

Debere Birhan Polytechnic College

DATABASE ADMINISTRATION SERVICES Level III


MODULE TITLE: **Using Basic Structured Query Language**

LO1. Write an SQL Statement to retrieve and sort data

1.1 Introduction to SQL

1.1.1 What is SQL?

SQL (Structured Query Language) is a programming language designed for managing data in relational database management systems (RDBMS).

 It is a standard (main query) language for relational DBMSs, like Microsoft Access, Microsoft SQL Server, and Oracle, that used to communicate with a database.

SQL is used to perform basic data management tasks, such as the insertion, modification, and deletion of data from the tables using the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop".

With SQL, you can easily enter data into the database, modify data, delete data, and retrieve data. SQL is a non-procedural language, which means that you tell the database server what data to access, not necessarily what methods should be used to access the data. SQL contains the following five sub-languages that allow you to perform nearly any operation desirable within a relational database:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data control language (DCL)
- Transaction control commands(TCC)

Data Definition Language (DDL)

DDL is used to define the structure of the database, to include creating tables, dropping tables, defining views, indexes, and constraints, and so on. DDL commands exist in the following

- ✎ **CREATE** commands allow database objects, such as tables, to be created
- ✎ **DROP** commands allow database objects to be removed (deleted) from the database.
- ✎ **ALTER** commands allow the structure of database object to be modified

Popular **CREATE** commands include:

- ✎ **CREATE TABLE**
- ✎ **CREATE INDEX**
- ✎ **CREATE VIEW**
- ✎ **CREATE PROCEDURE**

Popular **ALTER** commands include:

- ✎ **ALTER TABLE**
- ✎ **ALTER PROCEDURE**
- ✎ **ALTER VIEW**
- ✎ **ALTER TRIGGER**

Popular **DROP** commands include:

- ✎ **DROP TABLE**
- ✎ **DROP INDEX**
- ✎ **DROP VIEW**
- ✎ **DROP PROCEDURE**

Manipulating Data Language (DML)

DML allows you to modify data within the database. It allows a database user to affect data in the database. With DML, the user can populate tables with new data, update exist data in tables, and delete data from tables. Simple database queries can also be performed with a DML command. There are three basic DML commands in SQL.

- ✎ **INSERT**
- ✎ **UPDATE**
- ✎ **DELETE**

INSERT Allows for the insertion of new data into an existing table, UPDATE commands allows existing data to be changed or updated, DELETE command also allows data to be deleted from database tables.

Data Query Language (DQL)

Data Query Language (DQL) is the component of SQL that allows the database users to query or request information from the database. A query is an inquiry into the database using SELECT statement. A query is used to extract data from the database in a readable format according to the users' request. For example, if you have an employee table you might issues a SQL statement that returns the employee who is paid most. This request to the database for usable employee information is a typical query that cans be performed in relational database.

The SELECT statement, the command that represents DQL in SQL, is the statement used to construct database queries. The SELEGRANTCT statement is not a standalone statement, which means that one or more additional clauses (elements) are required for a syntactically correct query. In addition to the required clauses, there are optional clauses that increase the overall functionality of the SELECT statements. The SELECT statement is by far one of the most powerful statements in SQL. The FROM clauses is the mandatory clauses and must have always be used in conjunction with SELECT statement.

A simple SQL query using the SELECT statement might appear as follows:-

```
SELECT EMPLOYEE_NAME FROM EMPLOYEES
```

This query would return a list itself of all employees' name as found in the EMPLOYEES table.

Though composed of only one command, DQL is the most concentrated focus of SQL for modern relational database users. This commands, accompanied by many options and clauses, is used to compose queries against a relational database, queries, from simple to complex, from vague to specific, can be easily created.

Data Control Language (DCL)

Data control commands in SQL enables you to control access to data within the database. Control is providing not only for access into the database via user authentication, but for access to database objects through privileges. These DCL commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some common data control are as follows:-

✎ **ALTER PASSWORD**

✎ **GRANT**

✎ **REVOKE**

Transaction Control Commands (TCC)

In addition to previously introduced commands categories of commands, commands exist that allow the user to manage database transactions. Database transactions are conducted using any combination of the DML commands. These are transactional control commands. Transactional control commands provide two basic benefits to the database server.

- 1) The capability to finalize a transaction and save changes to the database
- 2) The capability to undo a transaction

Transactional control commands exist in the following forms:

✎ COMMIT used to save database transaction

✎ ROLLBACK used to undo database transaction

✎ SAVEPOINT creates points within groups of transaction in which to ROLLBACK, or undo work.

✎ SET TRANSACTION used to label a set of DML commands as a single transaction

1.1.2. Categories of SQL Application

Microsoft SQL Server is a powerful and reliable data management system that delivers a rich set of features, data protection, and light Web applications.

- ❖ Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, MS- Access, etc.
- ❖ SQL Application is a database language that allow a user to:
 - Define (create) the database, table structures, and controlling access to the data using DDL.
 - Perform both simple and complex queries using DML.

1.2 Installing DBMS software

Introduction

A *database management system* is the software that enables users to define, create, and maintain the database and also provides controlled access to this database.

❖ Some of the most common database applications are:

- ✓ Microsoft Access
- ✓ Microsoft SQL
- ✓ Oracle, and Informix

1.2.1 Hardware Requirements

A processor with high speed of data processing and memory of large size (RAM and Hard disk space) is required to run the DBMS software.

Example: The Minimum hardware requirements for SQL Server 2008 installations include:

- ✎ Processor
 - Pentium 600 MHz or higher is required
 - 1 GHz or higher is recommended.
- ✎ Memory
 - 512 MB is required
 - 1 GB or more is recommended
 - Express Edition requires 192 MB and recommended 512 MB or more.
- ✎ Disk space
 - Database components: 280 MB
 - Analysis services: 90 MB
 - Reporting services: 120 MB
 - Integration services: 120 MB
 - Client components: 850 MB

1.2.2 Operating System Requirements

The supported operating system for DBMS software may depends on the type and version of the software.

Example: The Supported operating systems for SQL Server 2008 installations include:

- ✎ Windows 7
- ✎ Windows Server 2003 Service Pack 2
- ✎ Windows Server 2008
- ✎ Windows Server 2008 R2
- ✎ Windows Vista Service Pack 1
- ✎ Windows XP Service Pack 3

1.2.3 Install DBMS (SQL Server)

📖 To install SQL Server 2008 Express, you must have administrative rights on the computer.

📖 The different editions/version of SQL Server share several common software requirements such as Microsoft Windows installer and Microsoft .Net Framework.

Example: Before installing **SQL Server 2008 Express SP1**, first you have to install **Microsoft Windows installer 4.5** and **Microsoft .NET Framework version 3.5 SP1**.

The SQL Server Installation Wizard provides a single feature tree to install all SQL Server components such as:

- ✎ Database Engine
- ✎ Analysis Services
- ✎ Reporting Services
- ✎ Integration Services
- ✎ Master Data Services
- ✎ Data Quality Services
- ✎ Management tools
- ✎ Connectivity components

You can install each component individually or select a combination of the components listed above.

A Step by Step guide to installing SQL Server 2008 simply and successfully:

📖 This will teach you the basics required for a typical (problem-free) installation of SQL Server 2008.

📖 Before you start the installation, you will need to install the .Net 3.5 Framework and windows installer 4.5.

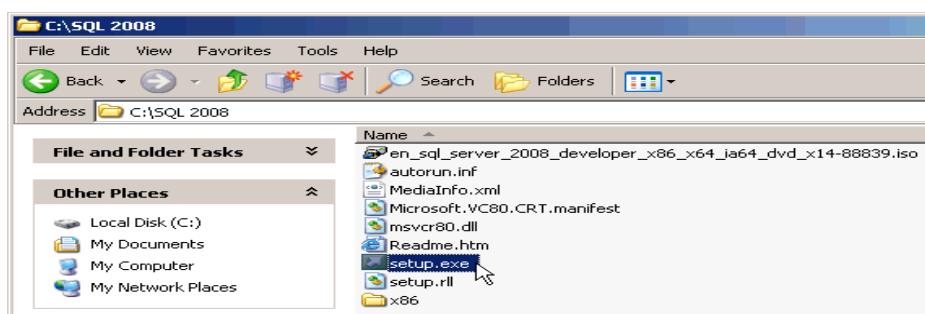
STEP 1: Copy the installation files

First it is recommended that you copy the entire directory structure from the SQL Server 2008 installation disc to the C: drive of the machine you are going to install it on.

This has three advantages:

- It makes the installation process much faster than running it from CD/DVD once it gets started.
- It allows you to easily add or remove components later, without having to hunt around for the CD/DVD.
- If your media is damaged and a file won't copy, you get to find out now, rather than halfway through the installation.

