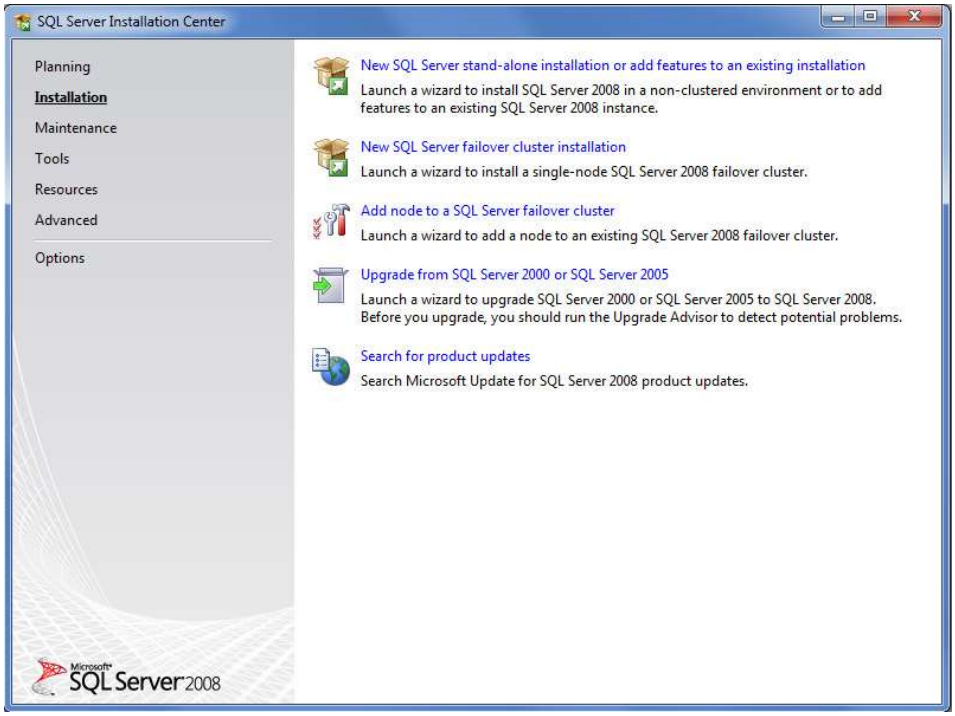
Install SQL Server 2008 Express(Including Management Studio)

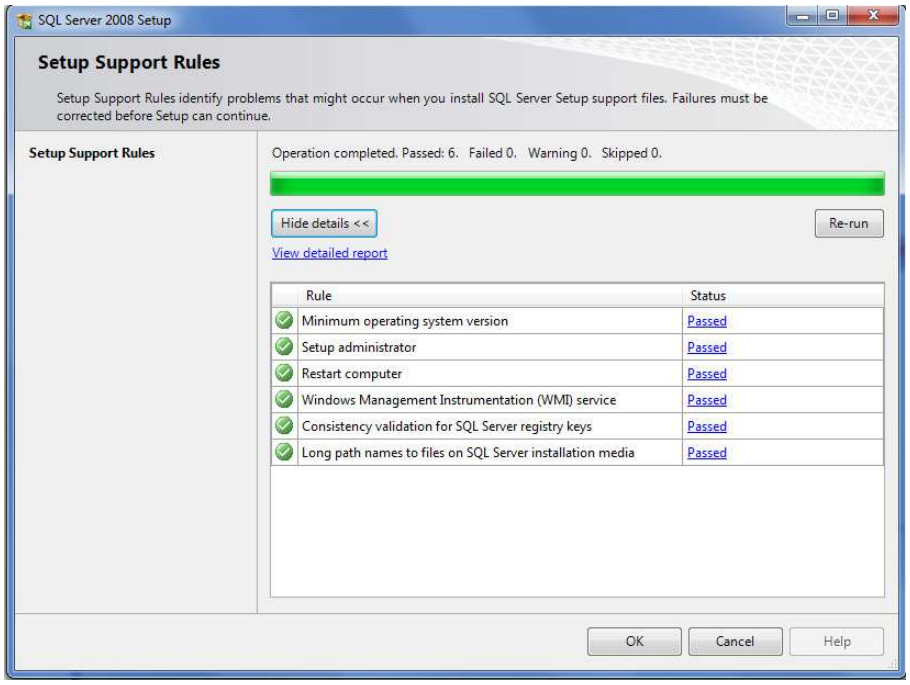
Microsoft SQL Server 2008 Express with Tools is a free, easy-to-use version of the SQL Server Express data platform that includes the graphical management tool: SQL Server Management Studio (SMSS) Express. SQL Server 2008 Express provides rich features, data protection, and fast performance. It is ideal for small server applications and local data stores.

Steps:

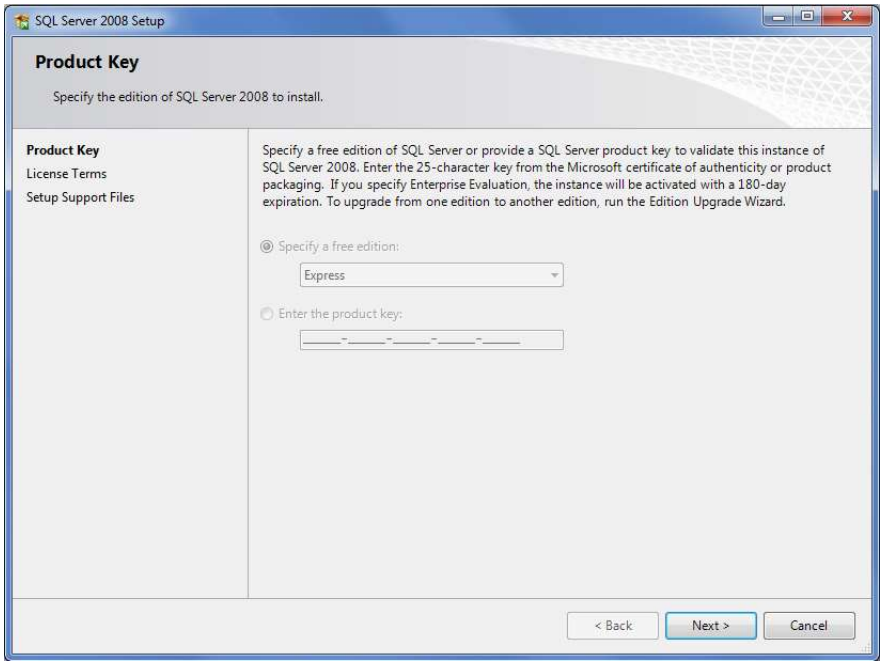
1. On the left column, click on "Installation" Then click on the first link: "new SQL Server stand-alone installation"



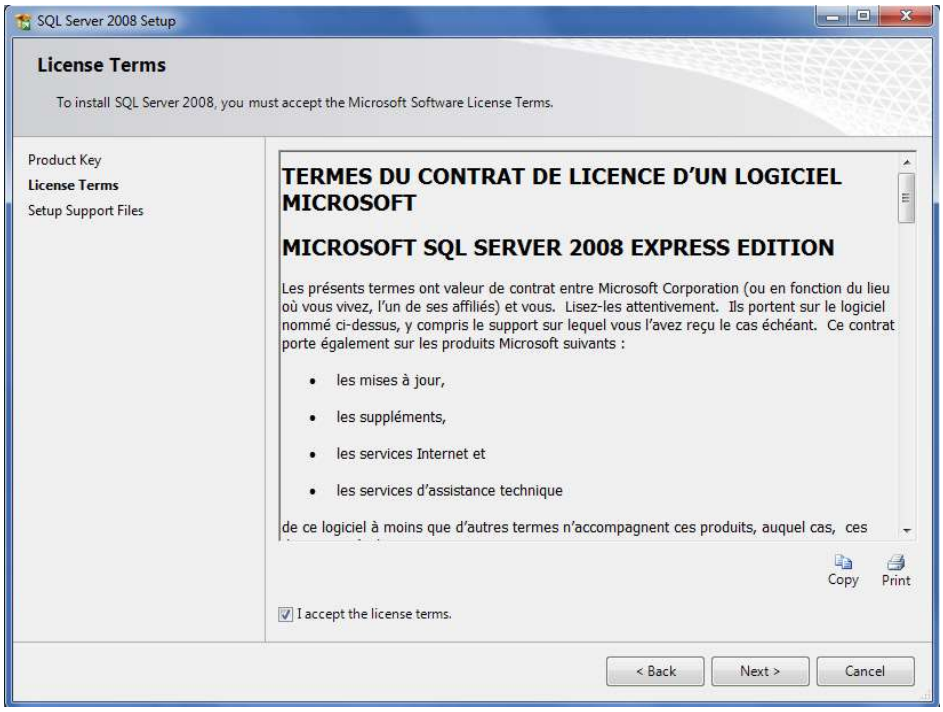
2. Setup support rules reopen. Once all operations completed, click OK



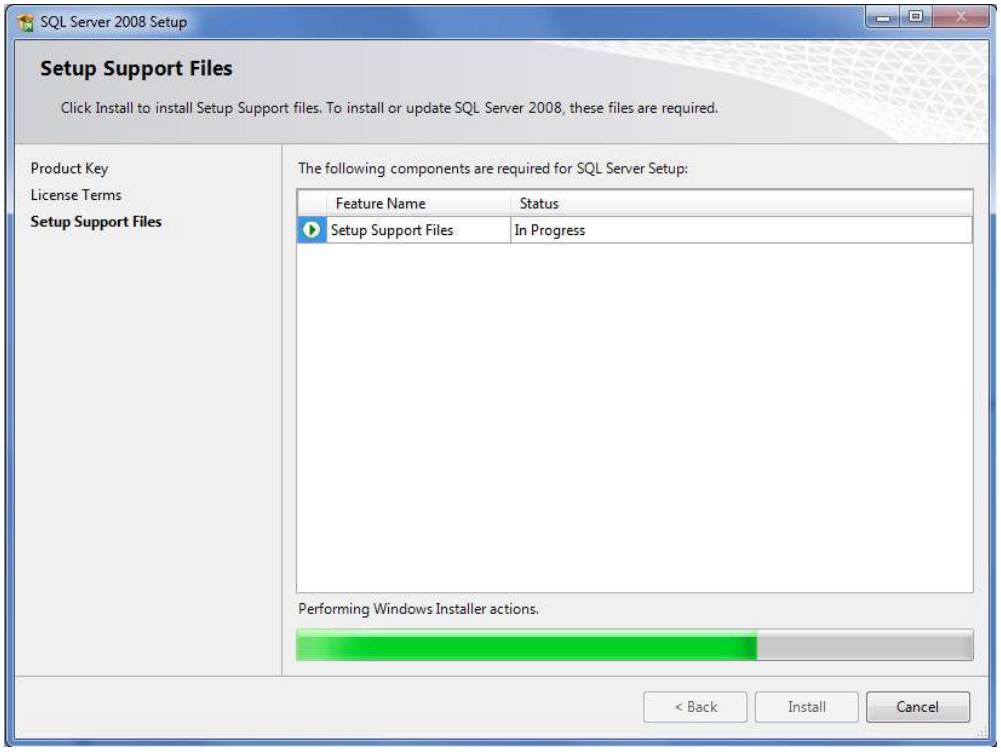
3. The SQL Server 2008 Setup opens with a first "product key" tab As we use a specific free version of SQL Server, no activation key is needed. Simply click "next"



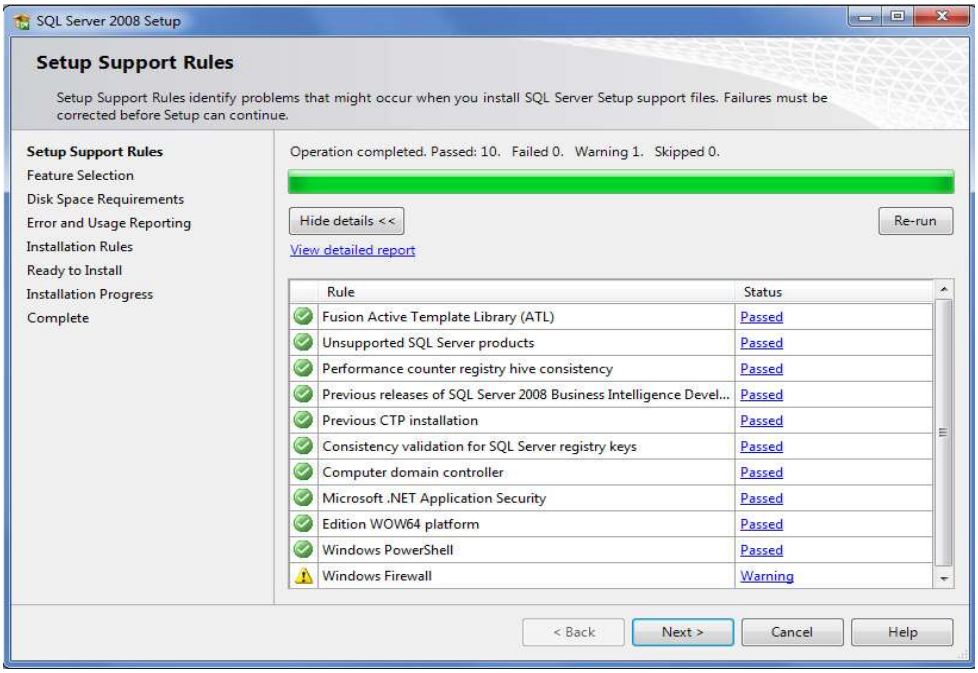
4. Check the box "I accept the licence terms" Click on "Next"



