

Database Lab

CSE 3104

Session 02

❖ Introduction to SQL

SQL (*Structured Query Language*) is a 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:

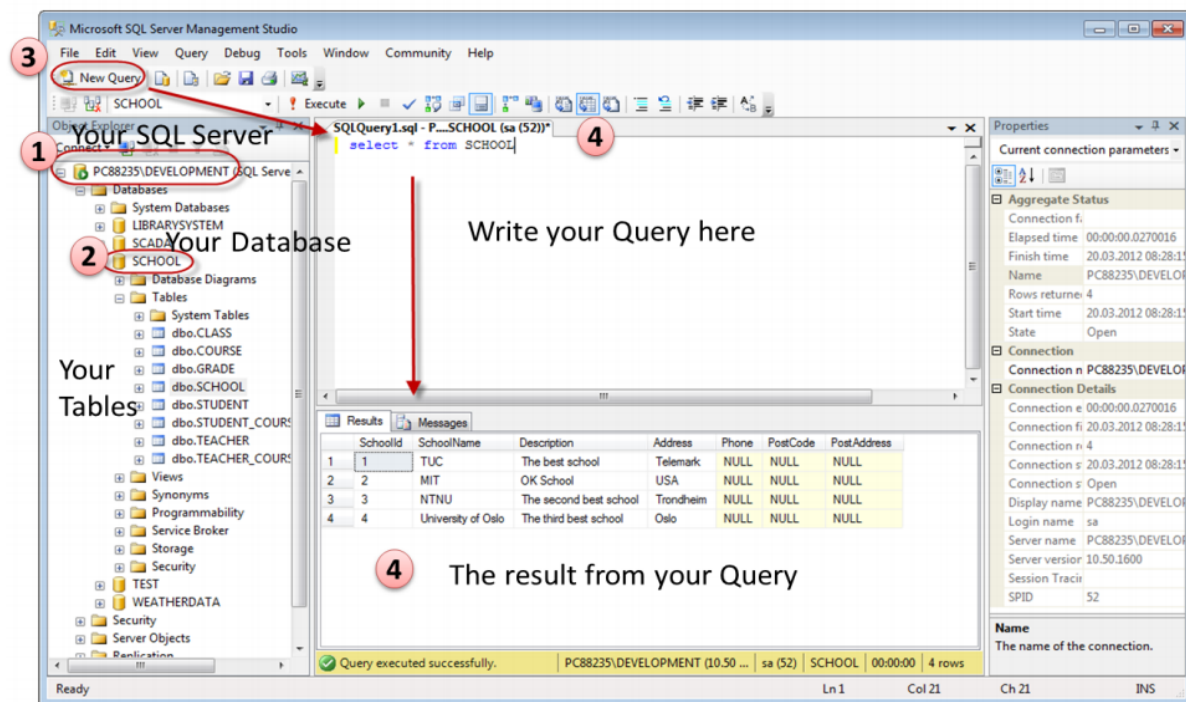
- Microsoft SQL Server
 - Enterprise, Developer versions, etc
 - **Express version is free of charge**
- Oracle
- MySQL (Oracle, previously Sun Microsystems)- MySQL can be used free of charge (open source license), Web sites that use MySQL: YouTube, Wikipedia, Facebook
- Microsoft Access
- IBM DB2
- Sybase
- ... lots of other systems



❖ SQL Server Management 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.

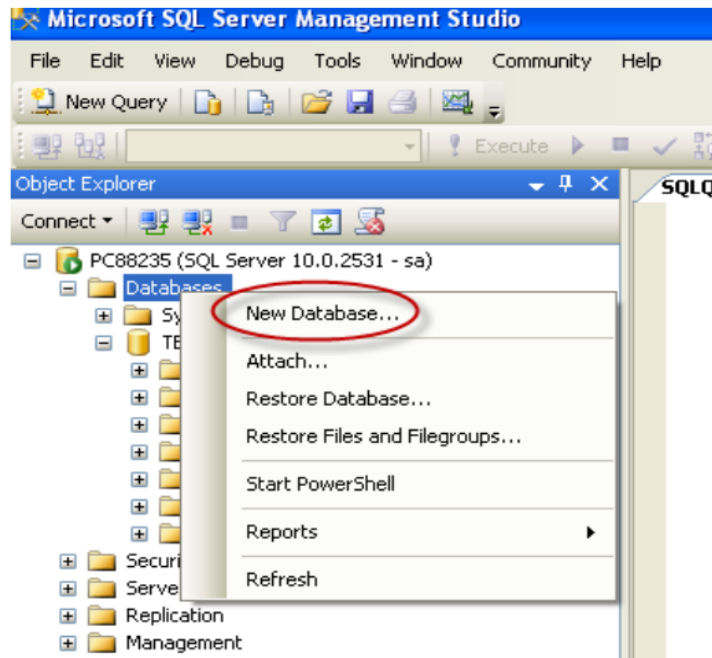


When creating SQL commands and queries, the “Query Editor” (select “New Query” from the Toolbar) is used (shown in the figure above).

