meet some condition or contain a specific value)
  - Obtaining the sum of a group of rows in a table
  - Finding the highest, lowest, and average values in a table column (either for all rows or for specific rows)

149

Presented by
Ranjan Bhatnagar

Aggregate Functions & Examples

- Avg() Returns a column's average value
```
SELECT Avg(prod_price) AS avg_price FROM products;
SELECT Avg(prod_price) AS avg_price FROM products
        WHERE vend_id = 1003;
```
- Count() Returns the number of rows in a column
```
SELECT Count(*) AS num_cust FROM customers;
SELECT Count(cust_email) AS num_cust FROM customers;
```
- Max() Returns a column's highest value
```
SELECT Max(prod_price) AS max_price FROM products;
```
- Min() Returns a column's lowest value
```
SELECT Min(prod_price) AS min_price FROM products;
```
- Sum() Returns the sum of a column's values
```
SELECT Sum(quantity) AS items_ordered FROM orderitems
WHERE order_num = 20005;
```

150

Aggregates on Distinct Values

- The five aggregate functions can all be used in two ways:
  - To perform calculations on all rows, you specify the ALL argument or specify no argument at all (because ALL is the default behavior).
  - To include only unique values, you specify the DISTINCT argument.
- The following example uses the Avg() function to return the average product price offered by a specific vendor.
- Here the DISTINCT argument is used so the average only takes into account unique prices
```
SELECT Avg(DISTINCT prod_price) AS avg_price
FROM products WHERE vend_id = 1003;
```

151

Combining Aggregate Functions

- SELECT statements may contain as few or as many aggregate functions as needed.
```
SELECT Count(*) AS num_items,
    Min(prod_price) AS price_min,
    Max(prod_price) AS price_max,
    Avg(prod_price) AS price_avg
FROM products;
```
- Here, a single SELECT statement performs four aggregate calculations in one step and returns four values (the number of items in the products table as well as the highest, lowest, and average product prices).

152

Grouping Data

- Groups are created using the GROUP BY clause in your SELECT statement.
- The best way to understand this is to look at an example:
- If you want to return the number of products offered by each vendor? Or products offered by vendors who offer a single product, or only those who offer more than 10 products? This is where groups come into play.
```
SELECT vend_id, Count(*) AS num_prods
FROM products GROUP BY vend_id;
```
- Grouping enables you to divide data into logical sets so you can perform aggregate calculations on each group.

153

Important Rules while Grouping

- GROUP BY clauses can contain as many columns as you want. This enables you to nest groups, providing you with more granular control over how data is grouped.
- If you have multiple groups specified in your GROUP BY clause, data is summarized at the last specified group.
- Every column listed in GROUP BY must be a retrieved column or a valid expression. If an expression is used in the SELECT statement, that same expression must be specified in it.
- Aliases cannot be used.
- Aside from the aggregate calculations statements, every column in your SELECT statement should be present in the GROUP BY clause.
- If the grouping column contains a row with a NULL value, NULL will be returned as a group. If there are multiple rows with NULL values, they'll all be grouped together.
- The GROUP BY clause must come after any WHERE clause and before any ORDER BY clause.

154

Filtering Groups

- In addition to being able to group data using GROUP BY, SQL Server also allows you to filter which groups to include and which to exclude.
- For example, you might want a list of all customers who have made at least two orders.
- To obtain this data, you must filter based on the complete group, not on individual rows.
- In such cases WHERE will not work, because WHERE filters specific rows, not groups.
- T-SQL provides yet another clause for this purpose: the **HAVING** clause.

155

Presented by Ranjan Bhatnagar

Having Example

```
SELECT cust_id, Count(*) AS orders FROM orders
GROUP BY cust_id HAVING Count(*) >= 2;
```

- The first three lines of this SELECT statement are similar to the statements shown previously.
- The final line adds a HAVING clause that filters on those groups with a Count(*) >= 2, that is, two or more orders.
- As you can see, a WHERE clause does not work here because the filtering is based on the group aggregate value, not on the values of specific rows.

156

Where n Having Together

- So is there ever a need to use both the WHERE and HAVING clauses in one statement?
- Actually, yes, there is. Suppose you want to further filter the previous statement so it returns any customers who placed two or more orders in the past 12 months.
- To do that, you can add a WHERE clause that filters out just the orders placed in the past 12 months.
- You then add a HAVING clause to filter just the groups with two or more rows in them.
- To better demonstrate this, look at the following example, which lists all vendors who have two or more products priced at 10 or more:

```
SELECT vend_id, Count(*) AS num_prods FROM products
WHERE prod_price >= 10 GROUP BY vend_id HAVING Count(*) >= 2;
```

157

Grouping and Sorting

- GROUP BY and ORDER BY perform different but related functions.
- As a rule, any time you use a GROUP BY clause, you should also specify an ORDER BY clause.
- That is the only way to ensure that data is sorted properly. Never rely on GROUP BY to sort your data.
- The following SELECT statement is similar to the ones shown previously.
- It retrieves the order number and total order price of all orders with a total price of 50 or more:

```
SELECT order_num, Sum(quantity*item_price) AS ordertotal
FROM orderitems GROUP BY order_num
HAVING Sum(quantity*item_price) >= 50
ORDER BY ordertotal;
```

158

SELECT Clauses and Their Sequence

Clause	Description	Required
SELECT	Columns or expressions to be returned	Yes
FROM	Table to retrieve data from	Only if selecting data from a table
WHERE	Row-level filtering	No
GROUP BY	Group specification	Only if calculating aggregates by group
HAVING	Group-level filtering	No
ORDER BY	Output sort order	No

159

Presented by
Ranjan Bhatnagar