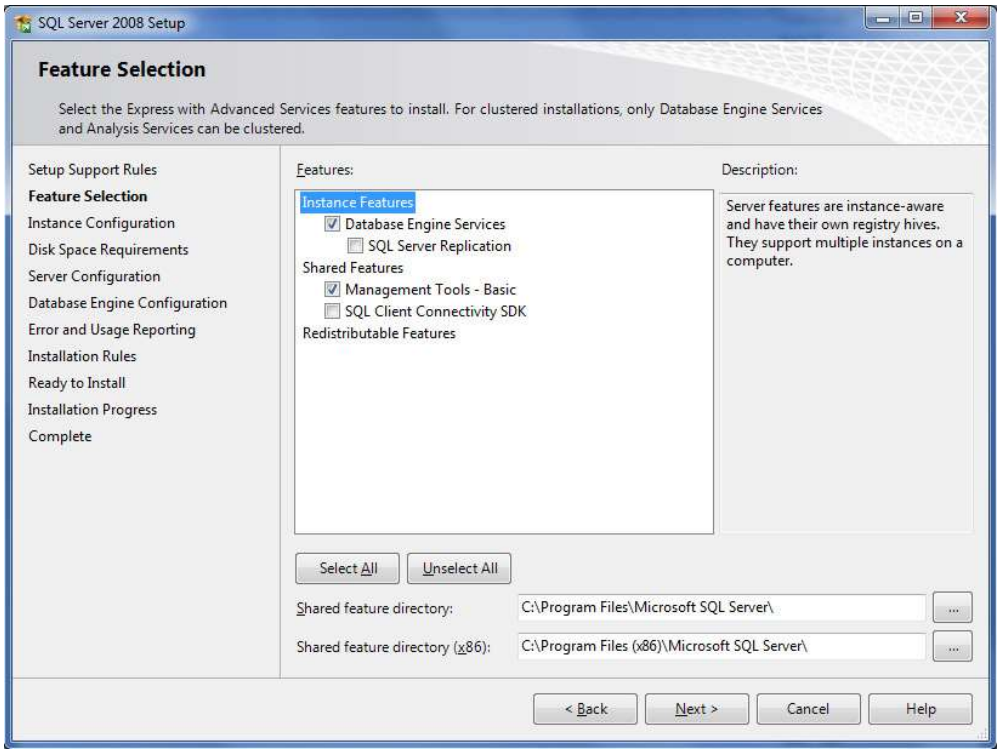
5. The "setup support files" tab opens. Click on "install"



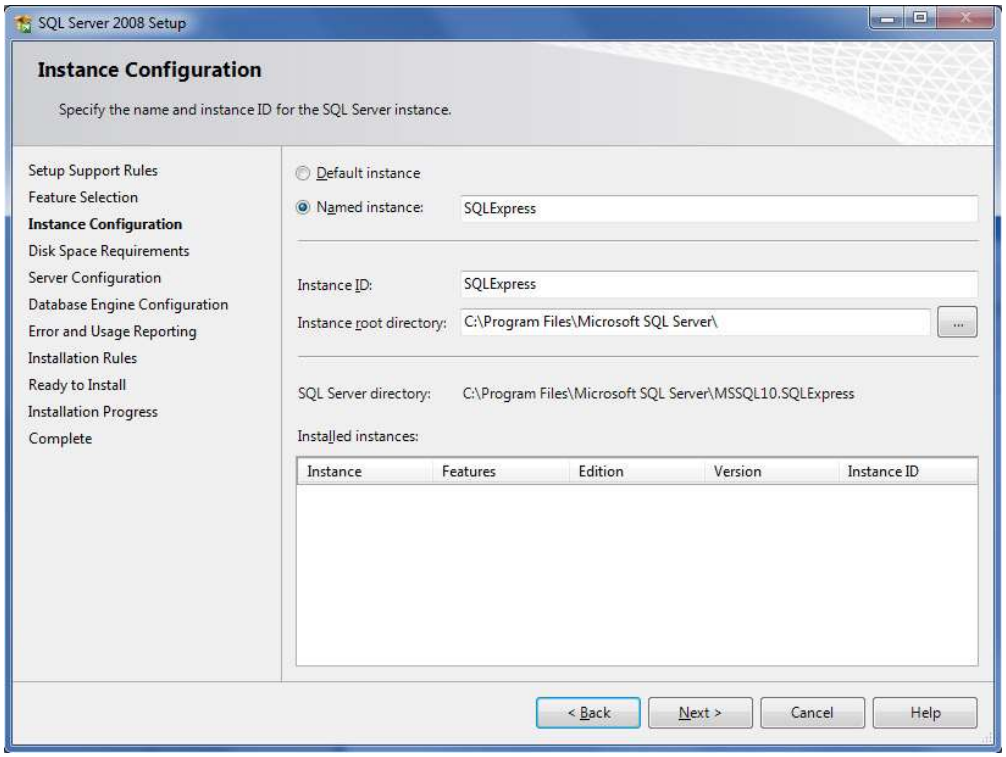
6. Once installed, the Setup Support Rules reopen. Check that all operation passsed, then click "next"



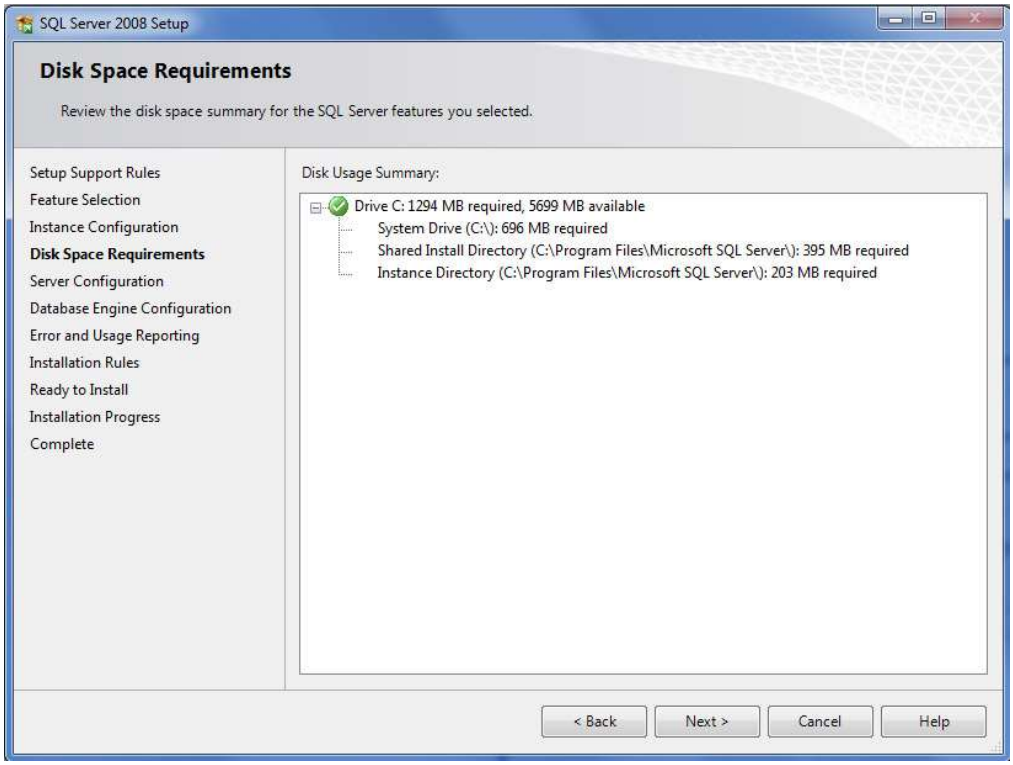
7. On the "features selection" tab choose "Data Engine Services" and "Management tools" (this is the Management Studio Express)check the features directory (no need to change normally) click "next"



8. On the "instance Configuration" check that the parameters are like the screenshot on the left (nothing to change normally) click "Next"

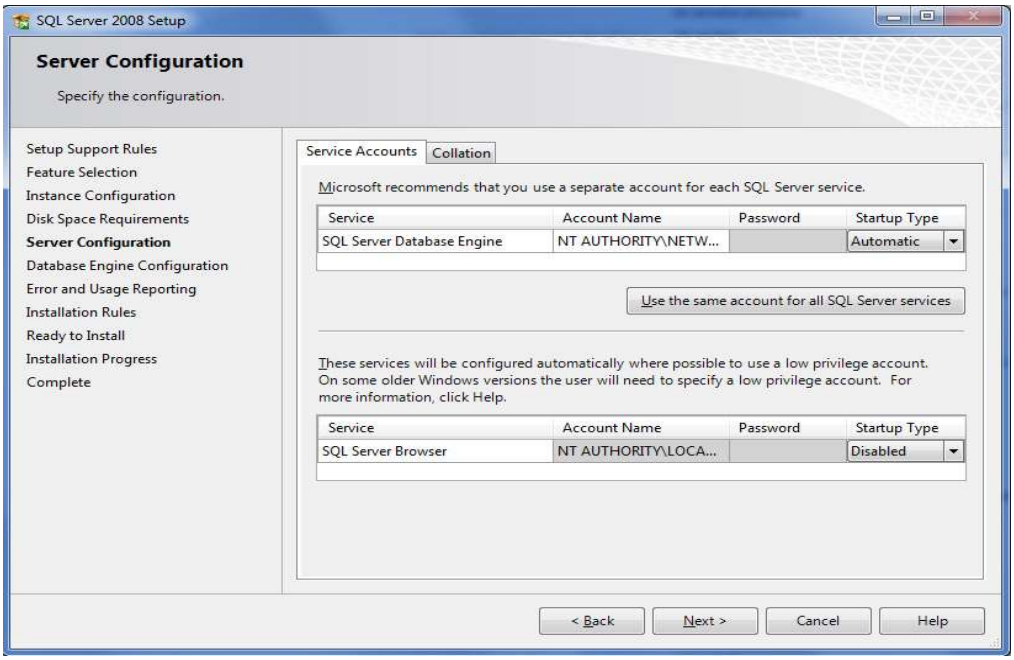


9. Disk space requirements tap -> Click Next



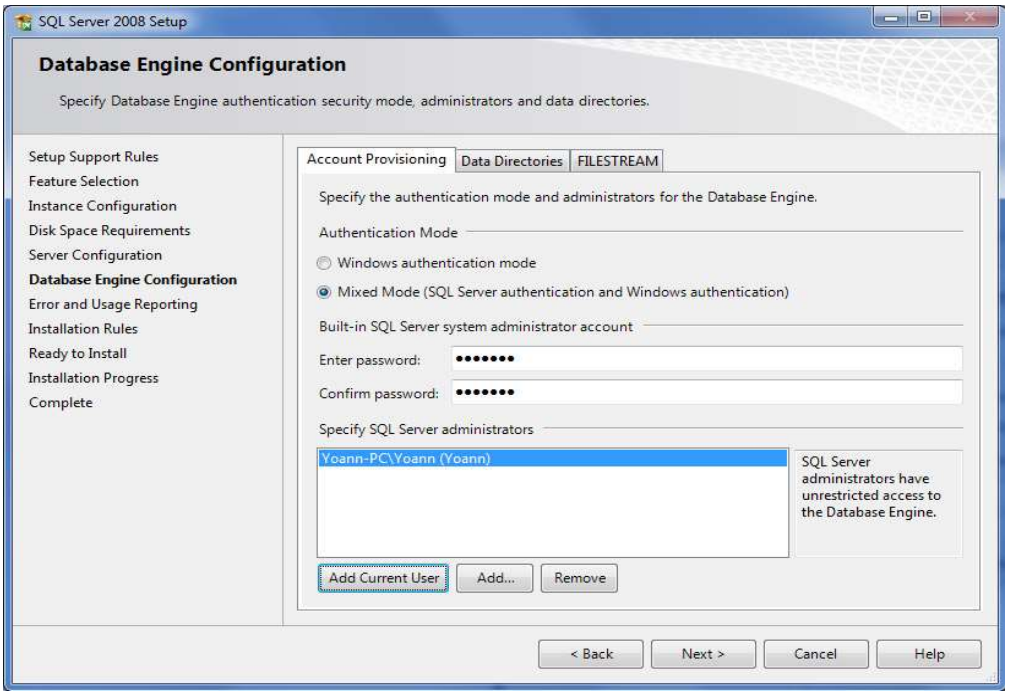
10. **Server Configuration:**

For the first service "SQL Server Database Engine", on the account name, choose the first result of the listbox (in our case NT AUTHORITY\NETWORK). Then click "Next".