Here's what my system looks like after the copy:



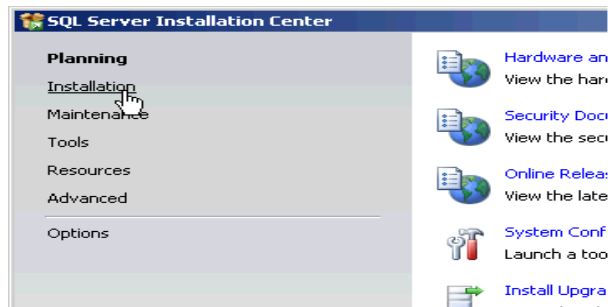
STEP 2: Double click on the setup.exe file. After a few seconds a dialog box appears:



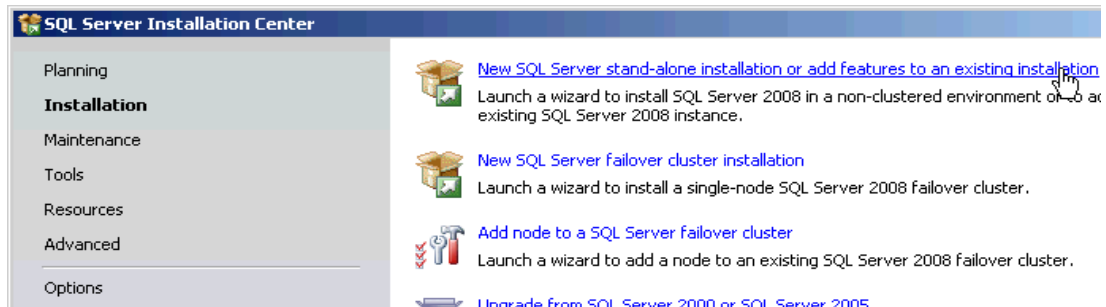
This will disappear from the screen and then the main installation page appears:



STEP 3: Click on the **Installation** hyperlink on the left hand side of the screen:



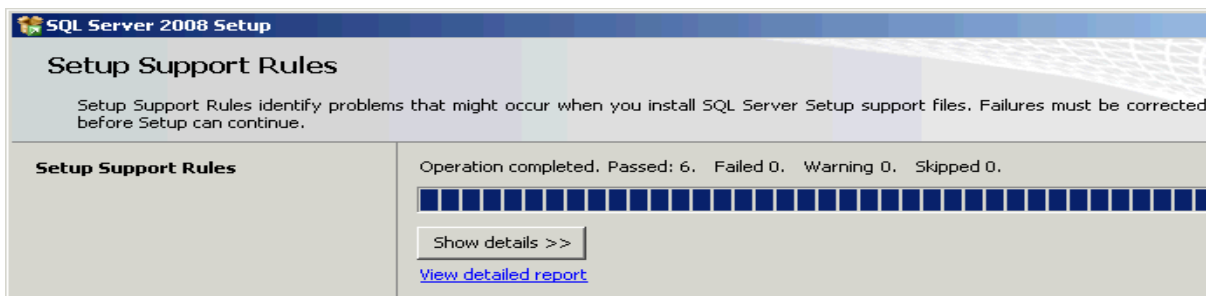
STEP 4: Click on the "New Server stand-alone installation" link on the right side of the screen:



The following dialog appears on the screen whilst the install program prepares for installation:



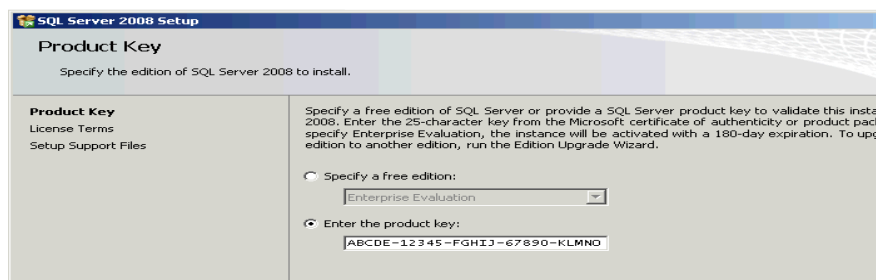
After a minute or so (the timing will vary according to your system), the following screen appears:



STEP 5 (optional): If any checks have failed, click on the Show details button or "View detailed report link" to find out the cause, correct it, and then click on the Re-run button to perform the checks again.

STEP 6: Product key

If all checks have passed, click on the OK button. After a few moments, the option to select the edition and to enter the license key (or "product key") will appear. Note that the product key box may already be populated, depending on which edition you have. Don't enter the product key we've shown here, it won't work on your system!:

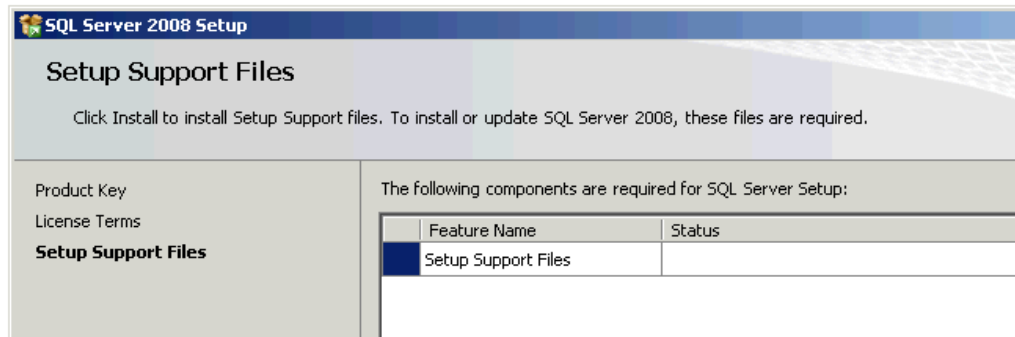


STEP 7: License Terms

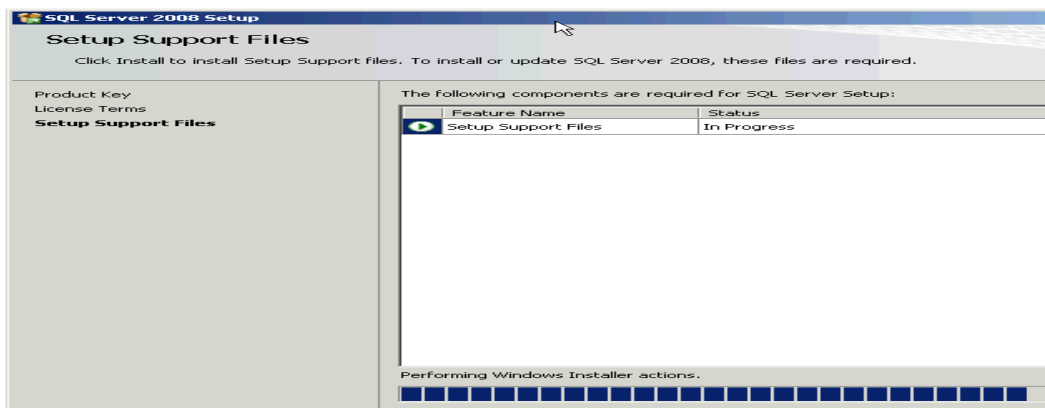
Enter the product key into the box, or choose the free edition if you're evaluating SQL Server 2008, and click on the Next button:

Click in the **"I accept the license terms"** check box, and then click on the **Next** button again.

STEP 8: click on the **Install** button:



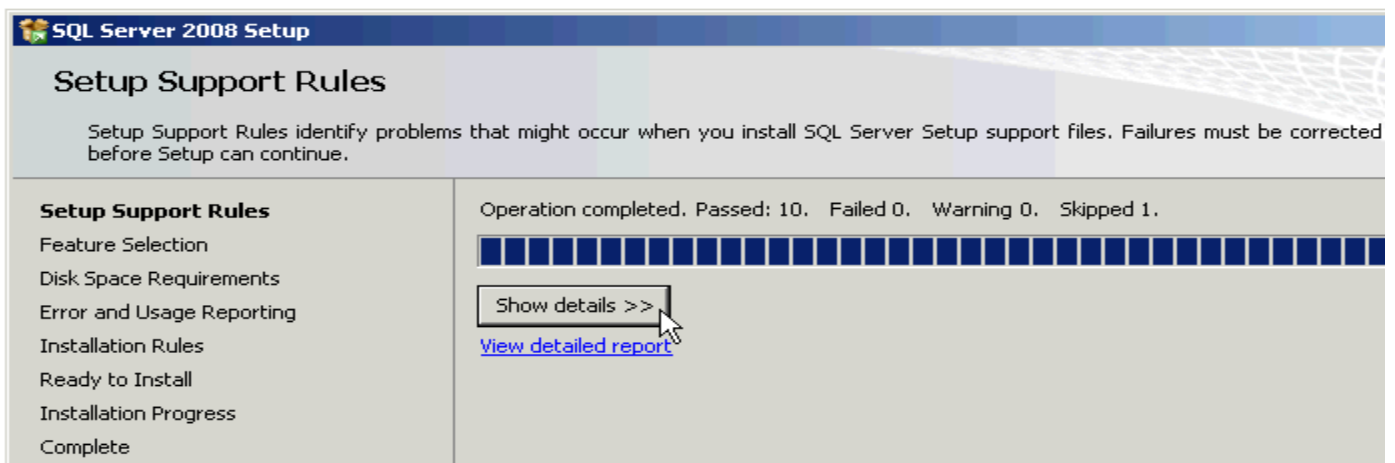
The following screen will appear whilst Windows Installer prepares itself for the installation. This will take a short while:



After 30 seconds or so the dialog appears again:



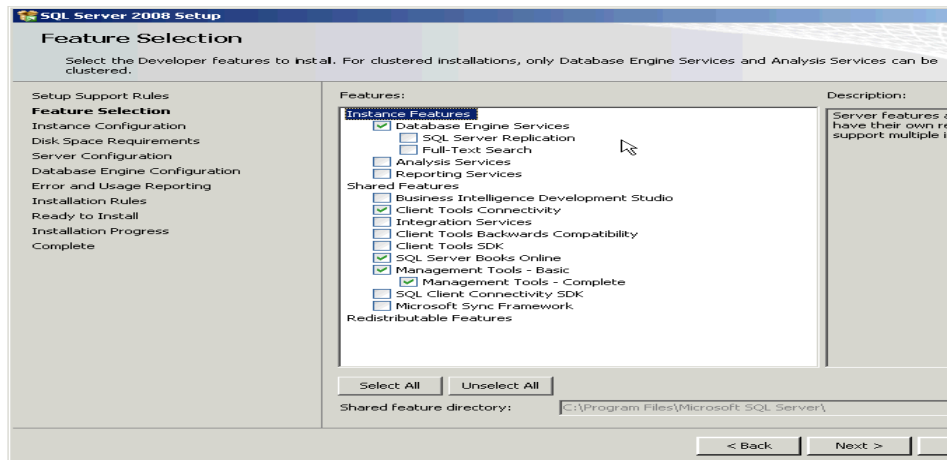
STEP 9: Setup Support Rules. If all is well, the following screen appears:



Click on the **Next** button again.

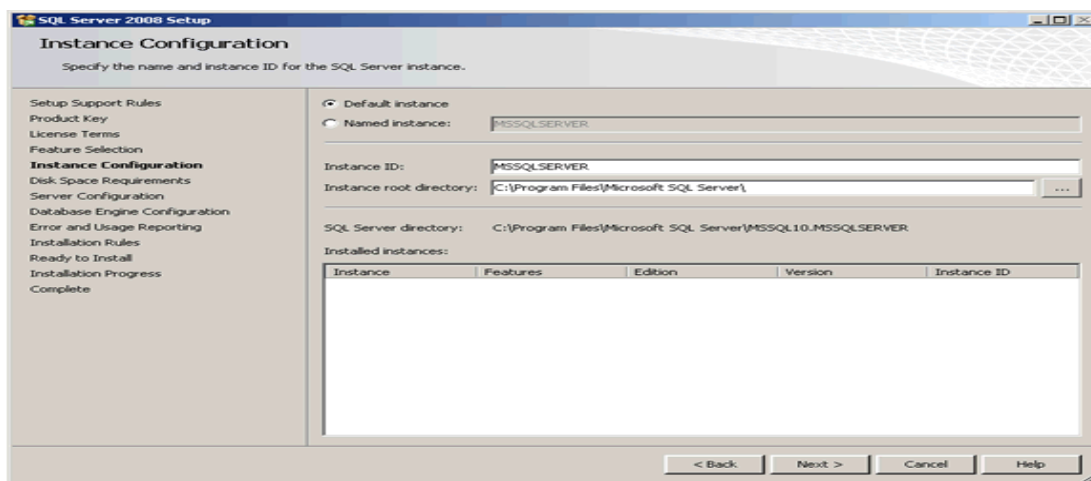
STEP 10: Feature Selection. Select the features you want to install.

At a minimum, the following are useful (I'd argue essential), but what you need will depend on your needs:



Click on the **Next** button.

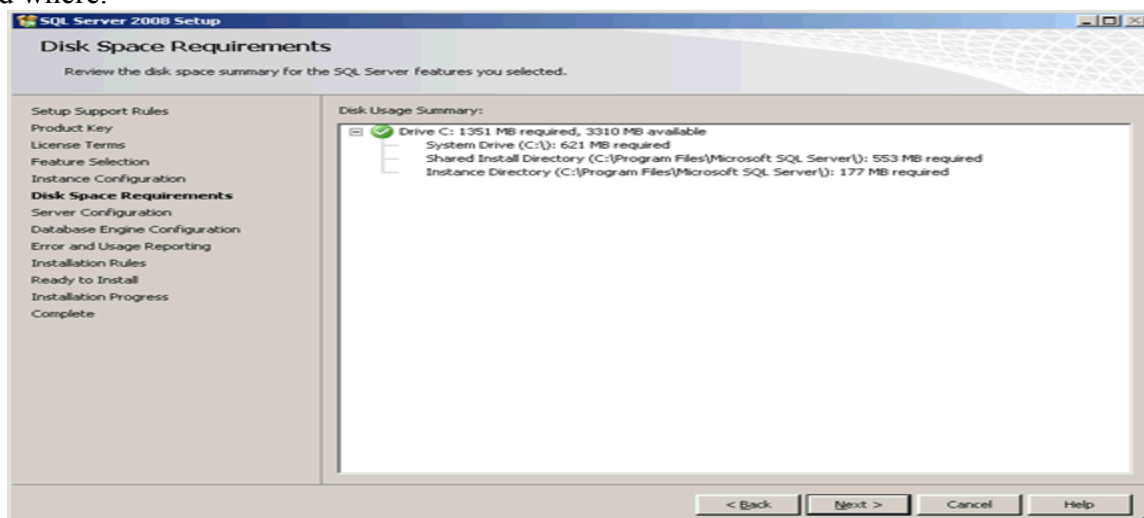
STEP 11: Instance Configuration. After a short while the following screen appears:



For most installations, keep the default settings. Click on the **Next** button.

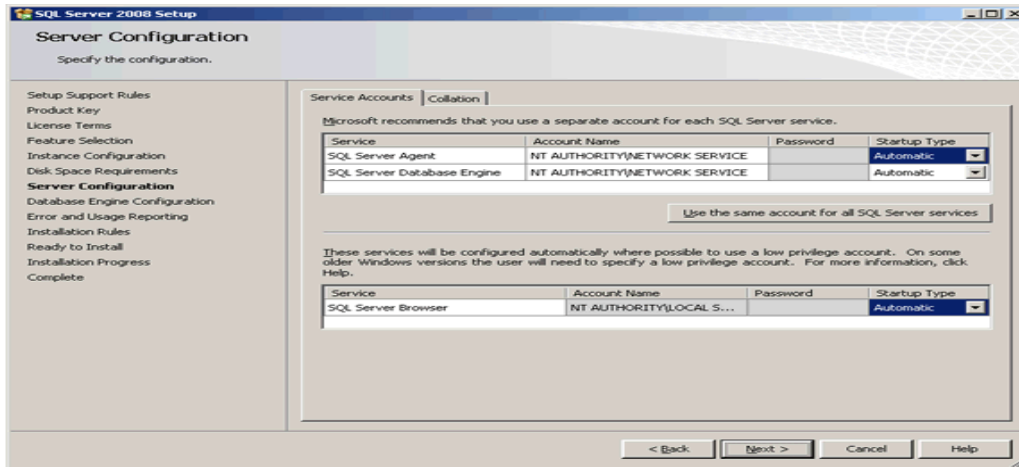
STEP 12: Disk Space Requirements

This screen just tells you if you have sufficient disk space on the drive you're installing to, and what's going to be installed where.



Click on **Next**.

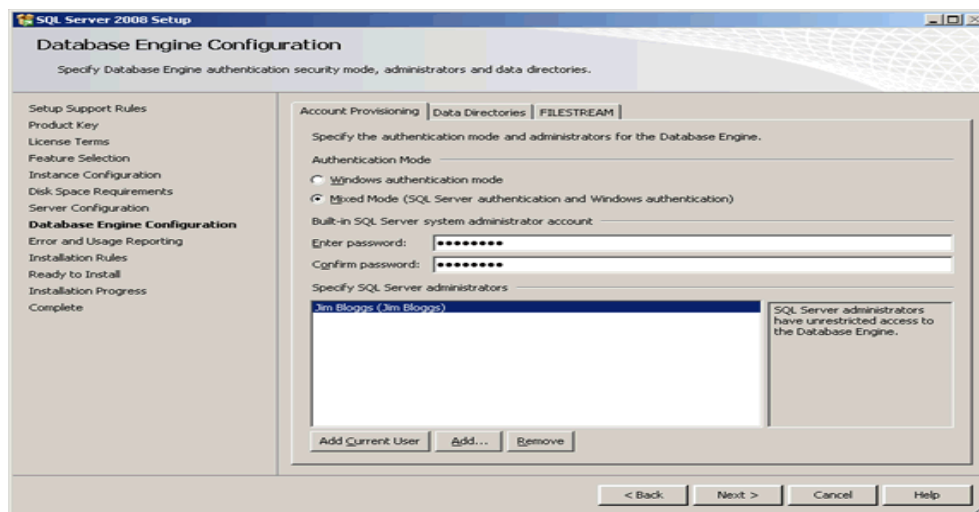
STEP 13: Server Configuration. This step allows you to set up the service accounts that will be used to run SQL Server. If you have created Windows NT or Active Directory accounts for use with services, use these. If not, then just to get the installation up and working, use the built-in Network Service account for all three services listed (this account does not require a password). This allows SQL Server to start up after installation. However, it can be easily changed later to another account through the Services applet (Control Panel -> Administrator Tools -> Services):



In addition, remember to change the **Startup Type** to **Automatic**, for all three services. This automatically starts the SQL Server database engine, SQL Agent and SQL Browser services when the server is re-booted. The first service runs the SQL Server database engines executable process. The other two services allow scheduled jobs to run after installation (and after a re-boot), and allow the SQL Server to be found by clients on the network. Finally, click on **Next**.