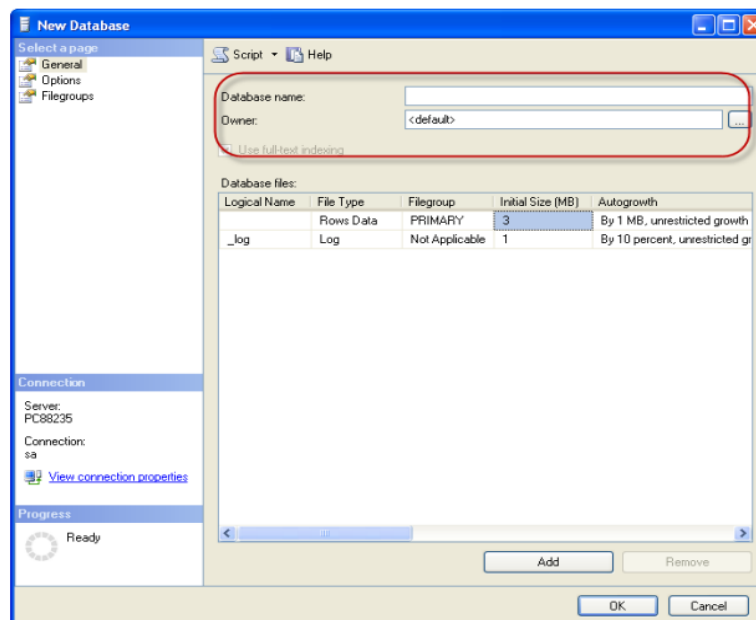
With SQL and the “Query Editor” we can do almost everything with code, but sometimes it is also a good idea to use the different Designer tools in SQL to help us do the work without coding (so much).

❖ Creating 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...”