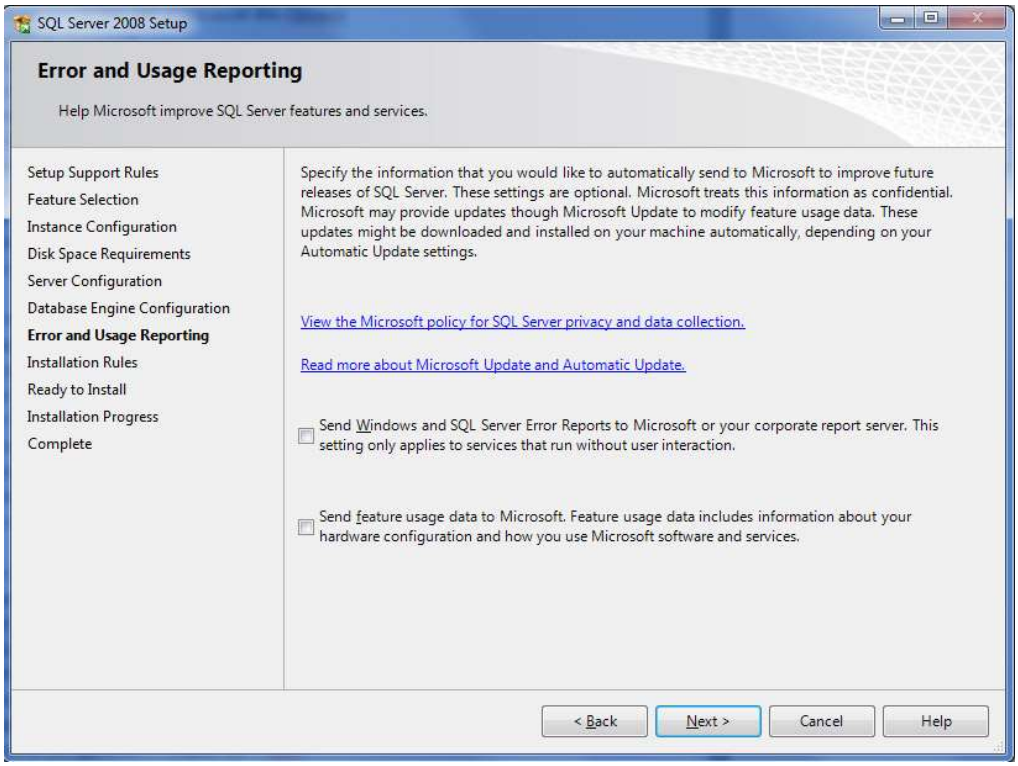


11. **Database Engine Configuration:**

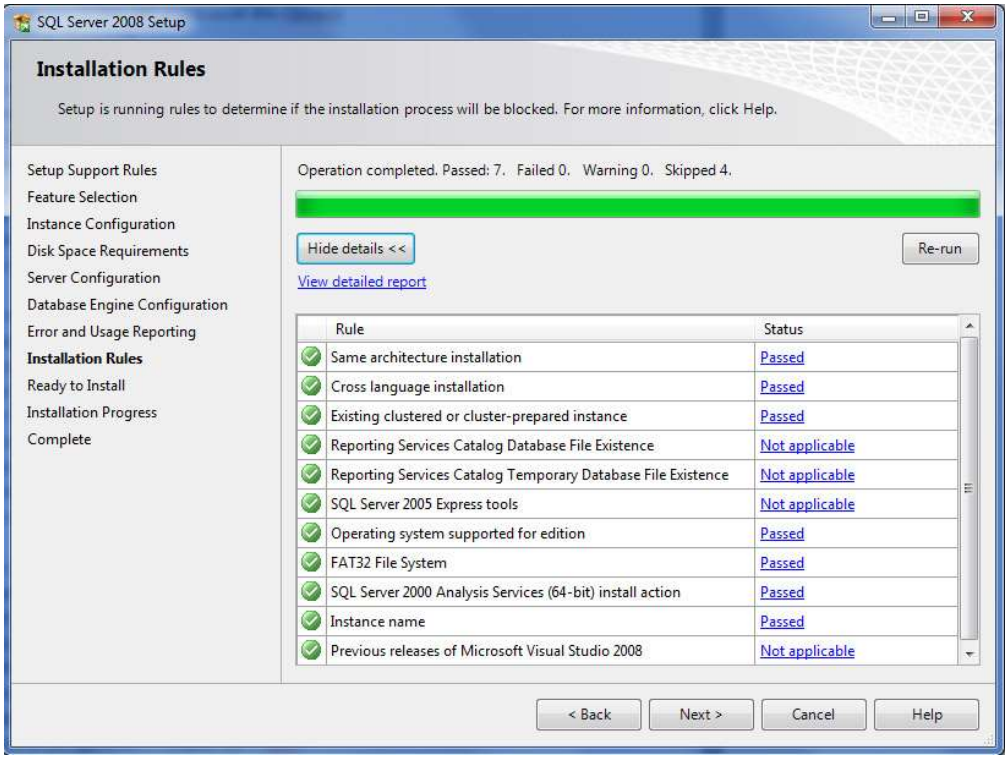
- Chose "Mixed mode" and Enter the password **: p@ssword13**
- Specify SQL Server administrator : Click on "add current user". Click "Next".



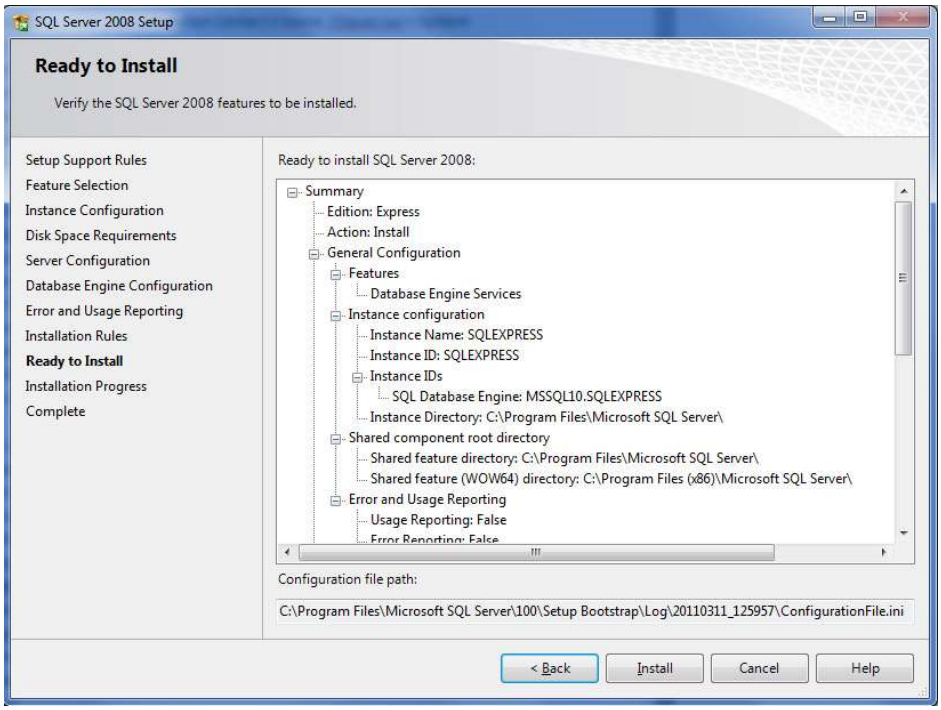
12. Error and Usage Reporting. Click next.



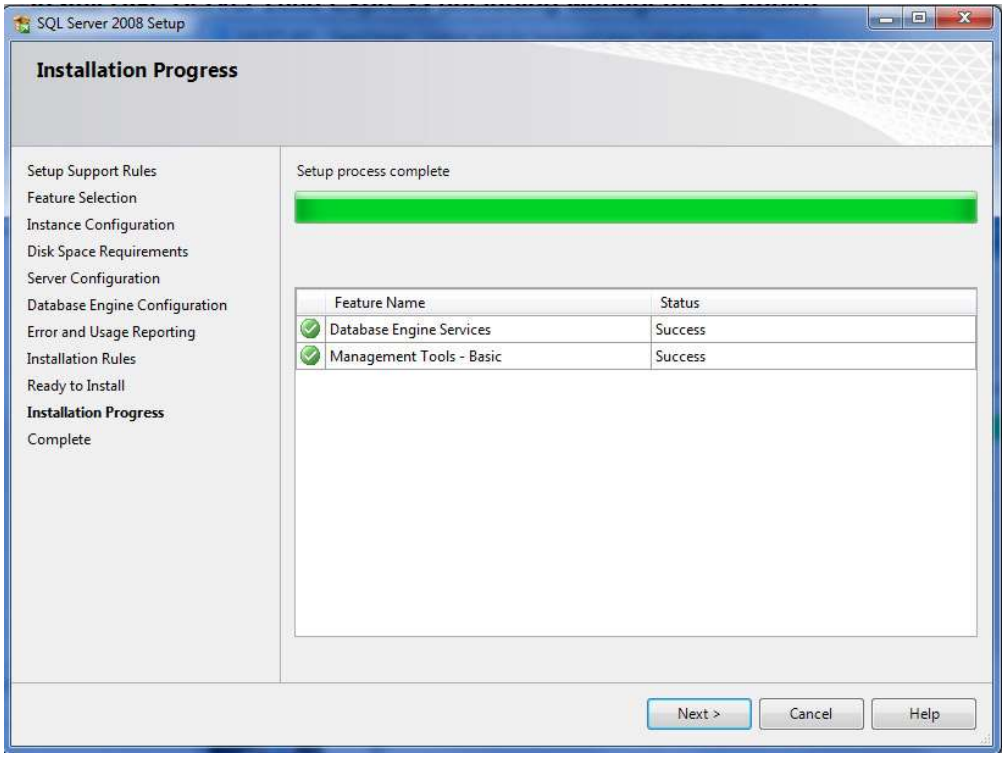
13. Installation Rules : Once Operation completed, click "Next"



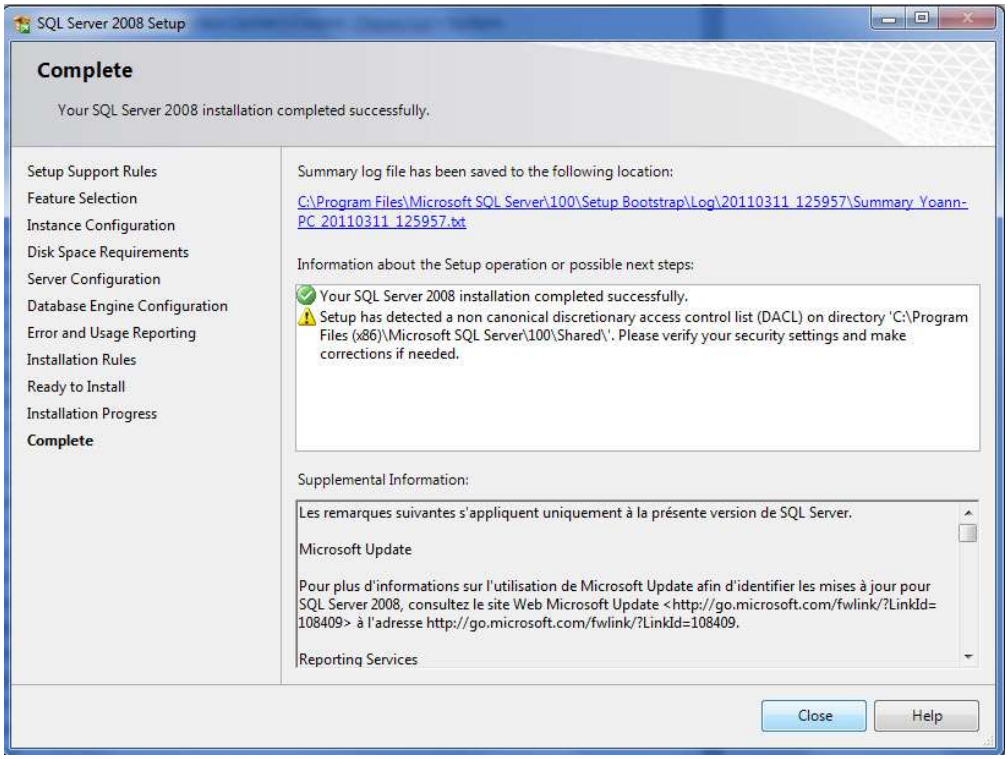
14. Ready to install: Click "Install"



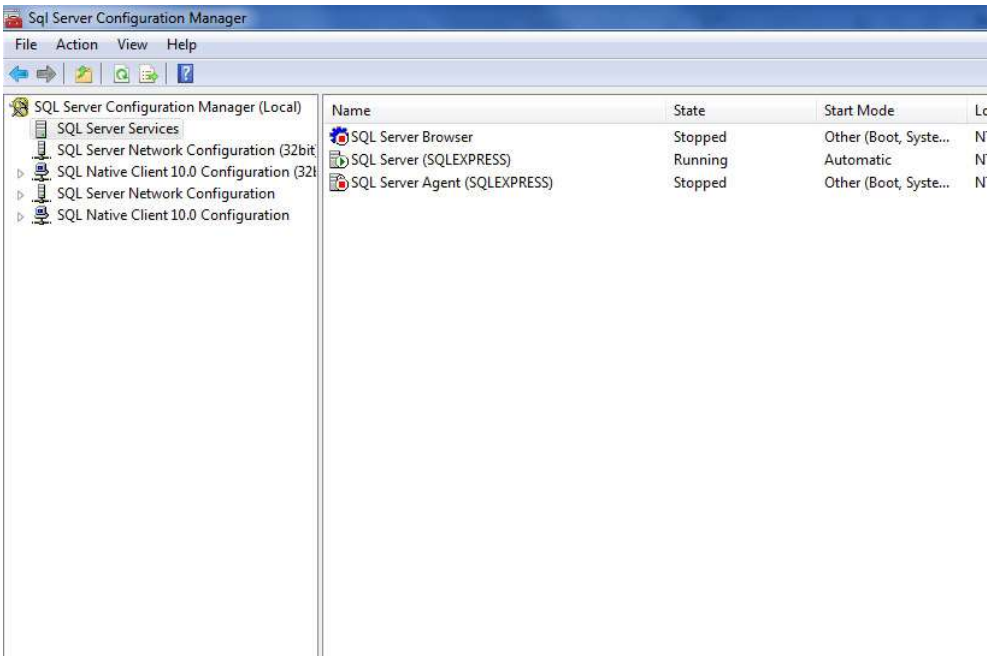
15. Installation Progress: Wait until Installation is finished.Ensure "Database Engine Services" and "Management Tools Basic" are successfully installed. Click "Next".



16. Complete: Once installed, click on "Close"



17. To ensure SQL Express is running correctly, run "SQL Server Configuration Manager" On the left column click on "SQL Server Services On the right column, ensure that SQL Server (SQLEXPRESS) is running correctly (otherwise right-click on the device to start it)



18. The SQL Server Connection Windows appears.

Choose the following parameters:

- Server Type: Database Engine
- Server Name: (Name of your computer)\SQLEXPRESS*
- Authentication: SQL Server Authentication (if you can only see windows authentication, you probably missed to choose "Mixed Mode" at the Database Engine Configuration step
- Login: sa (by default)
- Password: the password you defined at the Database Engine Configuration step (**p@ssword13**)
- You can find the name of your computer on the control panel>System

Database Lab

CSE 3104

Lab-02

1 Introduction to SQL

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

SQL is a standardized computer language that was originally developed by IBM for querying, altering and defining relational databases, using declarative statements.

Query Examples:

```
• insert into STUDENT (Name , Number, SchoolId)
  values ('John Smith', '100005', 1)

• select SchoolId, Name from SCHOOL

• select * from SCHOOL where SchoolId > 100

• update STUDENT set Name='John Wayne' where StudentId=2

• delete from STUDENT where SchoolId=3
```

We have 4 different Query Types: **INSERT**, **SELECT**, **UPDATE** and **DELETE**

What can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database

• SQL can set permissions on tables, procedures, and views

Even if SQL is a standard, many of the database systems that exist today implement their own version of the SQL language. In this document we will use the Microsoft SQL Server as an example.

There are lots of different database systems, or DBMS–Database Management Systems, such as:

• Microsoft SQL Server

    o Enterprise, Developer versions, etc

    o Express version is free of charge

• Oracle

• MySQL (Oracle, previously Sun Microsystems)- MySQL can be used free of charge (open source license), Web sites that use MySQL: YouTube, Wikipedia, Facebook

• Microsoft Access

• IBM DB2

• Sybase

• … lots of other systems

1.1 Data Definition Language (DDL)

The **Data Definition Language (DDL)** manages table and index structure. The most basic items of DDL are the CREATE, ALTER, RENAME and DROP statements:

- **CREATE** creates an object (a table, for example) in the database.
- **DROP** deletes an object in the database, usually irretrievably.
- **ALTER** modifies the structure an existing object in various ways—for example, adding a column to an existing table.

1.2 Data Manipulation Language(DML)

The **Data Manipulation Language (DML)** is the subset of SQL used to add, update and delete data.