STEP 14: Database Engine Configuration – Account Provisioning.

This screen allows you to set up database engine security.



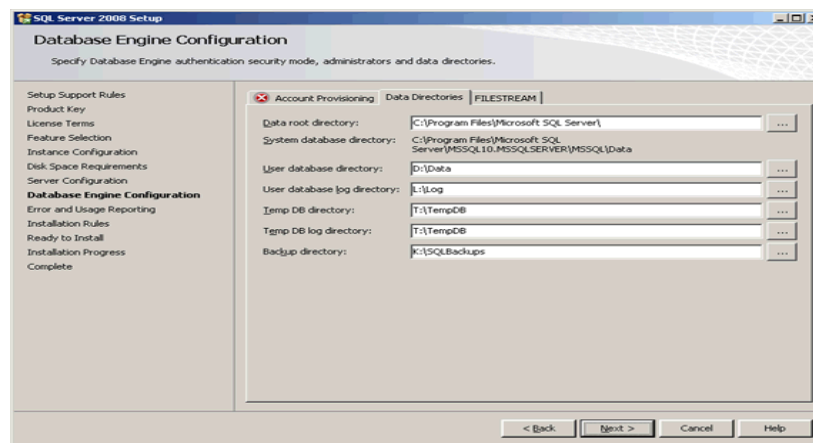
Change the **Authentication Mode** to **Mixed Mode** unless you are **certain** you only need Windows-only authentication.

- **Many third party applications rely on SQL Server logins to operate correctly, so if you are setting up a server for a third party application, rather than one developed in-house, enabling Mixed Mode authentication is a good idea.**

If you pick Mixed Mode security, you must also enter a password for the sysadmin account (sa). Enter and confirm a secure password for the sa account and keep it somewhere safe.

Note that you **MUST** also provide a Windows NT account on the local machine as a SQL Server administrator. If you do not want Windows system administrators to be able walk up to the box and login to SQL Server, create a new, local, dummy Windows user and add this account instead. Otherwise, add in the local administrator account, or your own Windows account on the domain in which the SQL Server will reside.

STEP 15: Database Engine Configuration – Data Directories. Click on the **Data Directories** tab.



Change the directories to specify which drives in your system will be used for the various types of database files.

Generally it's advisable to put the User database directory and User log directory on separate physical drives for performance, but it will depend on how Windows has been configured and how many disk drives you have available.

If you are installing on a single drive laptop or desktop, then simply specify:

Data root directory C:\Program Files\Microsoft SQL Server

User database directory C:\Data

User log directory C:\Logs

Temp DB directory C:\TempDB

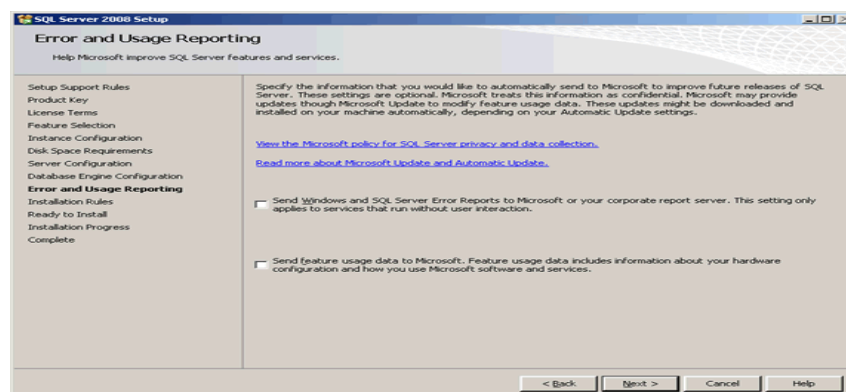
Temp Log directory C:\TempDB

Backup directory C:\Backups

Do not click on the **FILESTREAM** tab unless you know you need to change these options, as it is not generally required for most installations, but can easily be changed by using sp_configure 'filestream_access_level', "after SQL Server has been installed. Click on **Next**.

STEP 16: Error Usage Reporting

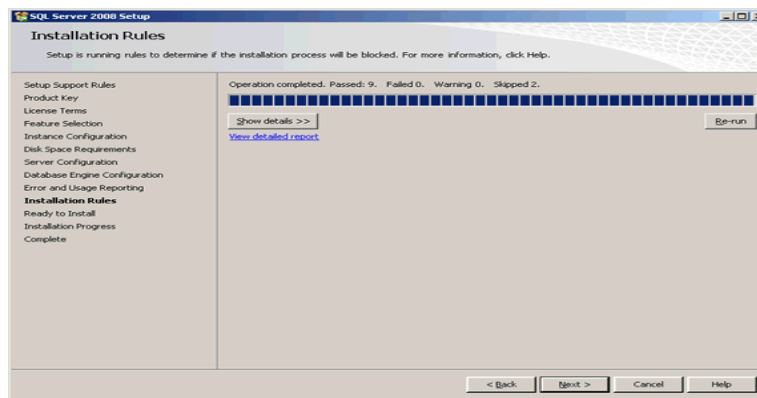
This screen simply asks if you want to send error information to Microsoft and can safely be skipped if you do not want to share any information.



Click boxes if you want to help Microsoft help you. Click on **Next** again...

STEP 16: Installation Rules

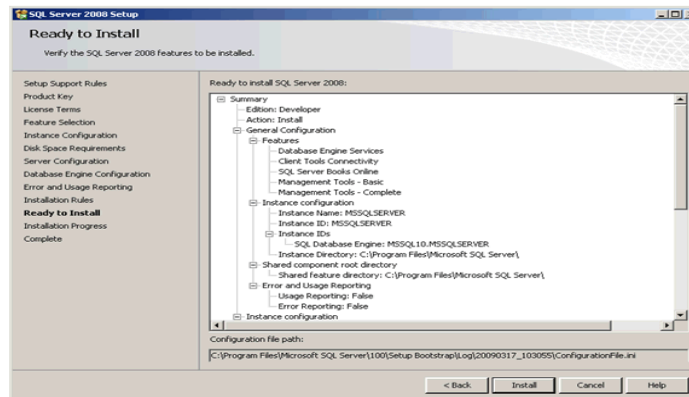
This screen simply checks if there are any processes or other installations running which will stop the installation of SQL Server 2008.



Click on **Next** again – you're almost ready to install:

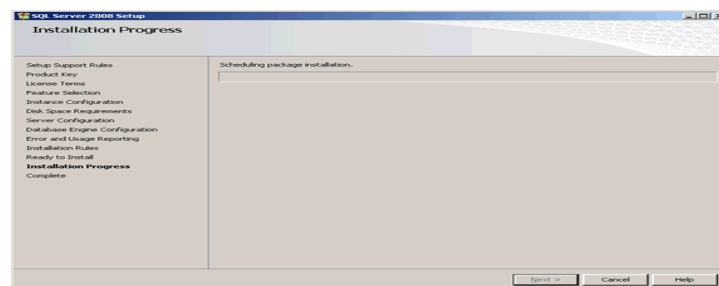
STEP 17: Ready to Install

This screen summarizes what you are about to install and gives you a last chance to cancel or change anything that's wrongly configured:

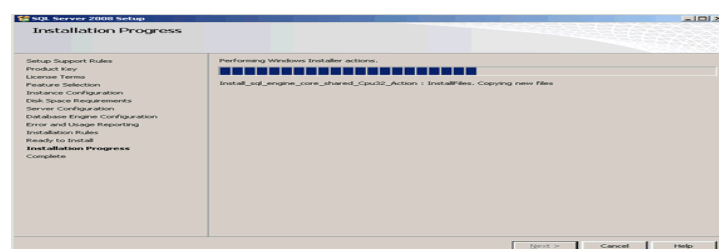


Check that what's being installed is what you want and then click on **Install** when you're sure you want to start the installation process:

Installation Progress: SQL Server 2008 will now install. How long it takes depends on the speed of your machine.