The acronym **CRUD** refers to all of the major functions that need to be implemented in a relational database application to consider it complete. Each letter in the acronym can be mapped to a standard SQL statement:

Operation	SQL	Description
Create	INSERT INTO	inserts new data into a database
Read (Retrieve)	SELECT	extracts data from a database
Update	UPDATE	updates data in a database
Delete (Destroy)	DELETE	deletes data from a database

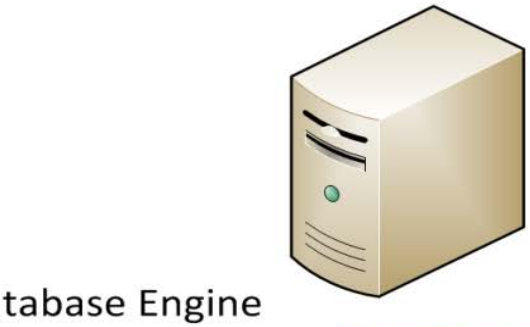
2 Introduction to SQL Server

Microsoft is the vendor of SQL Server. The newest version is “SQL Server 2012”.

We have different editions of SQL Server, where SQL Server Express is free to download and use.

SQL Server uses T-SQL (Transact-SQL). T-SQL is Microsoft's proprietary extension to SQL. T-SQL is very similar to standard SQL, but in addition it supports some extra functionality, built-in functions, etc. T-SQL expands on the SQL standard to include procedural programming, local variables, various support functions for string processing, date processing, mathematics, etc.

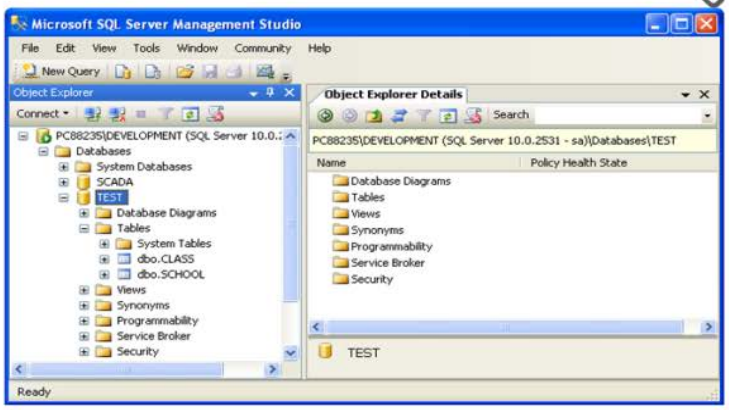
SQL Server consists of a **Database Engine** and a **Management Studio** (and lots of other stuff which we will not mention here). The Database engine has no graphical interface - it is just a service running in the background of your computer (preferable on the server). The Management Studio is graphical tool for configuring and viewing the information in the database. It can be installed on the server or on the client (or both).



Database Engine

A Service running on the computer in the background

Management Studio

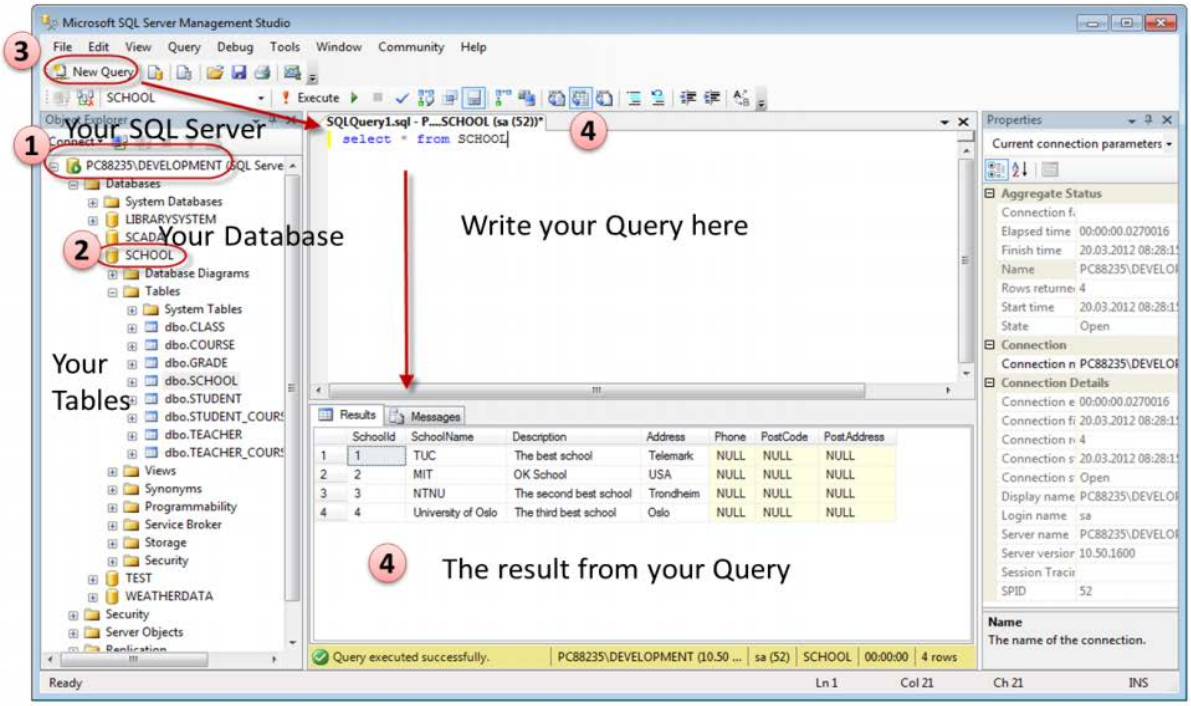


A Graphical User Interface to the database used for configuration and management of the database

2.1 SQL ServerManagement Studio

SQL Server Management Studio is a GUI tool included with SQL Server for configuring, managing, and administering all components within Microsoft SQL Server. The tool includes both script editors and graphical tools that work with objects and features of the server. As mentioned earlier, version of SQL Server Management Studio is also available for SQL Server Express Edition, for which it is known as SQL Server Management Studio Express.

A central feature of SQL Server Management Studio is the Object Explorer, which allows the user to browse, select, and act upon any of the objects within the server. It can be used to visually observe and analyze query plans and optimize the database performance, among others. SQL Server Management Studio can also be used to create a new database, alter any existing database schema by adding or modifying tables and indexes, or analyze performance. It includes the query windows which provide a GUI based interface to write and execute queries.

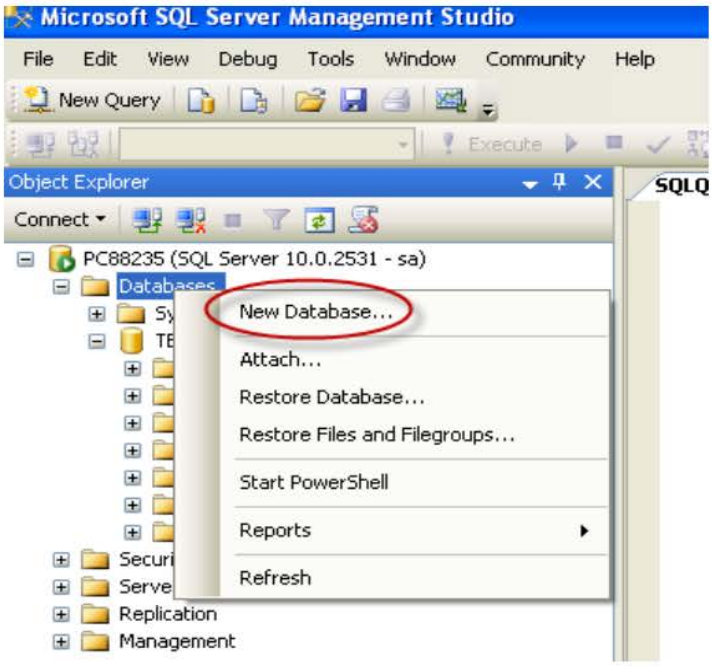


When creating SQL commands and queries, the “Query Editor” (select “New Query” from the Toolbar) is used (shown in the figure above).

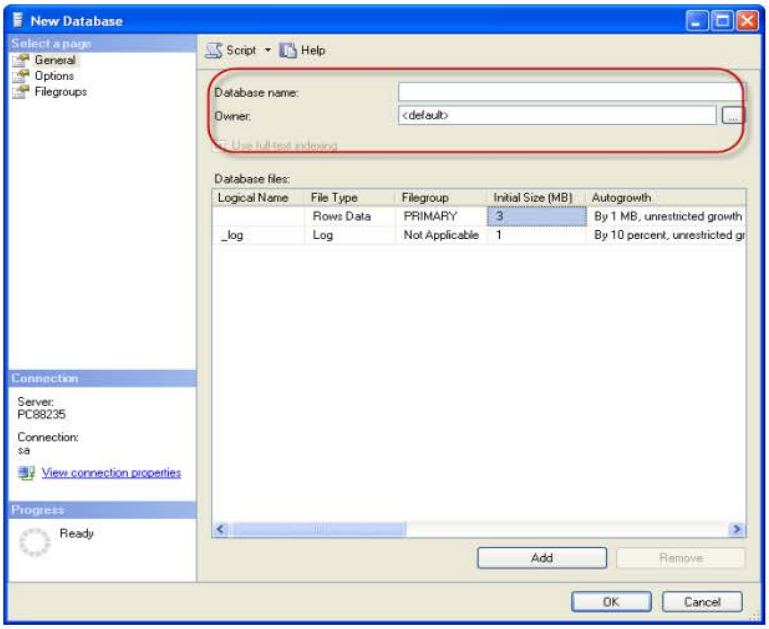
With SQL and the “Query Editor” we can do almost everything with code, but sometimes it is also a good idea to use the different Designer tools in SQL to help us do the work without coding (so much).

2.1.1 Create a new Database

It is quite simple to create a new database in Microsoft SQL Server. Just right-click on the "Databases" node and select "New Database…"

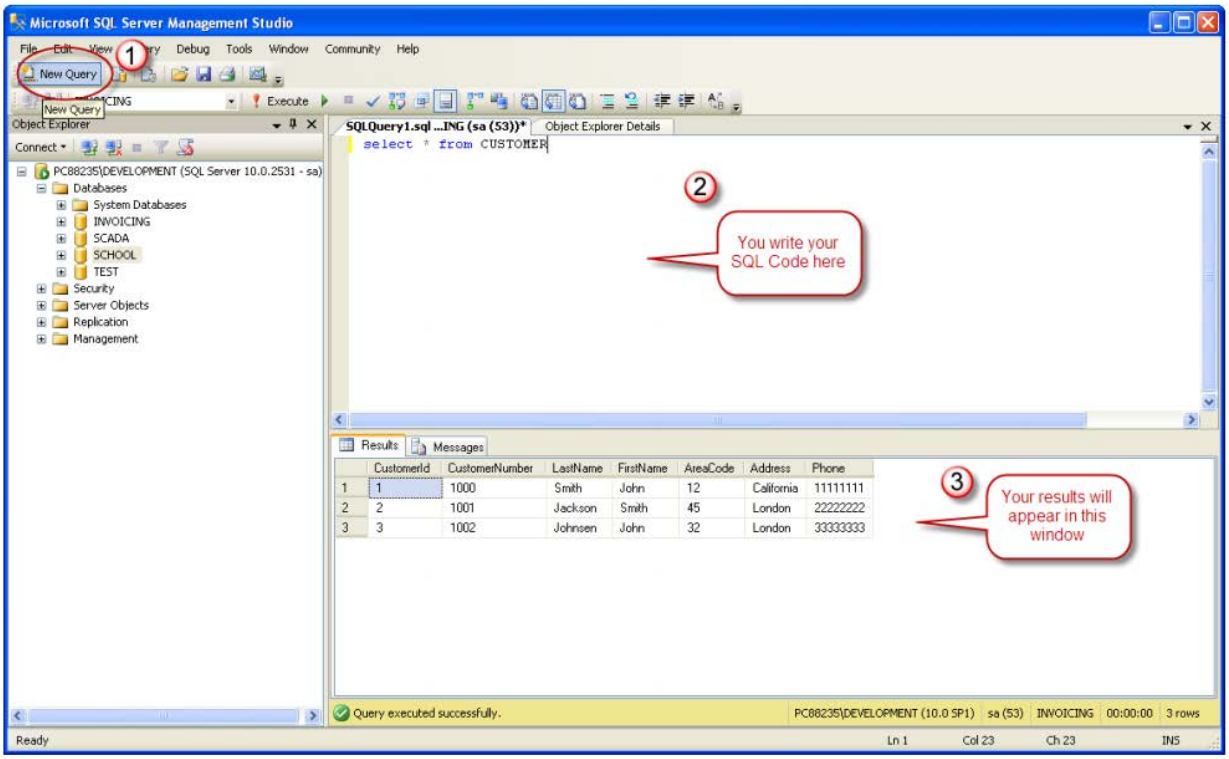


There are lots of settings you may set regarding your database, but the only information you must fill in is the name of your database:



2.1.2 Queries

In order to make a new SQL query, select the “New Query” button from the Toolbar.

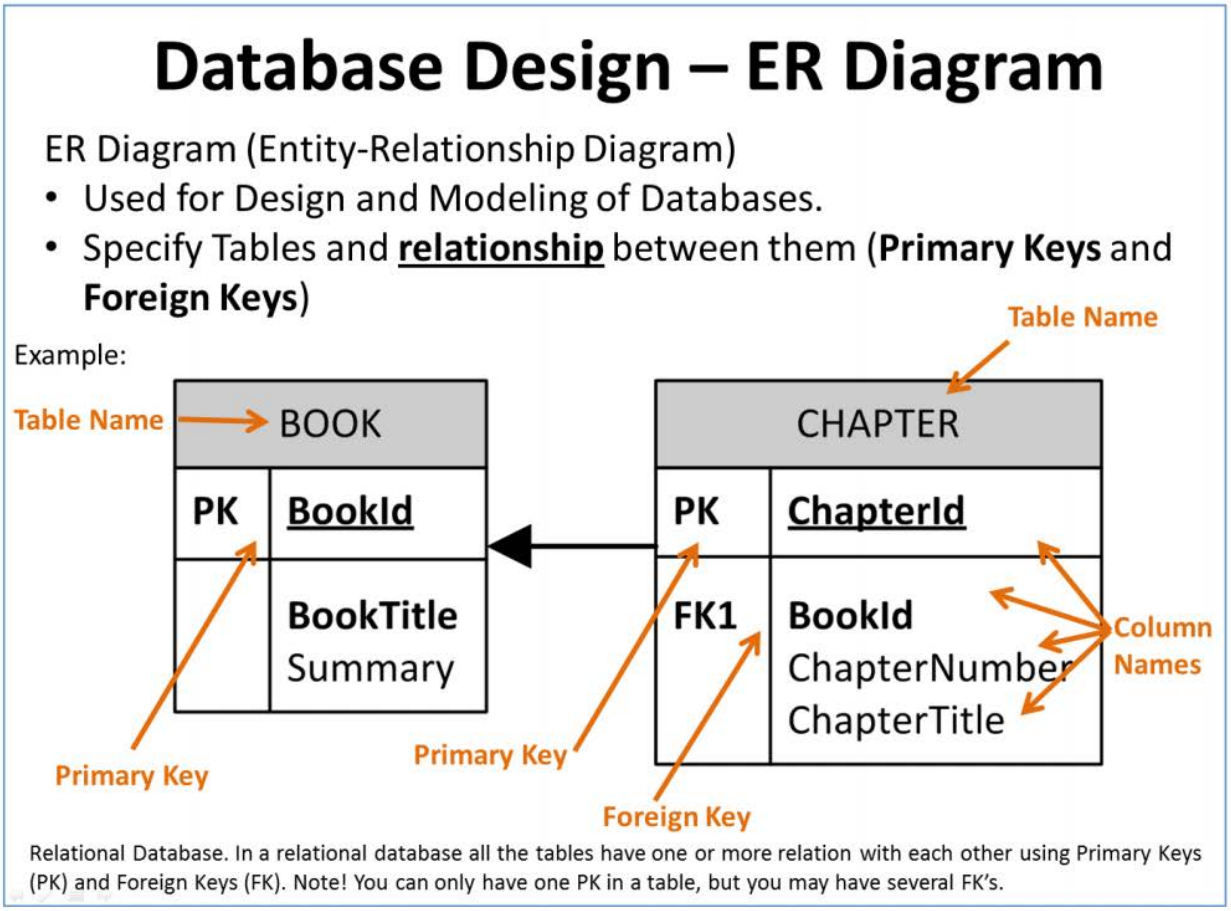


Here we can write any kind of queries that is supported by the SQL language.

Note: You may also use the SQL language to create a new database, but sometimes it is easier to just use the built-in features in the Management Studio.

3 CREATE TABLE

Before you start implementing your tables in the database, you should always spend some time design your tables properly using a design tool like, e.g., ERwin, Toad Data Modeler, PowerDesigner, Visio, etc. This is called Database Modeling.



The **CREATE TABLE** statement is used to create a table in a database.

Syntax:

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

The data type specifies what type of data the column can hold.

You have special data types for numbers, text dates, etc.

Examples:

- Numbers: **int**, **float**
- Text/Stings: **varchar(X)** – where X is the length of the string
- Dates: **datetime**
- etc.

Example: We want to create a table called "CUSTOMER" which has the following columns and data types:

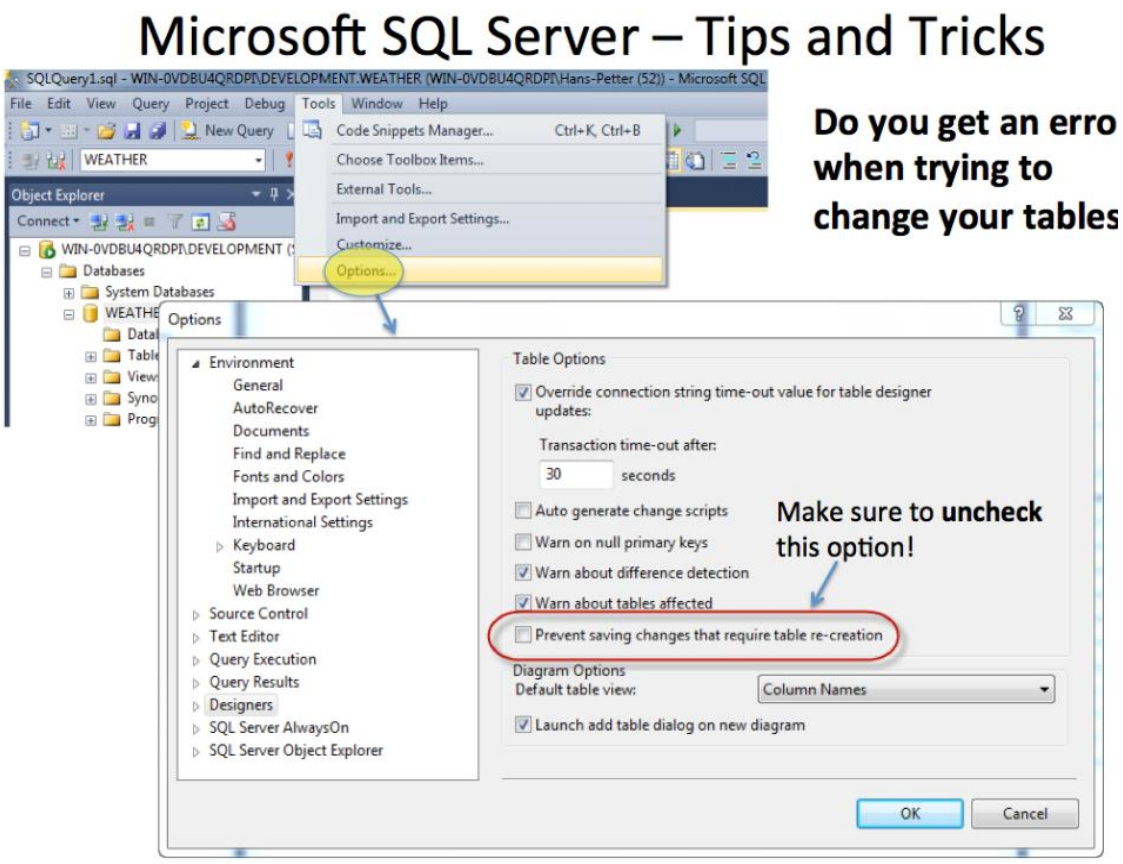
	Column Name	Data Type	Allow Nulls
🔑	CustomerId	int	☐
	LastName	varchar(50)	☐
	FirstName	varchar(50)	☐
	AreaCode	int	☑
	Address	varchar(200)	☑
	Phone	varchar(11)	☑
			☐

```
CREATE TABLE CUSTOMER
(
    CustomerId int IDENTITY(1,1) PRIMARY KEY,
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(200) NULL,
    Phone varchar(11) NULL,
)
```

Best practice:

When creating tables you should consider following these guidelines:

- Tables: Use upper case and singular form in table names – not plural, e.g., "STUDENT" (not students)
- Columns: Use Pascal notation, e.g., "StudentId"
- Primary Key:
  - If the table name is "COURSE", name the Primary Key column "CourseId", etc.
  - "Always" use Integer and Identity(1,1) for Primary Keys. Use UNIQUE constraint for other columns that needs to be unique, e.g. RoomNumber
- Specify Required Columns (NOT NULL) – i.e., which columns that need to have data or not
- Standardize on few/these Data Types: int, float, varchar(x), datetime, bit
- Use English for table and column names
- Avoid abbreviations! (Use RoomNumber – not RoomNo, RoomNr, ...)

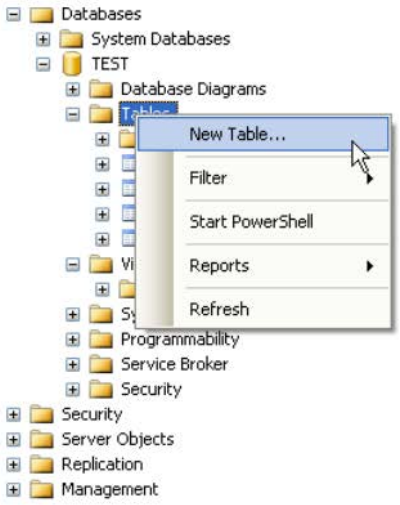


3.2 Create Tables using the Designer Tools

Even if you can do “everything” using the SQL language, it is sometimes easier to do it in the designer tools in the Management Studio in SQL Server.

Instead of creating a script you may as well easily use the designer for creating tables.

Step1: Select “New Table …”:



Step2: Next, the table designer pops up where you can add columns, data types, etc.

Column Name	Data Type	Allow Nulls
CustomerId	int	☐
LastName	varchar(50)	☐
FirstName	varchar(50)	☐
AreaCode	int	☑
Address	varchar(200)	☑
Phone	varchar(11)	☑
		☐

In this designer we may also specify Column Names, Data Types, etc.

Step 3: Save the table by clicking the Save button.

4 INSERT INTO

The INSERT INTO statement is used to insert a new row in a table.

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name

VALUES (value1, value2, value3,...)
```

Example:

```
INSERT INTO CUSTOMER

VALUES ('Rahman', 'Karim', 1203, 'Dhaka','01912584949')
```

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)

VALUES (value1, value2, value3,...)
```

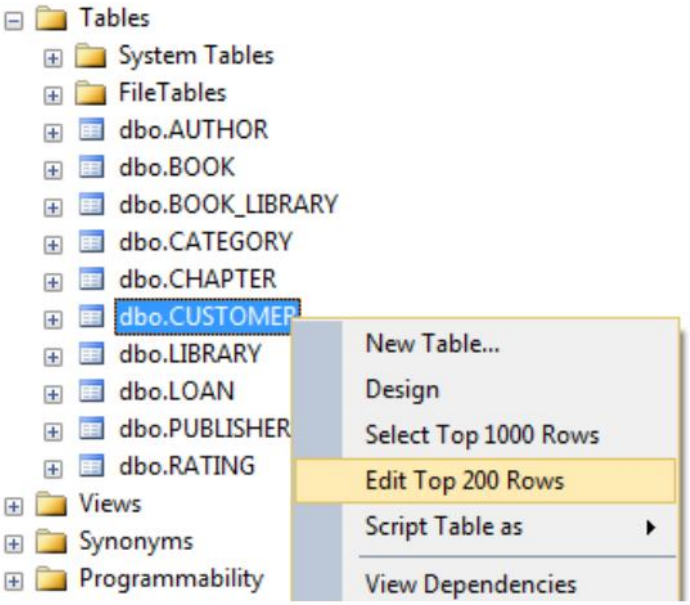
This form is recommended!

Insert Data Only in Specified Columns:

It is also possible to only add data in specific columns.

Insert Data in the Designer Tools:

When you have created the tables you can easily insert data into them using the designer tools. Right-click on the specific table and select “Edit Top 200 Rows”:



Then you can enter data in a table format, similar to, e.g., MS Excel:

5 ALTER TABLE

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

6 SQL Constraints

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

Here are the most important constraints:

- PRIMARY KEY
- NOT NULL
- UNIQUE
- FOREIGN KEY
- CHECK
- DEFAULT
- IDENTITY

In the sections below we will explain some of these in detail.

6.1 PRIMARY KEY

The PRIMARY KEY constraint uniquely identifies each record in a database table.

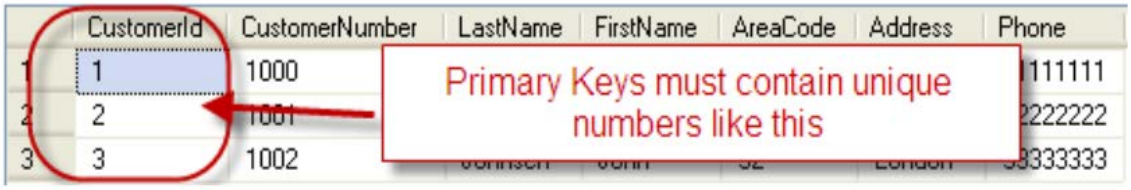
Primary keys must contain unique values. It is normal to just use running numbers, like 1, 2, 3, 4, 5, … as values in Primary Key column. It is a good idea to let the system handle this for you by specifying that the Primary Key should be set to **identity(1,1)**. IDENTITY(1,1) means the first value will be 1 and then it will increment by 1.

Each table should have a primary key, and each table can have only ONE primary key.

If we take a closer look at the CUSTOMER table created earlier:

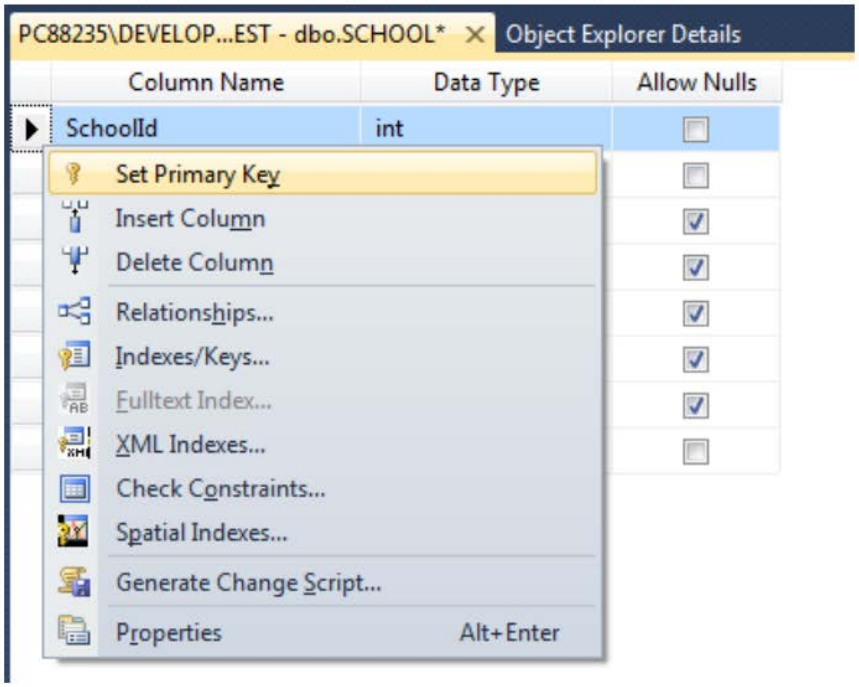
```
CREATE TABLE CUSTOMER
(
        CustomerId int IDENTITY(1,1) PRIMARY KEY,
        LastName varchar(50) NOT NULL,
        FirstName varchar(50) NOT NULL,
        AreaCode int NULL,
        Address varchar(200) NULL,
        Phone varchar(11) NULL,
)
```

As you see we use the “Primary Key” keyword to specify that a column should be the Primary Key.