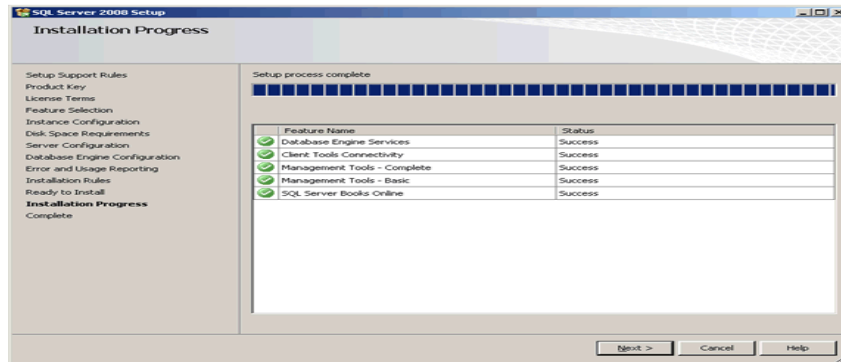


...More Installation Progress

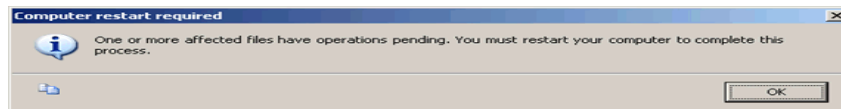


... and Finally

Finally, the installation will complete:



...and the following dialog box will appear:

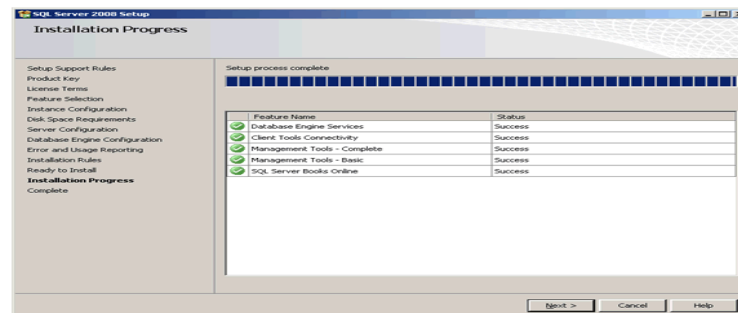


Click on OK, the machine will NOT reboot.

The following will appear:

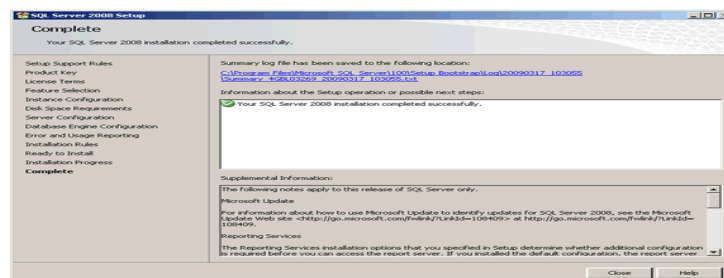


...followed by:



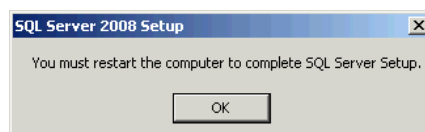
Click on the Next button again...

STEP 18: Installation Complete. The following screen appears:



It may be worth clicking on the installation log at the top of the screen to check everything's gone as expected. Not that this is MUCH smaller than the usual SQL Server installation log files of old.

Finally, **click on the Close button.** The following dialog will appear:



Click on OK – your server will NOT re-boot at this point.

The dialog box will disappear and you will be returned to the Installation Center:



Click on the Close button (the “x”) in the top right of the screen.

Finally, manually reboot your machine to complete the SQL Server 2008 installation.

Top Tips: How to check that SQL Server 2008 has installed correctly

Here are a short number of post-installation checks which are useful to perform after re-booting your new SQL Server.

Check 1: Has the SQL Server Service Started?

Check SQL Server 2008 has started.

SQL Server (MSSQLSERVER)	Provides storage, processing and controlled access of data, ...	Started	Automatic	NT AUTHORITY\NETWORK SERVICE
SQL Server Agent (MSSQLSERVER)	Executes jobs, monitors SQL Server, fires alerts, and allows ...	Started	Automatic	NT AUTHORITY\NETWORK SERVICE
SQL Server Browser	Provides SQL Server connection information to client comput...	Started	Automatic	NT AUTHORITY\LOCAL SERVICE
SQL Server VSS Writer	Provides the interface to backup/restore Microsoft SQL serv...	Started	Automatic	Local System

Check 2: Does Management Studio Work? Check Management Studio works by firing it up.

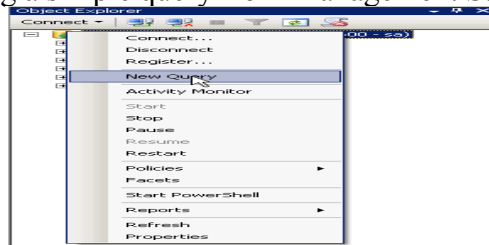


Click on NO when you see this dialog box:

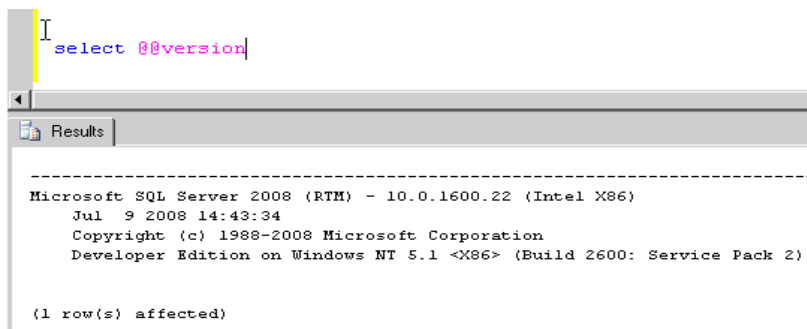


Check 3: Can you run a basic query against the new SQL Server?

Check SQL Server works by running a simple query from Management Studio:



Enter the query shown below and hit F5 to run it:



1.3 Create database

Before any database objects or data can be defined, the database itself must be created. When speaking of database creation, two ideas may come to mind depending on your background. To some individuals, the database corresponds to the database software and background structure that hosts the environment for database objects and data itself. Both descriptions accurately compose a database. During database creation you will explore:

- ✎ The creation of the database environment refers to the database software and background structure that is used to host database objects and data.
- ✎ The creation of objects that compose a database, referring to elements such as tables, indexes, and data.

Creating database environment

The creation of the database environment typically includes the following steps:

- ✎ Installation of the database server
- ✎ Configuration of database server
- ✎ Creation of the database
- ✎ Establishment of database users to create database objects.

Create object that composes the a database

When you create database objects, a full utilization of SQL is involved. More specifically, DDL, the component of the SQL language that is used to create and manage database objects, is used to manage database objects in the following ways:

- ✎ **CREATE:** This command is used to create or defined database objects.
- ✎ **ALTER:** This command is used to alter the structure or definition of database objects.
- ✎ **DROP:** This command is used to delete database objects.

The following list defines the most common database objects that are created and managed SQL's DDL.

- ✎ **Tables**
- ✎ **Indexes**
- ✎ **Views**
- ✎ **Stored procedures**
- ✎ **Constraints**

Data definition

Before create the actual database objects, it is important to have a firm grasp on data definition itself. Data is a collection of stored elements in the database as one of several data types. Data includes name, numbers, dollar amounts, text, graphics, decimals, figures, calculations, summarization, and just about else you can possibly imagine. Data can be stored in uppercases, lowercases, mixed case letters. Data can be manipulated or changed: most data does not remain static for its life time.

DDL is used to define data and implement the assignment of data types, which enables the storage of data.

Creating a Database

📖 When you create a database you need to use the **Master database**.

1. To create a database using New Database wizard:

- ✓ Right-click on **Databases**, and then select **New Database**.
- ✓ In **Database name** field, enter a "**database name**".
- ✓ To create the database by accepting all default values, click OK; otherwise, continue with the following optional steps.
- ✓ To change the owner name, click (...) to select another owner.

2. To create a database using "create database statement":

1. From the Standard bar, click "New Query".
2. On the sql editor write "**create database database_name**"
3. Select the statement **<create database database_name>** and execute it to create.

Example:

```
USE master;
GO
CREATE DATABASE Sales
ON
(
NAME = Sales_dat,
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\saledat.mdf',
SIZE = 10MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
)
GO
```

✎ **Deleting a database**

You can delete database by executing the DROP DATABASE statement.

Example: DROP DATABASE **<database name>**

Exercises: 1. Create a database called **library**.

2. Delete the database that you have already created.

1.3.1 Creating, modifying and deleting Tables:

✎ Creating a table

- When you create a table you need to use the **current database**.
- SQL Server databases store all of your data in **tables**.

Syntax:

```
CREATE TABLE table_name
(
    <Column_Name 1> <data type> <field size> <constraints>,
    <Column_Name 2> <data type> <field size> <constraints>,
    . . .
    <Column_Name n> <data type> <field size> <constraints>
);
```

✎ Modifying a table

- You can modify the table by adding a new column and deleting a column from the table.
- The type of information that you specify when you add a column is similar to the activity that you perform when you create a table.

Syntax: -

```
ALTER TABLE <table_name> ADD <column_name> <data type> <field size> <constraint>
ALTER TABLE <table name> DROP COLUMN <column name>
```

✎ Deleting a Table

- Deleting a table removes that table definition and all data, as well as the permission specification for that table.
- Before you delete a table, you should remove any dependencies between the table and other objects.