Setting Primary Keys in the Designer Tools:

If you use the Designer tools in SQL Server you can easily set the primary Key in a table just by right-click and select “Set primary Key”.



The primary Key column will then have a small key in front to illustrate that this column is a Primary Key.

6.2 AUTO INCREMENT or IDENTITY

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.

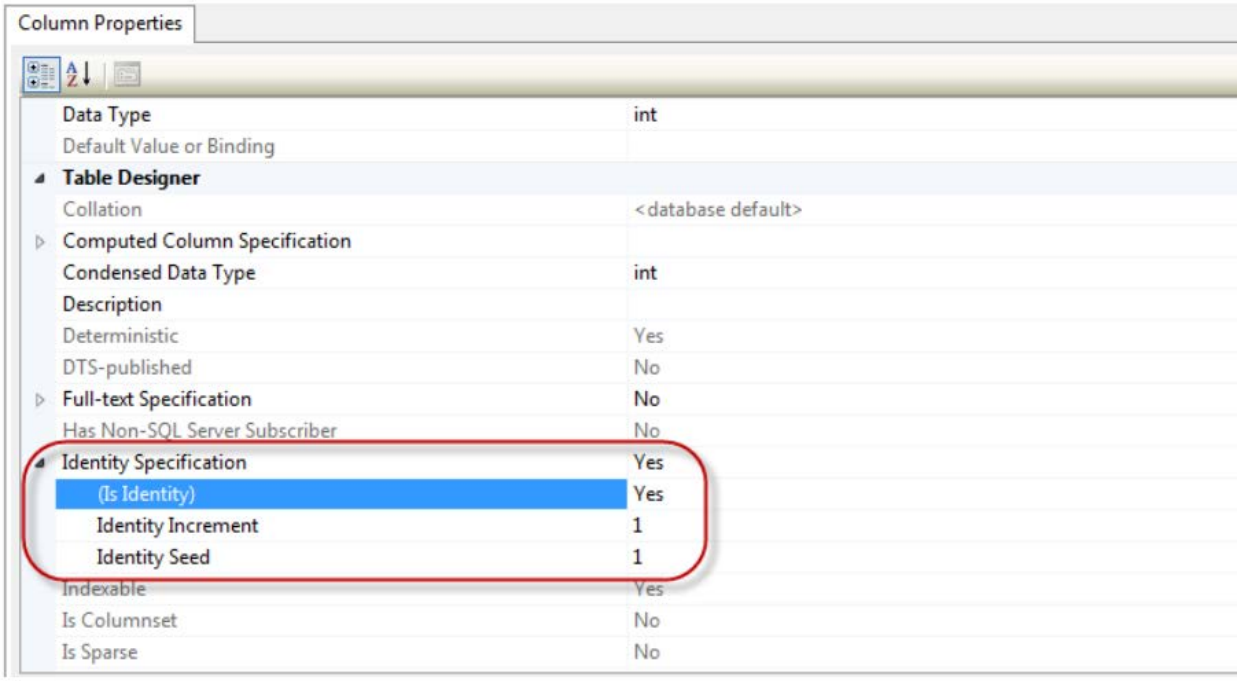
```
CREATE TABLE CUSTOMER
(
        CustomerId int IDENTITY(1,1) PRIMARY KEY,
        LastName varchar(50) NOT NULL,
        FirstName varchar(50) NOT NULL,
        AreaCode int NULL,
        Address varchar(200) NULL,
        Phone varchar(11) NULL,
)
```

As shown below, we use the IDENTITY () for this. IDENTITY (1, 1) means the first value will be 1 and then it will increment by 1.

Setting identity(1,1) in the Designer Tools:

We can use the designer tools to specify that a Primary Key should be an identity colu
that is automatically generated by the system when we insert data in to the table.

Click on the column in the designer and go into the Column Properties window:



Database Lab

CSE 3104

Lab-03

6 SQL Constraints

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

Here are the most important constraints:

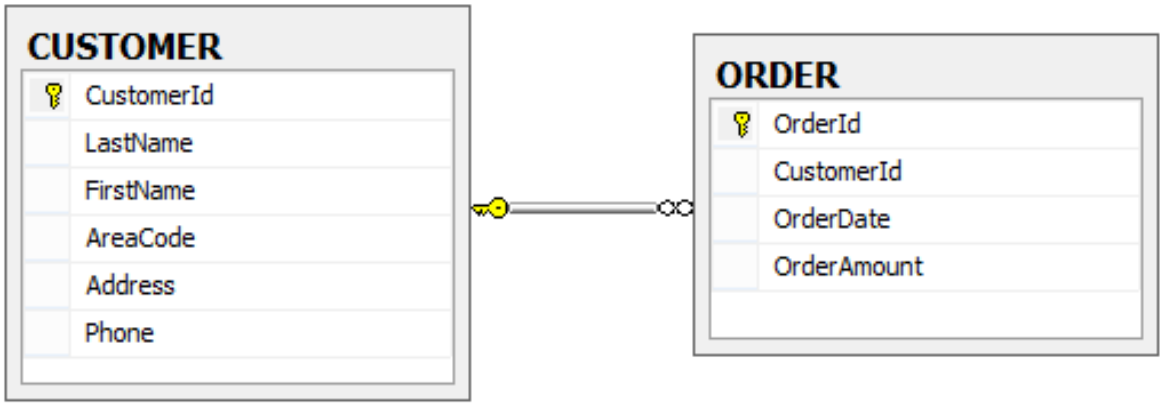
- PRIMARY KEY
- NOT NULL
- UNIQUE
- FOREIGN KEY
- CHECK
- DEFAULT
- IDENTITY

In the sections below we will explain some of these in detail.

6.3 FOREIGN KEY

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Example:



At First create a table called ORDER using the following Query.

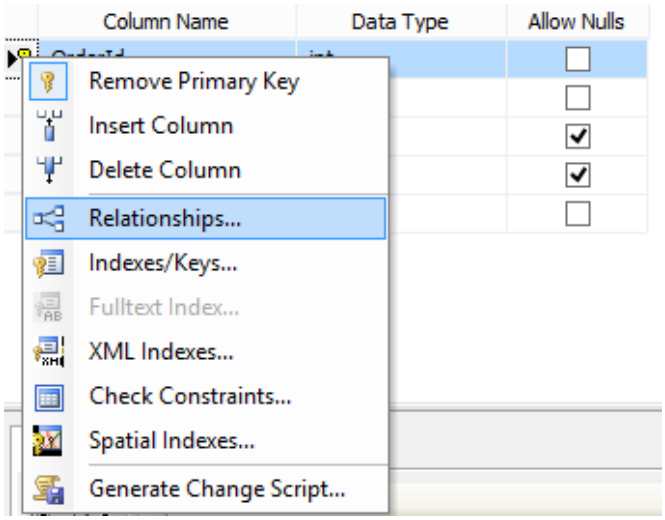
```
CREATE TABLE ORDER

(
OrderId int IDENTITY (1, 1) PRIMARY KEY,

CustomerId int NOT NULL FOREIGN KEY REFERENCES CUSTOMER (CustomerId),

OrderDate date NULL,

OrderAmount money NULL,

)
```

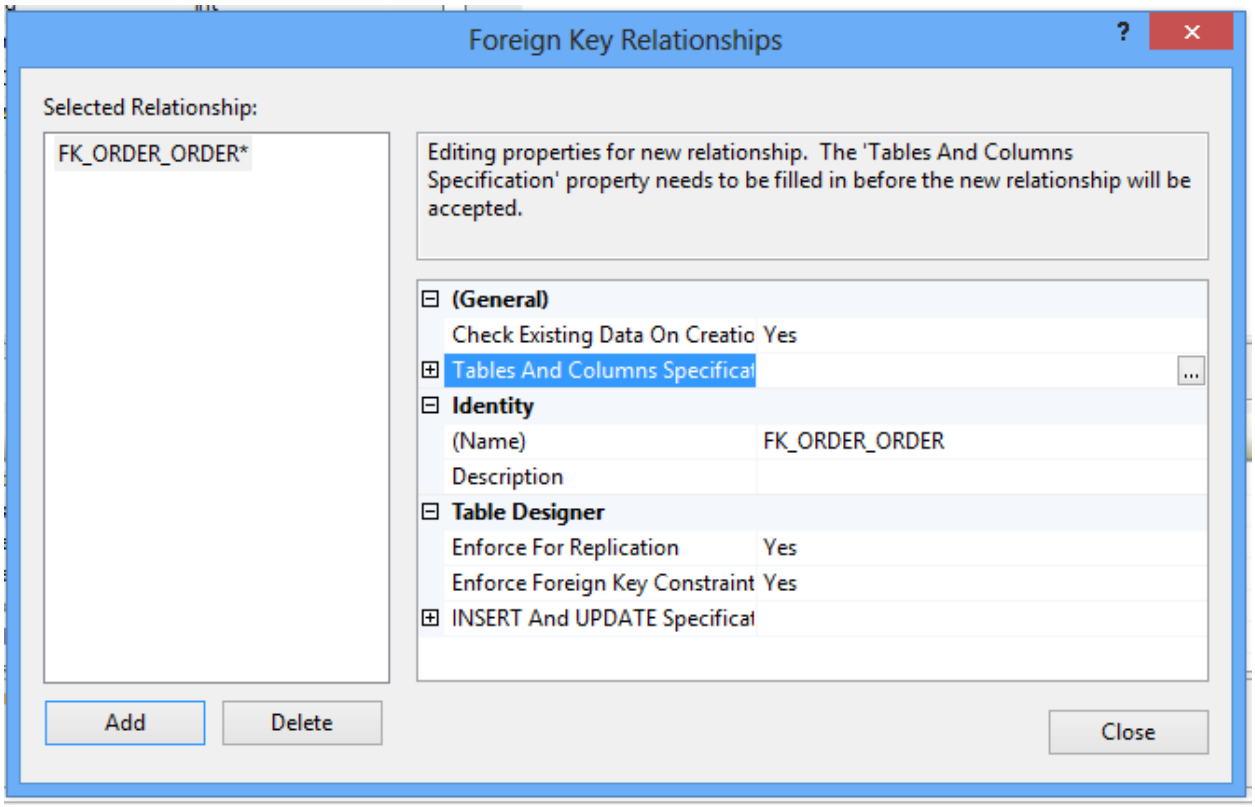
The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

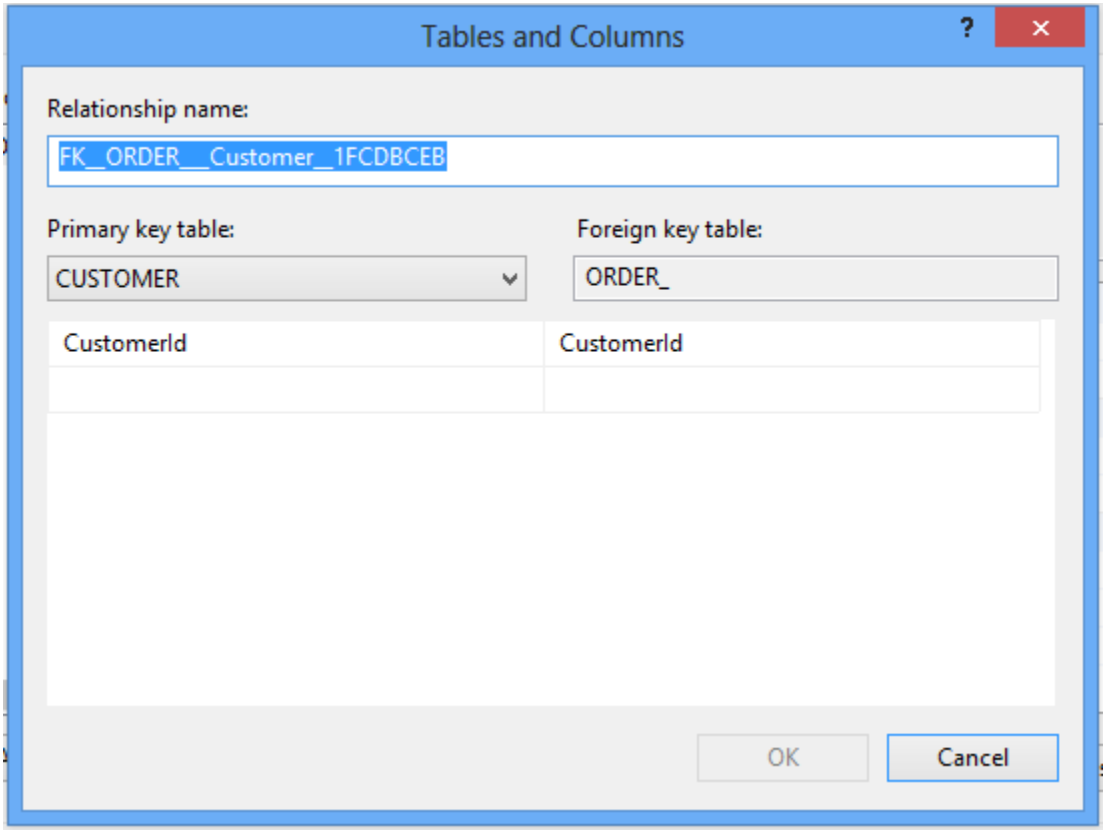
The FOREIGN KEY constraint also prevents that invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

Setting Foreign Keys in the Designer Tools:

If you want to use the designer, right-click on the column that you want to be the Foreign Key and select "**Relationships…**":




Foreign Key Relationships
Selected Relationship:
FK_ORDER_ORDER*
Editing properties for new relationship. The 'Tables And Columns Specification' property needs to be filled in before the new relationship will be accepted.
(General)
Check Existing Data On Creatio Yes
Tables And Columns Specificat
Identity
(Name) FK_ORDER_ORDER
Description
Table Designer
Enforce For Replication Yes
Enforce Foreign Key Constraint Yes
INSERT And UPDATE Specificat
Add
Delete
Close


Tables and Columns
Relationship name:
FK__ORDER__Customer__1FCDBCEB
Primary key table:
CUSTOMER
Foreign key table:
ORDER_
CustomerId
CustomerId
OK
Cancel

6.4 UNIQUE

The **UNIQUE** constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

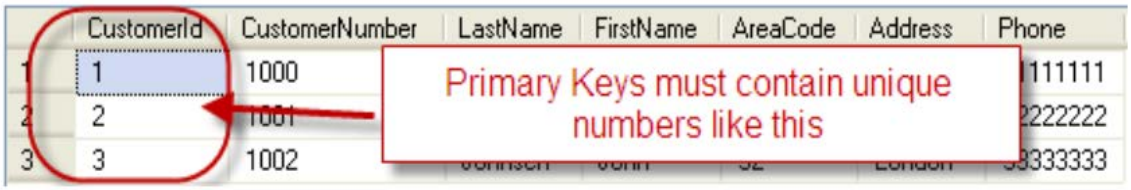
A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note! You can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

If we take a closer look at the CUSTOMER table created earlier:

```
CREATE TABLE CUSTOMER
(
        CustomerId int IDENTITY(1,1) PRIMARY KEY,
        CustomerNumber int NOT NULL UNIQUE ,
        LastName varchar(50) NOT NULL,
        FirstName varchar(50) NOT NULL,
        ------------------------------
)
```

As you see we use the “Primary Key” keyword to specify that a column should be the Primary Key.



6.5 CHECK

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

```
CREATE TABLE CUSTOMER
(
        CustomerId int IDENTITY(1,1) PRIMARY KEY,
        CustomerNumber int NOT NULL UNIQUE  CHECK(CustomerNumber>1000),
        LastName varchar(50) NOT NULL,
        FirstName varchar(50) NOT NULL,
        ------------------------------
)
```

In this case, when we try to insert a Customer Number less than zero we will get an error message.

6.5 DEFAULT

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

Example:

```
CREATE TABLE CUSTOMER
(
        CustomerId int IDENTITY(1,1) PRIMARY KEY,
        CustomerNumber int NOT NULL UNIQUE  CHECK(CustomerNumber>1000),
        LastName varchar(50) NOT NULL,
        FirstName varchar(50) NOT NULL,
        AreaCode int NULL,
        Address varchar(200) NULL  DEFAULT 'Dhaka',
        Phone varchar(11) NULL,
)
```

7 UPDATE

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

The syntax is as follows:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note! Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Example:

```
UPDATE CUSTOMER set AreaCode=46 where CustomerId=2
```

Note: If you don't include the WHERE clause then result becomes updated to all records. So make sure to include the WHERE clause when usingthe UPDATE command!

8 DELETE

The DELETE statement is used to delete rows in a table.

Syntax:

```
DELETE FROM table_name
WHERE some_column=some_value
```

Note! Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Example:

```
delete from CUSTOMER where CustomerId=2
```

Delete All Rows:

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
```

Note! Make sure to do this only when you really mean it! You cannot UNDO this statement!

9 SELECT

The SELECT statement is probably the most used SQL command. The SELECT statement is used for retrieving rows from the database and enables the selection of one or many rows or columns from one or many tables in the database.

We will use the CUSTOMER table as an example.

Example:

```
select * from CUSTOMER
```

This simple example gets all the data in the table CUSTOMER. The symbol "*" is used when you want to get all the columns in the table.

If you only want a few columns, you may specify the names of the columns you want to retrieve, example:

```
select CustomerId, LastName, FirstName from CUSTOMER
```

So in the simplest form we can use the SELECT statement as follows:

```
select <column_names> from <table_names>
```

If we want all columns, we use the symbol "*"

Note! SQL is not case sensitive. SELECT is the same as select.

The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

```
SELECT
[ ALL | DISTINCT ]
    [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
select_list [ INTO new_table ]
[ FROM table_source ] [ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

It seems complex, but we will take the different parts step by step in the next sections.

10 The ORDER BY Keyword

If you want the data to appear in a specific order you need to use the "order by" keyword.

Example:

```
select * from CUSTOMER order by LastName
```

You may also sort by several columns, e.g. like this:

```
select * from CUSTOMER order by Address, LastName
```

If you use the “order by” keyword, the default order is ascending (“asc”). If you want the order to be opposite, i.e., descending, then you need to use the “desc” keyword.

```
select * from CUSTOMER order by LastName desc
```

11 SELECT DISTINCT

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

The syntax is as follows:

```
select distinct <column_names> from <table_names>
```

Example:

```
select distinct FirstName from CUSTOMER
```

12 The WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

The syntax is as follows:

```
select <column_names>
from <table_name>
where <column_name> operator value
```

Example:

```
select * from CUSTOMER where CustomerNumber='1001'
```

Note! SQL uses single quotes around text values, as shown in the example above.

13 Operators

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Examples:

```
select * from CUSTOMER where AreaCode>30
```

Database Lab

CSE 3104

Lab-04

13 Operators

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Examples:

```
select * from CUSTOMER where AreaCode>30
```

13.1 LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:

→The percent sign (%)

→The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and _ is as follows:

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

Following is an example, which would display all the records from CUSTOMER table where SALARY starts with 200.

```
SELECT  *  FROM CUSTOMER WHERE SALARY LIKE '200%'
```

You may also combine with the **NOT** keyword.(Logical Negation) The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.

```
SELECT  *  FROM CUSTOMER WHERE SALARY NOT LIKE '200%'
```

13.2 IN Operator

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

Syntax:

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

```
SELECT * FROM CUSTOMER WHERE AGE IN ( 25, 27 )
```

13.3 BETWEEN Operator

The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

```
SELECT * FROM CUSTOMER WHERE AGE BETWEEN 23 AND 27
```

13.4 AND & OR Operators

The AND operator displays a record if both the first condition and the second condition is true.

The OR operator displays a record if either the first condition or the second condition is true.

Examples:

```
SELECT * FROM CUSTOMER WHERE AGE >= 25 AND SALARY >= 6500
```

```
SELECT * FROM CUSTOMER WHERE AGE >= 25 OR SALARY >= 6500
```

You can also combine AND and OR (use parenthesisto form complex expressions).

```
SELECT * FROM CUSTOMER WHERE NAME LIKE 'Ka%' AND (AGE >= 25 OR SALARY >= 6500)
```

14 SELECT TOP Clause

The TOP clause is used to specify the number of records to return.

The TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
```

```
SELECT TOP 3 * from CUSTOMER
```

```
SELECT TOP 60 percent * from CUSTOMER
```

15 Built-in Functions

SQL has many built-in functions for performing calculations on data.

We have 2 categories of functions, namely **aggregate** functions and **scalar** functions. Aggregate functions return a single value, calculated from values in a column, while scalar functions return a single value, based on the input value.

Aggregate functions - examples:

- **AVG()** - Returns the average value
- **STDEV()** - Returns the standard deviation value
- **COUNT()** - Returns the number of rows
- **MAX()** - Returns the largest value
- **MIN()** - Returns the smallest value
- **SUM()** - Returns the sum
- etc.

Scalar functions - examples:

- **UPPER()** - Converts a field to upper case
- **LOWER()** - Converts a field to lower case
- **LEN()** - Returns the length of a text field
- **ROUND()** - Rounds a numeric field to the number of decimals specified
- **GETDATE()** - Returns the current system date and time
- etc.

16 The Group By Statement

Aggregate functions often need an added GROUP BY statement.

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.

If we try the following:

```
SELECT Age, MAX(Salary) FROM CUSTOMER
```

We will get the following error message

```
Msg 8120, Level 16, State 1, Line 1
Column 'CUSTOMER.Age' is invalid in the select list because it is not
contained in either an aggregate function or the GROUP BY clause.
```

The solution is to use GROUP BY:

```
SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age
```

17 The Having Clause

The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

```
SELECT
[ ALL | DISTINCT ]
    [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
select_list [ INTO new_table ]
[ FROM table_source ] [ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

If we try the following:

```
SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age HAVING NAME LIKE 'Ka%'
```

We will get the following error message

```
Msg 8121, Level 16, State 1, Line 1
Column 'CUSTOMER.Name' is invalid in the HAVING clause because it is not
contained in either an aggregate function or the GROUP BY clause.
```

```
SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age HAVING Age >=25
```

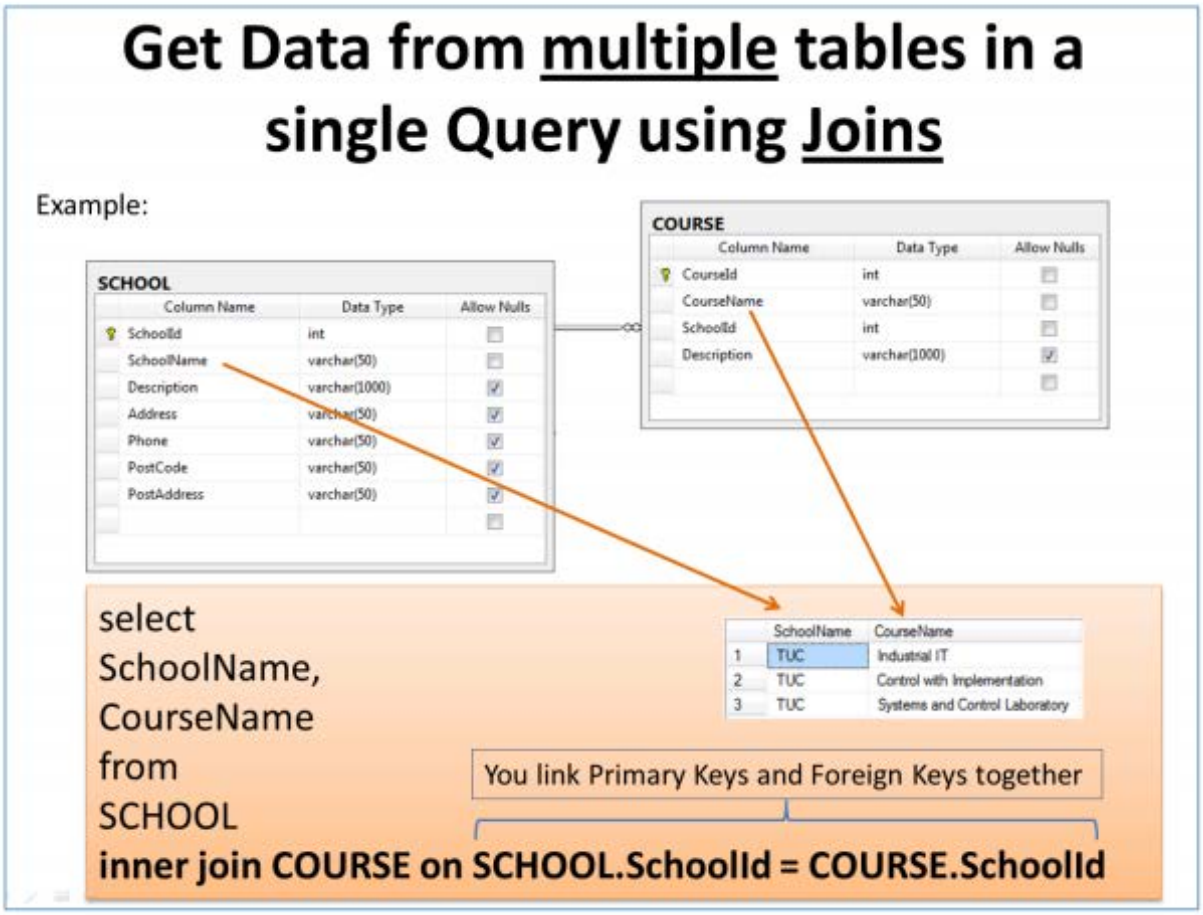
Database Lab

CSE 3104

Lab-05

5 Joins

SQL joins are used to query data from two or more tables, based on a relationship between certain columns in these tables.



We want to get the following information using a query:

SchoolName	ClassName
...	...
...	...

In order to get information from more than one table we need to use the JOIN. The JOIN is used to join the primary key in one table with the foreign key in another table.

```
select
SCHOOL.SchoolName,
CLASS.ClassName
from
SCHOOL
INNER JOIN CLASS ON SCHOOL.SchoolId = CLASS.SchoolId
```

5.1 Different SQL Joins

Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

- JOIN: Return rows when there is at least one match in both tables
- LEFT JOIN: Return all rows from the left table, even if there are no matches in the right table
- RIGHT JOIN: Return all rows from the right table, even if there are no matches in the left table
- FULL JOIN: Return rows when there is a match in one of the tables

5.2 Join Operations

Consider the CUSTOMER and ORDERS table

Now, let us join these two tables in our SELECT statement as follows:

```
SELECT CUSTOMER.CustomerId , Name, Age, Amount
FROM CUSTOMER, ORDERS
WHERE CUSTOMER.CustomerId = ORDERS.CustomerId
```

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=,! =, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

5.2.1 Inner Join

The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN.