Syntax: DROP TABLE *table_name*

✎ Generating Column Values

❖ **Identity Property**

- You can use the Identity property to create columns (referred to as identity columns) that contains system generated sequential values identifying each row inserted into a table.

Syntax:

```
CREATE TABLE table
(
    Column_name data type IDENTITY (seed, increment) NOT NULL
)
```

Example:

Create table student

```
(
    Student_id int identity (1, 1) NOT NULL,
    Student_name char (20)
)
```


Consider the following requirements for using the **Identity** property

- ✓ Only one identity column is allowed per table
- ✓ It must be used with integer data types.
- ✓ It cannot be updated
- ✓ It does not allow null values

Exercise

1. Create table whose name is employee with attributes like name, Id, sex, salary, Nationality and age.
2. Modify the employee table by adding a column named Qualification.
3. Modify the employee table by modifying a column named age.
4. Delete the table Employee.

1.3.2 Inserting new rows

The **INSERT** command is used to add a single tuple to a relation.

- ☞ We must specify the relation (table) name and a list of values for the tuple.
- ☞ The values should be listed in *the same order* in which the corresponding attributes were specified in the CREATE TABLE command.

Syntax:

```
Insert into <TABLE_NAME> values ('value1'<for_column 1>,  
                                'value2'<for_column 2>,  
                                'value3'<for_column 3>,  
                                . . . ,  
                                'value n'<for_column n>);
```

The INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command. This is useful if a relation has many attributes.

Syntax:

```
Insert into <TABLE_NAME> (column1, column2, column3)  
values ('value1'<for_column 1>, 'value2'<for_column 2> 'value 3'<for_column 3>);
```

1.3.3 Select statement

The **Select statement** is the most commonly used SQL command that allows you to retrieve records from one or more tables in your database.

- The basic SELECT statement in sql has 3 clauses: SELECT, FROM and WHERE

- ☞ The SELECT clause specifies the table columns that are retrieved.
- ☞ The FROM clause specifies the table or tables from which columns and rows are returned.
- ☞ The WHERE clause specifies the condition restricting the query. You can restrict the number of rows by using comparison operators, character strings, and logical operators as search conditions.
- ☞ The WHERE clause is optional; if missing, all table rows are accessed.

Syntax of SQL SELECT Statement:

```
SELECT <column_list> FROM <table_name_list> < [WHERE Clause]> < [search_condition]>
```

- ☞ **Table-name_list** includes one or more tables from which the information is retrieved.
- ☞ **column_list** includes one or more columns from which data is retrieved.
- ☞ The code within the brackets is optional.

Example: SELECT ID, Fname, Dname FROM STUDENT, DEPARTEMENT WHERE Sex='Female' AND Dnumber='cou005'

1.3.3.1 Literals and data types

Literals are letters, numbers, or symbols that are used as specific values in a result set (in output).

- ☞ Literals mean constants which are the values we write in a conventional form.
- ☞ You can include literals in the select list to make result sets more readable.

Syntax:

SELECT *column_name1* 'string literal', *column_name2* 'string_literal', ... FROM *table_name*

Example: SELECT firstname, lastname, 'Identification number:', employeeid FROM employee

Output:

Firstname	lastname	employeeid
Nancy	David	Identification number : 1
Andrew	Fuller	Identification number : 2

Data type is a constraint that specifies the type of data stored in a table field.

- * Common examples of data type in MS-Access are:
 - Auto-number, Text, Number, Date/Time, Currency, Yes/No, and so on.
- * Common examples of data type in MS-SQL server are:
 - Char, varchar, int, float, double, datetime, and so on.

1.3.3.2 Expressions

An **Expression** is a combination of symbols and operators that the SQL Server Database Engine evaluates to obtain a single data value.

- ☞ Operands are values or variables, whereas operators are symbols that represent particular actions.
- ☞ Operators can be used to join two or more simple expressions into a complex expression.
- ☞ Every expression consists of at least one operand and can have one or more operators.
 - **Example:** - In the expression, SELECT ((5+5) * (5+5)), 5 is an operand, and +, * are operators.
 - In database systems, you use expressions to specify which information you want to retrieve.

These types of expressions are called queries.

1.3.3.2.1 Comparison operators

A **comparison (or relational) operator** is a mathematical symbol or a keyword which is used to compare between two values.

- ☞ Comparison operators are used in conditions that compare one expression with another. The result of a comparison can be TRUE, FALSE, or UNKNOWN.
- ☞ SQL Comparison operator is used mainly with the SELECT statement to filter data based on specific conditions.

Comparison operator	Description
=	equal to
<>, !=	is not equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to
In	"Equivalent to any member of" test, Equivalent to "= ANY".
Not In	Equivalent to "! = ANY".
All	Compares a value with every value in a list
[Not] between	[Not] greater than or equal to <i>x</i> and less than or equal to <i>y</i> .
Is [not]null	Tests for nulls

Note: The != operator is converted to <> in the parser stage.

1.3.3.2.2 Boolean operators

Boolean Operators are simple words (AND, OR, or NOT) used as conjunctions to combine or exclude keywords in a search.

- 👉 Boolean operators are used widely in programming and also in forming database queries.
- 👉 When you want retrieving data using a SELECT statement, you can use logical operators in the WHERE clause to combine more than one condition.

Example: - `SELECT first_name, last_name, subject FROM student_details
WHERE subject = 'Maths' OR subject = 'Science'`

- 👉 The **AND** and **OR** operator combines two logical operands.
- 👉 The **NOT** operator inverts the result of a comparison expression or a logical expression.

1.3.3.2.3 Arithmetical operators

SQL mathematical operations are performed using mathematical operators (+, -, *, /, and % (Modulo)).

- ✎ When you need to perform calculations in SQL statement, you can use arithmetic expression.
- ✎ An arithmetic expression can contain column names, numeric numbers, and arithmetic operators.

Arithmetic operators are responsible for performing most calculations that use values such as:

- ✓ Literals/ Constants
- ✓ Variables
- ✓ Expressions
- ✓ Function

1.3.3.2.4 Mathematical functions

A function is a relation in which any value inputted in the relation yields exactly one output.

A mathematical function performs a mathematical operation on numeric expressions and returns the result of the operation. Multiply function is basically used in SQL

Example: - `SELECT round(salary), firstname FROM employee_info
- SELECT sum(salary), ID, firstname FROM employee_info`

1.3.3.3 Assigning names to result columns

When the result of a SELECT statement is determined, you can specify your own names for the result table columns.

- ✎ The AS clause can be used to assign a different name, or alias, to the result set column. This can be done to increase readability.
- ✎ This capability is particularly useful for a column that is derived from an expression or a function.

Example: - `SELECT SALARY+BONUS+COMM AS TOTAL_SAL FROM EMPLOYEE;
- select ID 'identifier', fname 'first name', lname as last_name from EMPLOYEE;`

Updating tables

Updating values in rows

The UPDATE statement is used to change existing records in a table.

- Use a table reference to indicate which table needs to be updated.

Syntax:

```
UPDATE table_name  
SET <column1='value1'>, <column2='value2'>, ...  
[WHERE <search-condition>]
```

- ✎ The **SET** clause is used to assign new values to one or more columns.
- ✎ The search condition (**WHERE** clause) specifies which rows in the table are to be updated. If no search condition is specified, all rows will be updated.

Example: `UPDATE PROJECT
SET PLOCATION = 'Debere Birhan', BNUM = 5
WHERE PNUMBER=10;`

Updating a primary key value may propagate to the foreign key values of records in other relations if such a *referential triggered action* is specified in the referential integrity constraints.

How you update a table name without affecting stored values in the table?

```
sp_RENAME 'OldTable_name', 'Newtable_name'
```

Deleting rows in the table

The DELETE statement removes rows from a table.

Syntax: `DELETE FROM <Table_Name> [WHERE <search-condition>];`

The search condition specifies which rows in the table are to be deleted. If no search condition is specified, all rows will be deleted (the table become empty, but not dropped).

Example: Delete all countries that begin with the letter 'D' from the COUNTRIES table:

```
Delete from countries where country LIKE 'D%';
```

Populating a Table with Rows from Another Table

You can place the result set of any query into a new table by using the **SELECT INTO** statement.

- 👉 You can use the **SELECT INTO** statement to create a table and to insert rows into the table in a single operation
- 👉 Use the **SELECT INTO** statement to populate new tables in a database with imported data from another table.

Syntax: `SELECT <select_list> INTO <new_table-name>
FROM <sources_table_name>
WHERE <search_condition>`

Example: `select fname, lname, ID into special_Table from EMPLOYEE where Gender = 'female';`

Note: - The SELECT INTO statement selects data from one table and inserts it into a different table.

- The SELECT INTO statement is most often used to create backup copies of tables.

1.4 Combining table Expressions

With the help of **set** operators, the results of individual table expressions can be combined. This type of combination is called **UNION**. SQL supports other set operators besides the UNION operator.

Here is the complete list:

- ✂ UNION
- ✂ UNION ALL
- ✂ INTERSECT
- ✂ INTERSECT ALL
- ✂ EXCEPT
- ✂ EXCEPT ALL

1.4.1 Combining tables with union

The **UNION** operator is used to combine the result-set of two or more SELECT statements.