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

The basic syntax of INNER JOIN is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_filed = table2.common_field
```

```
SELECT CUSTOMER.CustomerId , Name, Age,
Amount,Date
FROM CUSTOMER
INNER JOIN ORDERS
ON CUSTOMER.CustomerId = ORDERS.CustomerId
```

5.2.2 Left Join

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

The basic syntax of LEFT JOIN is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
LEFT JOIN table2
ON table1.common_filed = table2.common_field
```

```
SELECT CUSTOMER.CustomerId , Name, Age,
Amount,Date
FROM CUSTOMER
LEFT JOIN ORDERS
ON CUSTOMER.CustomerId = ORDERS.CustomerId
```

5.2.3 Right Join

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

The basic syntax of RIGHT JOIN is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
RIGHT JOIN table2
ON table1.common_filed = table2.common_field
```

```
SELECT CUSTOMER.CustomerId , Name, Age,
Amount,Date
FROM CUSTOMER
RIGHT JOIN ORDERS
ON CUSTOMER.CustomerId = ORDERS.CustomerId
```

5.2.3 Full Join

The SQL FULL JOIN combines the results of both left and right outer joins.

The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

```
SELECT table1.column1, table2.column2...
FROM table1
FULL JOIN table2
ON table1.common_filed = table2.common_field
```

6 SQL Sub Queries

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

→ Subqueries must be enclosed within parentheses.

→ A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.

→An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.

→Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.

→A subquery cannot be immediately enclosed in a set function.

→The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement:

At first try with the following:

```
SELECT *
FROM CUSTOMER
WHERE CustomerId = (SELECT CustomerId
FROM CUSTOMER
WHERE Salary > 4500)
```

This will show an error message like:

```
Msg 512, Level 16, State 1, Line 1
Subquery returned more than 1 value. This is not permitted when the subquery
follows =, !=, <, <= , >, >= or when the subquery is used as an expression.
```

Correct form is:

```
SELECT *
FROM CUSTOMER
WHERE CustomerId IN (SELECT CustomerId
FROM CUSTOMER
WHERE Salary> 4500)
```

Note: Sub queries also can be performed on multiple tables.

Subqueries with the INSERT Statement:

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

Consider a table CUSTOMER_BKP with similar structure as CUSTOMER table. Now to copy complete CUSTOMER table into CUSTOMERS_BKP, following is the query:

```
INSERT INTO CUSTOMER_BKP
SELECT Name,Age,Address,Salary FROM CUSTOMER
WHERE CustomerId IN (SELECT CustomerId
FROM CUSTOMER)
```

Subqueries with the UPDATE Statement:

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

Following example updates SALARY by 0.25 times in CUSTOMER table for all the customers whose AGE is greater than or equal to 27:

```
UPDATE CUSTOMER
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMER_BKP
WHERE AGE >= 27 )
```

Subqueries with the DELETE Statement:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

```
UPDATE CUSTOMER
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMER_BKP
WHERE AGE >= 27 )
```

Practice Session:

1. **List Product ID , Product Name , Product Type Name of all products .**
2. **Show all product information along with Product Type Name that has a price greater than $29.95(use alias)**
3. **List all the last name of the customers, Order IDs and Product Name in customer name order.(all of the products a given customer has ordered)**
4. **Show those orders which have product ID = 3.**

Database Lab

CSE 3104

Lab-07

7.1 SQL DROP COMMAND

DROP command completely removes a table from a database. This command will also destroy the table structure. Also a database can be drop with this command.

Basic Structure-

DROP TABLE "table_name"

DROP DATABASE DatabaseName;

Example-

DROP TABLE COURSE

DROP DATABASE EMPLOYEE

NOTE: You can also DROP multiple by using comma (,)

7.2 SQL TRUNCATE COMMAND

To remove all the rows from a table, the TRUNCATE command is used. TRUNCATE deletes all the rows, but the table structure remains the same. TRUNCATE is a DDL command.

When we apply Truncate command on a table then its PRIMARY KEY is initialized.

Basic Structure-

TRUNCATE TABLE "table_name"

Example-

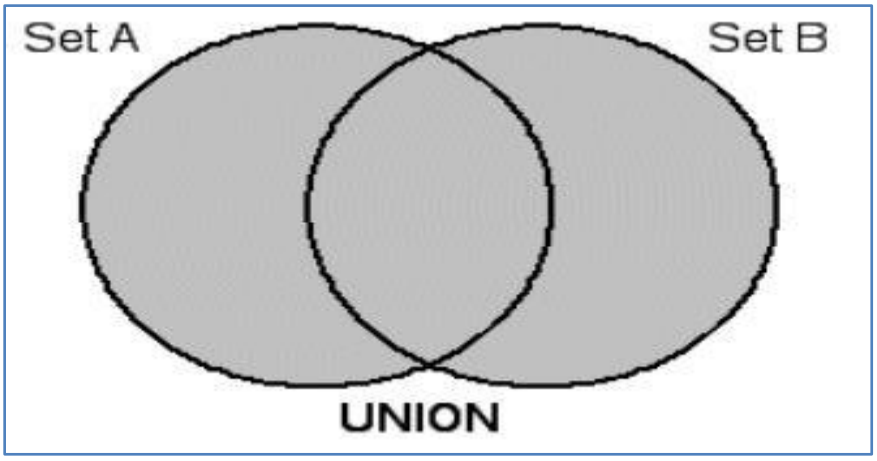
```
TRUNCATE TABLE STUDENT
```

NOTE: TRUNCATE & DELETE commands are not same. Delete command will delete all the rows from a table where as Truncate command re-initializes a table.

7.3 SQL UNION

The SQL **UNION operator** is used to combine the result sets of 2 or more SELECT statements. It removes duplicate rows between the various SELECT statements.

Each SELECT statement within the UNION must have the same number of fields in the result sets with similar data types, also same order, but they do not have to be the same length.



Basic Structure-

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2
```

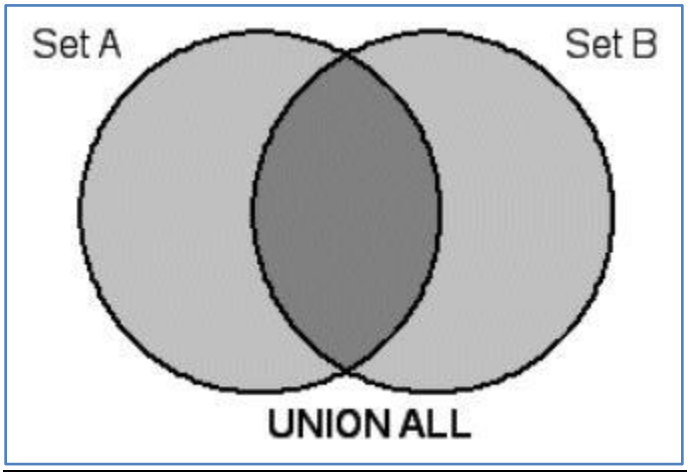
Example-

```
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
UNION
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
```

7.4 SQL UNION ALL

Same as UNION. The purpose of the SQL **UNION ALL** command is to combine the results of two queries together.

UNION and **UNION ALL** both combine the results of two SQL queries. The difference is that, while **UNION** only selects distinct values, **UNION ALL** selects all values.



Basic Structure-

SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2

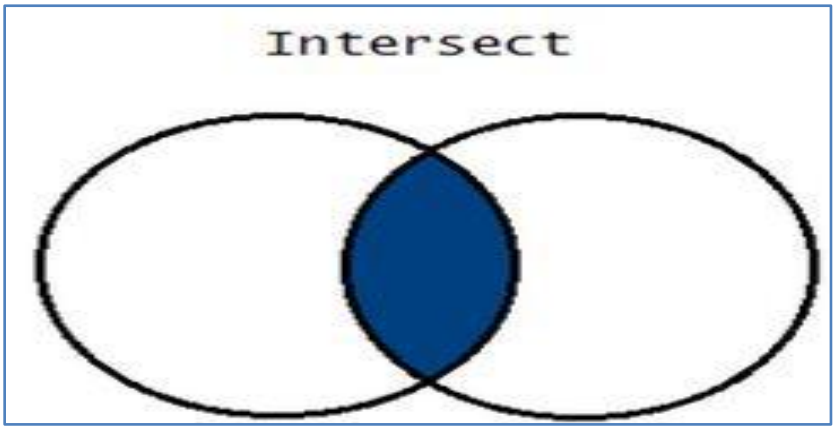
Example-

```
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
UNION ALL
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
```

7.5 SQL INTERSECT

The SQL **INTERSECT** clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT operator.



Basic Structure-

SELECT column_name(s)

FROM table

INTERSECT

SELECT column_name(s)

FROM table

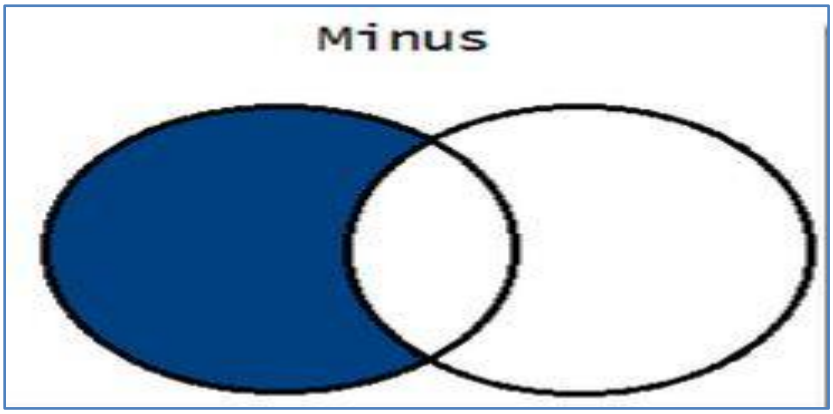
Example-

```
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
INTERSECT
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='KHULNA')
```

7.5 SQL EXCEPT

The **EXCEPT** command operates on two SQL statements. It takes all the results from the first SQL statement, and then subtract out the ones that are present in the second SQL **statement to get the final answer. If the second SQL statement includes results not present in the first SQL statement, such results are ignored**.

Each SELECT statement within the EXCEPT query must have the same number of fields in the result sets with similar data types.



Basic Structure-

SELECT column_name(s)

FROM table

MINUS

SELECT column_name(s)

FROM table

Example-

```
(SELECT EmployeeName,City
FROM EMPLOYEE
)
EXCEPT
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA'
)
```

7.6 ANY, ALL with a Multiple Row Sub query:

Both are used for comparing one value against a set of values. ALL specifies that all the values given in the list should be taken into account, whereas ANY specifies that the condition is satisfied when any of the values satisfies the condition.

The operator can be any one of the standard relational operators (=, >=, >, <, <=, !=) , and

list is a series of values.

operator ANY list

operator ALL list

The ANY keyword denotes that the search condition is TRUE if the comparison is TRUE for at least one of the values that is returned. If the subquery returns no value, the search condition is FALSE. **The SOME keyword is a synonym for ANY**.

Example-

```
SELECT * FROM EMPLOYEE
WHERE DepartmentId > ANY
(SELECT DepartmentId FROM DEPARTMENT WHERE Budget > 30000.00)
```

The ALL keyword specifies that the search condition is TRUE if the comparison is TRUE for every value that the subquery returns. If the subquery returns no value, the condition is FALSE.

Example-

```
SELECT * FROM EMPLOYEE
WHERE DepartmentId > ALL
(SELECT DepartmentId FROM DEPARTMENT WHERE Budget > 30000.00)
```

7.7 Using EXISTS, NOT EXISTS with a Subquery:

The EXISTS operator checks if a subquery returns any rows. The following illustrates the syntax of the EXISTS operator:

WHERE EXISTS (subquery)

The expression EXISTS (subquery) returns TRUE if the subquery returns at least one row, otherwise it returns FALSE. Notice that you put the subquery inside the parentheses followed by the EXISTS operator.

NOT operator with the EXISTS operator to inverse the meaning of the EXISTS operator.

WHERE NOT EXISTS (subquery)

The expression NOT EXISTS (subquery) returns TRUE if the subquery returns no row, otherwise it returns FALSE.

NOTE:

EXISTS is different from other operators like IN,ANY etc., because it doesn't compare values of columns, instead, it checks whether any row is retrieved from subquery or not.

Example-

EXISTS

```
SELECT EmployeeId,FirstName,DepartmentId
FROM EMPLOYEE
WHERE EXISTS
(SELECT d.DepartmentId FROM DEPARTMENT as d
JOIN EMPLOYEE as e on (e.DepartmentId=d.DepartmentId)
WHERE
d.DepartmentId =6)
```

Here in DepartmentId 6 there is no employee, so sub query returns Nothing. For this, overall outer query shows nothing.

Example-

NOT EXISTS

'TRY FOR ABOVE QUERY'

SQL STRING FUNCTIONS

Sql string function is a built-in string function.
It perform an operation on a string input value and return a string or numeric value.

Some common sql string functions:

1. LEFT - Returns left part of a string with the specified number of characters.
   Syntax-
   LEFT (string , integer)
   Example-

```
SELECT LEFT('Samia Tasnim',5) AS LeftName

-----OR----

SELECT LEFT(FirstName,2) FROM EMPLOYEE
```

2. RIGHT - Returns Right part of a string with the specified number of characters.
   Syntax-
   RIGHT (string , integer)
   Example-

```
SELECT RIGHT('DHAKA 1215',4) AS RightPart

------OR----

SELECT RIGHT(FirstName,2) FROM EMPLOYEE
```

3. SUBSTRING - Returns part of a string.
   Syntax-
   SUBSTRING (string, startindex , length)
   Example-

```
SELECT SUBSTRING('12-01-04-074',10,3) AS ROLL
----OR----
SELECT SUBSTRING(AddressOfEmployee,1,3) FROM EMPLOYEE
```

4. REVERSE - Returns reverse a string.

Syntax-

REVERSE(string)

Example-

```
SELECT REVERSE('MSSQL') AS REV
-----OR----
SELECT REVERSE(FirstName) AS RevFirstName FROM EMPLOYEE
```

---

5. CAST - Returns the value of an expression converted to a supplied data type.

Syntax-

CAST (expression AS [data type])

Example-

```
SELECT CAST(57.58 AS INTEGER) AS IntValue
-------OR--------
SELECT CAST(Budget AS INTEGER) AS Budget FROM DEPARTMENT
```

---

6. CONVERT - Converts a value to another data type. Similar to CAST.

Syntax-

CONVERT (expression, [data type])

Example-

```
SELECT CONVERT(INTEGER,78.8)AS Int_Value
------OR---------
SELECT CONVERT(Integer,Budget) AS Int_Budget FROM DEPARTMENT
```

---

7. CONCAT - This Keyword not use in SQL, But we can CONCAT two part as –

```
SELECT FirstName+' '+LastName FROM EMPLOYEE
```

SQL DATE FUNCTIONS

SQL Server date and time functions are scalar functions that perform an operation on a date and time input value and returns either a string, numeric, or date and time value.
Some useful Date Functions-

1. DATEADD - Returns a new datetime value based on adding an interval to the specified date.

Syntax-

DATEADD (datepart , number, date)

Example-

```
SELECT DATEADD(YEAR,3,'1992-04-07') As IncrementYear
------OR-----
SELECT DATEADD(MONTH,2,DOB) As IncrementMonth
FROM EMPLOYEE
WHERE EmployeeID=2
```

2.DATEDIFF - Returns the number of date and time boundaries crossed between two specified dates.

Syntax-

DATEDIFF (datepart , startdate , enddate)

Example-

```
SELECT DATEDIFF(YEAR,'1992-03-11','1995-08-23') AS DateDiff
```

---

3. DATEPART - Returns an integer that represents the specified datepart of the specified date.

Syntax-

DATEPART (datepart , date)

Example-

```
SELECT DATEPART(DAY,'2001-07-24') AS Day
----OR-----
SELECT DATEPART(DAY,DOB) As Day
FROM EMPLOYEE
WHERE EmployeeID=2
```

4. DAY - Returns an integer representing the day datepart of the specified date.

Syntax-

DAY (date)

Example-

```
SELECT DAY('1990-06-14') AS Day
--------OR---------
SELECT DAY(DOB) AS Day
FROM EMPLOYEE
```

Like DAY Function MONTH, YEAR also hold same structure.

5. The GETDATE function is used to retrieve the current database system time in SQL Server.

Syntex-

SELECT GETDATE()

Self Study:

Practice all the commands related to today's class. A practical session will be conducted at the very beginning of the next lab.