- ✓ Each SELECT statement within the UNION must have the same number of columns and similar data types.
- ✓ Also, the columns in each SELECT statement must be in the same order.

Syntax: `SELECT <column_name_list> FROM <table_name> UNION SELECT <column_name_list>
FROM <table_name>`

Example: `select fname, ID from EMPLOYEE union select dname, dnumber from DEPARTEMENT`

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

1.4.2 Rules for using UNION

The following rules must be applied to use the UNION operator:

- ✎ The SELECT clauses of all relevant table expressions must have the same number of expressions
- ✎ Each SELECT statement within the UNION must have similar data types. If this applies, the table expressions are union compatible.
- ✎ An ORDER BY clause can be specified only after the last table expression. The sorting is performed on the entire end result; after all intermediate results have been combined.
- ✎ The SELECT clauses should not contain DISTINCT because SQL automatically removes duplicate rows when using UNION.

1.4.3 Combining with INTERSECT

INTERSECT returns all rows that are both in the result of query1 and in the result of query2. Duplicate rows are eliminated unless ALL is specified.

If two table expressions are combined with the INTERSECT operator, the end result consists of those rows that appear in the results of both table expressions.

Example: `SELECT * FROM EMPLOYEE INTERSECT SELECT * FROM PROJECT;`

- Just as with the UNION operator, duplicate rows are automatically removed from the result.

1.4.4 Combining with EXCEPT

EXCEPT returns all rows those are in the result of query1 but not in the result of query2. Again, duplicates are eliminated unless ALL is specified.

If two table expressions are combined with the EXCEPT operator, the end result consists of only the rows that appear in the result of the first table expression but do not appear in the result of the second.

Example: `SELECT * FROM EMPLOYEE EXCEPT SELECT * FROM PROJECT;`

Just as with the UNION operator, duplicate rows are automatically removed from the result.

1.5 Keeping duplicate rows

All previous examples made it clear that duplicate rows are automatically removed from the end result if one of the set operators UNION, INTERSECT, or EXCEPT is used. Removing duplicate rows can be suppressed by using the ALL version of these operators. We illustrate this with the UNION ALL operator.

If two table expressions are combined with the UNION ALL operator, the end result consists of the resulting rows from both of the table expressions. The only difference between UNION and UNION ALL is that when you use UNION, the duplicate rows are automatically removed, and when you use UNION ALL, they are kept.

1.6 Set operators and NULL values

SQL automatically removes duplicate rows from the result if the set operators `UNION`, `INTERSECT`, and `EXCEPT` are specified. That is why the following (somewhat peculiar) `SELECT` statement produces only one row, even if both individual table expressions have one row as their intermediate result:

```
SELECT  PLAYERNO, LEAGUENO
FROM    PLAYERS
WHERE   PLAYERNO = 27
UNION
SELECT  PLAYERNO, LEAGUENO
FROM    PLAYERS
WHERE   PLAYERNO = 27
```

1.7 Combining multiple set operators

We have already seen a few examples in which multiple set operators are used within a single `SELECT` statement. Here is another example.

Get the numbers of each player who incurred at least one penalty and who is not a captain; add the numbers of the players who live in Eltham.

```
SELECT  PLAYERNO FROM PENALTIES EXCEPT
SELECT  PLAYERNO FROM TEAMS UNION
SELECT  PLAYERNO FROM PLAYERS
WHERE   TOWN = 'Eltham'
```

1.8 Set operator and Theory

We conclude this chapter with a rather theoretical discussion of set operators. We give a number of rules for working with multiple different set operators within one `SELECT` statement. All the rules are based on general rules (laws) that apply to mathematical operators and set theory. We define and explain each of these rules, and we use the following symbols and definitions:

- ✓ The symbol T_i represents the result of a random table expression (i is 1, 2, or 3).
- ✓ For each T_i , it holds that the `SELECT` clauses are union compatible.
- ✓ The symbol $T\emptyset$ represents the empty result of a table expression.
- ✓ The symbol \cup represents the `UNION` operator.
- ✓ The symbol \cap represents the `INTERSECT` operator.
- ✓ The symbol $-$ represents the `EXCEPT` operator.
- ✓ The symbol \cup^A represents the `UNION ALL` operator.
- ✓ The symbol \cap^A represents the `INTERSECT ALL` operator.
- ✓ The symbol $-^A$ represents the `EXCEPT ALL` operator.
- ✓ The symbol $=$ means “is equal to.”
- ✓ The symbol \neq means “is not always equal to.”
- ✓ The symbol θ represents a random set operator.
- ✓ The symbol \emptyset represents an empty result.

LO2. Write SQL statements that use functions

2.1 Introduction

In SQL Server, you can design your own functions to supplement and extend the system supplied (built-in) functions. A user-defined function takes zero, or more, input parameters and returns either a scalar value or a table.

Input parameters can be any data type except timestamp, cursor, or table.

2.2 Select clause and Aggregation function

Introduction

The SQL **SELECT statement** queries data from tables in the database.

The statement begins with the SELECT keyword. The basic SELECT statement has 3 clauses:

- ❖ SELECT
- ❖ FROM
- ❖ WHERE

📖 The SELECT clause specifies the table columns that are retrieved.

📖 The FROM clause specifies the tables accessed.

📖 The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

An **aggregation function** is a function that performs a computation on a set of values rather than on a single value. **Example:** finding the average or mean of a list of numbers is an aggregate function.

The aggregation functions are **AVG, COUNT, MAX, MIN, SUM, STDEV, STDEVP, VAR, and VARP**

The syntax of an aggregation function is illustrated as:

Aggregation-function ([ALL | DISTINCT] expression)

E.g.: **SELECT SUM** (Employee.salary) as Total_Salary **FROM** Employee **WHERE** Employee.Dnum =20

SELECT min(EMPLOYEE .salary)'min salary',**MAX** (EMPLOYEE .salary)'max salary' **FROM** EMPLOYEE

This aggregation computes the total salary for department 20.

Aggregate functions are functions that are used to get summary values.

2.3 Selecting ALL Columns (SELECT *)

Asterisk (*) is used to get all the columns of a particular table.

Example: the SQL **select *** from Employee will retrieve the entire copy of the employee table.

2.4 Expression in the select clause

Syntax: SELECT [ALL|DISTINCT]<select_list> FROM {<table_source >}

WHERE <search_condition> <search_condition> uses expression:

- ✎ Comparison Operators =, <, >, >=, <=, and <>
- ✎ String comparisons LIKE and NOT LIKE
- ✎ Logical Operators: combination of conditions AND, OR
- ✎ Logical Operators: negations NOT
- ✎ Range of values BETWEEN and NOT BETWEEN
- ✎ List of values IN and NOT IN
- ✎ Unknown values IS NULL and IS NOT NULL.

2.5 Removing duplicate rows with DISTINCT when two rows are equal

To eliminate the duplicates from the result set, we use the key word DISTINCT.

Example: **SELECT distinct** Publisher **FROM** BOOK

2.6 Introduction to aggregation function

- ☞ The **count function** Returns the number of items in *expression*. The data type returned is of type *int*.

Syntax: COUNT ([ALL | DISTINCT] <expression> | *)

Example: `select COUNT(*), AVG(Employee.Salary) from dbo. Employee`

- ☞ With the exception of the COUNT (*) function, all aggregate functions return a NULL if no rows satisfy the WHERE clause. The COUNT (*) function returns a value of zero if no rows satisfy the WHERE clause.

- ☞ The **MAX function** Returns the maximum value from *expression*. Max ignores any *NULL* values.

Syntax: MAX ([ALL | DISTINCT] <expression>)

Example: `select MAX(Employee.Salary) from dbo. Employee`

- ☞ The **MIN function** Returns the smallest value from *expression*. Min ignores any *NULL* values.

Syntax: MIN ([ALL | DISTINCT] <expression>)

Example: `select MIN(Employee.Salary) from dbo. Employee`

- ☞ The **SUM function** Returns the total of all values in *expression*. Sum ignores any *NULL* values.

Syntax: SUM ([ALL | DISTINCT] <expression>)

Example: `select SUM(Employee.Salary) from dbo. Employee`

- ☞ The **AVERAGE function** Returns the average of the values in *expression*. The *expression* must contain numeric values. Null values are ignored.

syntax: `AVG ([ALL | DISTINCT] <expression>)`

Example: `select ID, avg(Employee.Salary) from dbo. Employee`

- ☞ **The Variance and Standard Deviation function**

- **STDEV**: Returns the standard deviation of all values in *expression*. Stdev ignores any *NULL* values. **Syntax:** STDEV(<expression>)

Example: `select STDEV(Employee.Salary) from dbo. Employee`

- **STDEVP**: Returns the standard deviation for the population of all values in *expression*. Stdevp ignores any *NULL* values.

Syntax: STDEVP(<expression>)

Example: `select STDEVP(Employee.Salary) from dbo. Employee`

- **VAR**: Returns the variance of all values in *expression*. Var ignores any *NULL* values. **syntax:** VAR(<expression>)

Example: `select VAR(Employee.Salary) from dbo. Employee`

- **VARP**: Returns the variance for the population of all values in *expression*. Varp ignores any *NULL* values. **syntax:** VARP(<expression>)

Example: `select VARP(Employee.Salary) from dbo. Employee`

LO3. Write SQL statements that use aggregation and filtering

3.1 Aggregating data by multiple columns using “group by”

When an aggregate function is executed, SQL Server summarizes values for an entire table or for groups of columns within the table, producing a single value for each set of rows for the specified columns.

- ✎ You can use aggregate functions with the SELECT statement or in combination with the **GROUP BY** clause
- ✎ Use the **GROUP BY** clause on columns or expression to organize rows into groups and to summarize those groups. The **GROUP BY** clause groups rows on the basis of similarities between them.

When you use the **GROUP BY** clause, consider the following facts and guidelines:

- ✎ SQL Server returns only single rows for each group that you specify; it does not return detail information.
- ✎ All columns that are specified in the GROUP BY clause must be included in the select list.
- ✎ If you include a WHERE clause, SQL Server groups only the rows that satisfy the search conditions.
- ✎ Do not use the GROUP BY clause on columns that contain multiple null values.

Example: For each department, retrieve the department name, the number of employees in the department, and their average salary.

```
SELECT DEPARTEMENT.Dname , COUNT (*) 'number of employee', AVG (EMPLOYEE.salary) 'average salary'
FROM EMPLOYEE,DEPARTEMENT
where EMPLOYEE.dnum =DEPARTEMENT.Dnumber
GROUP BY DEPARTEMENT.Dname
```

3.2 Sorting aggregated data in the query output:

SQL allows the user to sort rows in the result set in ascending (ASC) or descending (DESC) order using **ORDER BY** clause. Sort is in ascending order by default

- You can sort by column names, computed values, or expressions

Example: Retrieve a list of employees and ordered them alphabetically by their department name, last name, first name.

```
SELECT EMPLOYEE.ID, EMPLOYEE.Fname, EMPLOYEE.Lname, EMPLOYEE.salary, DEPARTEMENT.Dname
FROM DEPARTEMENT,EMPLOYEE
WHERE EMPLOYEE.dnum=DEPARTEMENT .Dnumber
ORDER BY DNAME, LNAME, FNAME;
```

3.3 Filtering aggregated data using the “having” clause

Use the **HAVING** clause on columns or expressions to set conditions on the groups included in a result set.

When you use the HAVING clause, consider the following facts and guidelines:

- ✎ Use the HAVING clause only with the GROUP BY clause to restrict the grouping.
- ✎ Using the HAVING clause without the GROUP BY clause is not meaningful.

Example: For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
SELECT DNAME, Fname, COUNT (*) FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO AND SALARY>40000
GROUP BY DNAME DESC, Fname ASC
HAVING COUNT (*) > 5;
```

LO4: Write and execute SQL Queries

4.1 Single and nested queries

What is query?

A **query** is a request for information from a database (Queries are Questions).

Single query is a Single Block query.

Example: `SELECT distinct salary FROM Employee where Gender = 'female'`

A SQL **nested query** is a SELECT query that is nested inside a SELECT, UPDATE, INSERT, or DELETE SQL query.

Nested Queries are queries that contain another complete SELECT statements nested within it, that is, in the WHERE clause.

- 👉 The nested SELECT statement is called an “inner query” or an “inner SELECT.”
- 👉 The main query is called “outer SELECT” or “outer query.”
- 👉 The use of nested query in this case is to avoid explicit coding of JOIN which is a very expensive database operation and to improve query performance.

Example: `SELECT ID, LNAME, FNAME FROM EMPLOYEE
WHERE (SELECT COUNT (*) FROM DEPENDENT WHERE DEPENDENT.EmpID =EMPLOYEE.ID) >= 2`

4.2 Subqueries

A subquery is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery.

- 👉 Subquery is an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
- 👉 You use subqueries to break down a complex query into a series of logical steps and, as a result, to solve a problem with single statements.
- 👉 Each select statement in the subquery has its own:
 - ✓ select list
 - ✓ where clause

Example: For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than \$40,000.

```
SELECT DNUMBER, COUNT (*) FROM DEPARTMENT, EMPLOYEE  
WHERE DNUMBER=DNO AND SALARY>40000 AND DNO IN (SELECT DNO FROM EMPLOYEE  
GROUP BY DNO HAVING COUNT (*) > 5) GROUP BY DNUMBER;
```

4.3 Operators in sub queries

The IN, ALL, and ANY operators in subquery.

👉 **IN operator**

The IN operator is an operator that allows you to specify multiple values in a WHERE clause.

A row from a table satisfies a condition with the IN operator if the value of a particular column occurs in a set of expressions. The expressions in such a set entered one by one by a user.

Syntax to use **IN** operator: `SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)`

Example: `SELECT * FROM Persons WHERE LastName IN ('Aster', 'Maru')
SELECT * FROM employee
WHERE Dnum IN (SELECT Dnum FROM DEPARTEMENT WHERE Dname = 'Research');`

👉 Any operator

ANY operator is an operator that compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row. **ANY** must be preceded by comparison operators.

Syntax to use **Any** operator:

```
SELECT column_name(s)
FROM table_name
WHERE column_name =any(value1,value2,...)
```

Example:

```
SELECT ID ,fname,lname FROM EMPLOYEE
WHERE Dnum =any(SELECT COUNT(*) FROM DEPARTEMENT
WHERE Dnumber='ict001')
```

👉 All operators

ALL operator is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list or results from a query. The **ALL** must be preceded by the comparison operators and evaluates to **TRUE** if the query returns no rows. For example, **ALL** means greater than every value, means greater than the maximum value. Suppose **ALL** (1, 2, 3) means greater than 3.

Syntax to use **All** operator:

```
SELECT column_name(s)
FROM table_name
WHERE column_name =all(value1,value2,...)
```

Enample:

```
SELECT LNAME, FNAME FROM EMPLOYEE
WHERE SALARY> ALL (SELECT SALARY FROM EMPLOYEE WHERE Dnum=5);
```

👉 SOME operators

SOME operator is the same as **ANY** operator (**SOME** and **ANY** perform the same function).

SOME compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row.

SOME must match at least one row in the subquery and must be preceded by comparison operators.

Example:

```
SELECT ID,Fname,Lname FROM EMPLOYEE
WHERE ID =SOME(SELECT EmpID FROM DEPENDENTED WHERE Gender='Female');
```

👉 EXISTS operator

The **EXIST** operator checks the existence of a result of a subquery. The **EXISTS** subquery tests whether a subquery fetches at least one row. When no data is returned then this operator returns '**FALSE**'.

- ✓ A valid **EXISTS** subquery must contain an outer reference and it must be a correlated subquery.
- ✓ You can use the **EXISTS** and **NON EXISTS** operators to determine whether data exists in a list of values.
- ✓ Use the **EXISTS** and **NOT EXISTS** operators with correlated subqueries to restrict the result set of an outer query to rows that satisfy the subquery.

Example:

```
SELECT DepID ,Fname ,Gender FROM DEPENDENTED
WHERE exists(SELECT ID FROM EMPLOYEE WHERE EMPLOYEE.ID=DEPENDENTED.EmpID and
DEPENDENTED.Gender ='Female')
```

You can use the **EXISTS** operator with the **group by** as wel as **order by** clause to determine whether data exists in a list of values.

Example:

```
SELECT dnum,COUNT (*) 'Number of employee who has dependent' FROM EMPLOYEE
WHERE salary<3000 and exists (SELECT EmpID FROM DEPENDENTED
WHERE EMPLOYEE.ID =DEPENDENTED.EmpID
GROUP BY EmpID)

GROUP BY Dnum
ORDER BY Dnum
```

👉 Having clause

Use the HAVING clause on columns or expressions to set conditions on the groups included in a result set. The HAVING clause sets conditions on the GROUP BY clause in much the same way that the WHERE clauses interacts with the SELECT statement.

Example: `SELECT dnum , count (*) 'number of employee' FROM EMPLOYEE
WHERE exists (SELECT dname FROM DEPARTEMENT
WHERE EMPLOYEE.Dnum =DEPARTEMENT.Dnumber)
group by Dnum
having COUNT(*)>2`

4.4 Correlated subqueries

In a SQL database query, a **correlated sub-query** (also known as a synchronized subquery) is a sub-query (a query nested inside another query) that uses values from the outer query in its WHERE clause.

- ✎ Correlated subquery is one that is executed after the outer query is executed. So correlated subqueries take an approach opposite to that of the normal subqueries.
- ✎ In a correlated subquery, the inner query uses information from the outer query and executes once for every row in the outer query.
- ✎ A practical use of a correlated subquery is to transfer data from one table to another.

Syntax for correlated subquery: `select column_list from table_name a
where search_condition (select column_list from table_name b
where a.column_name_a=b.column_name_b)`

Example: find out the name of all EMPLOYEES who has less or equal to two dependent using correlated subquery.

```
select fname from EMPLOYEE a where 2<=(select COUNT(*) from DEPENDENTED b  
where b .EmpID=a.ID )
```