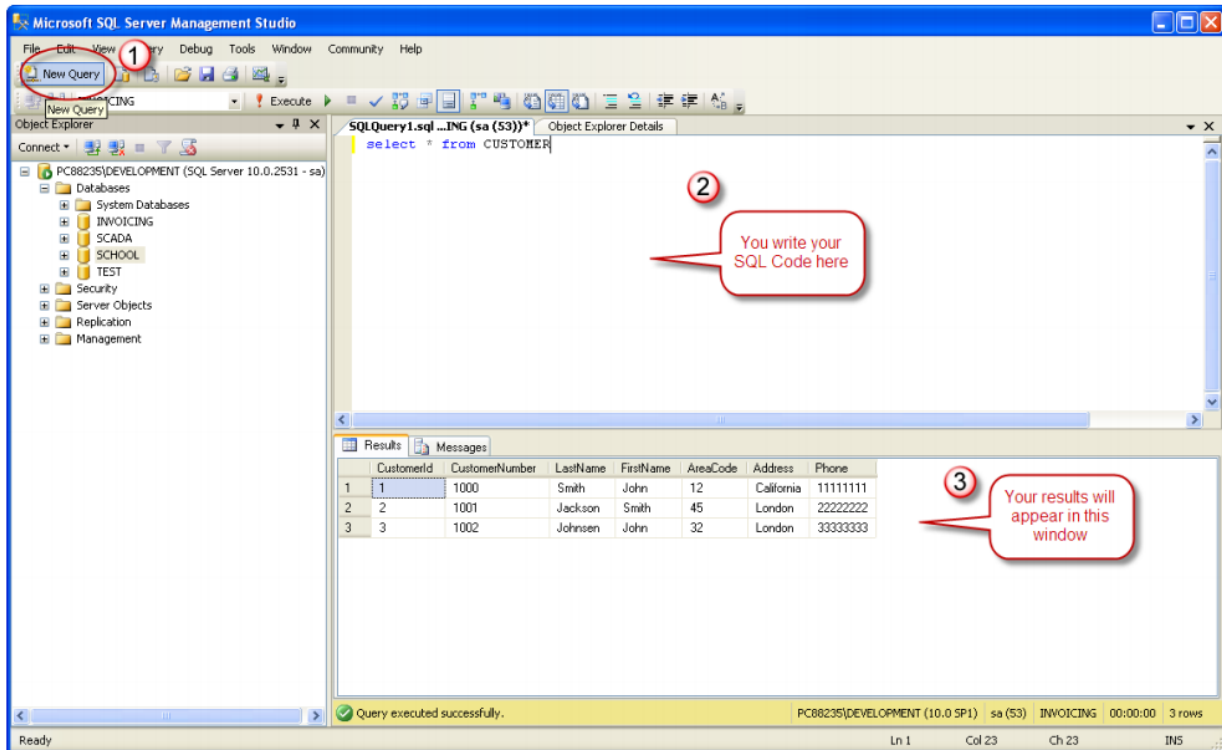


There are lots of settings you may set regarding your database, but the only information you must fill in is the name of your database:



❖ 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.

Database is a collection of objects such as table, view, stored procedure, function, trigger, etc.

In MS SQL Server, two types of databases are available.

- System databases
- User Databases

System Databases

System databases are created automatically when we install MS SQL Server. Following is a list of system databases –

- Master
- Model

- MSDB
- Tempdb

User Databases

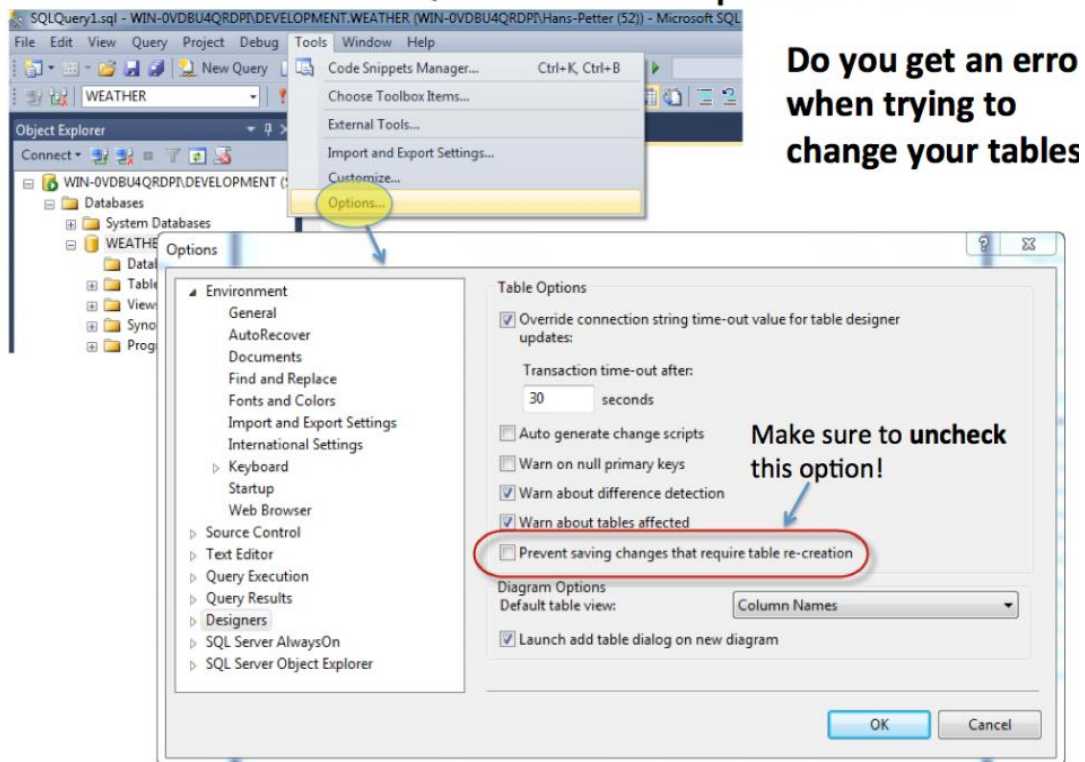
User databases are created by users (*Administrators, developers, and testers who have access to create databases*).

You can 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.

CREATE DATABASE TestDB

DROP DATABASE TestDB

Microsoft SQL Server – Tips and Tricks



1. CREATE TABLE

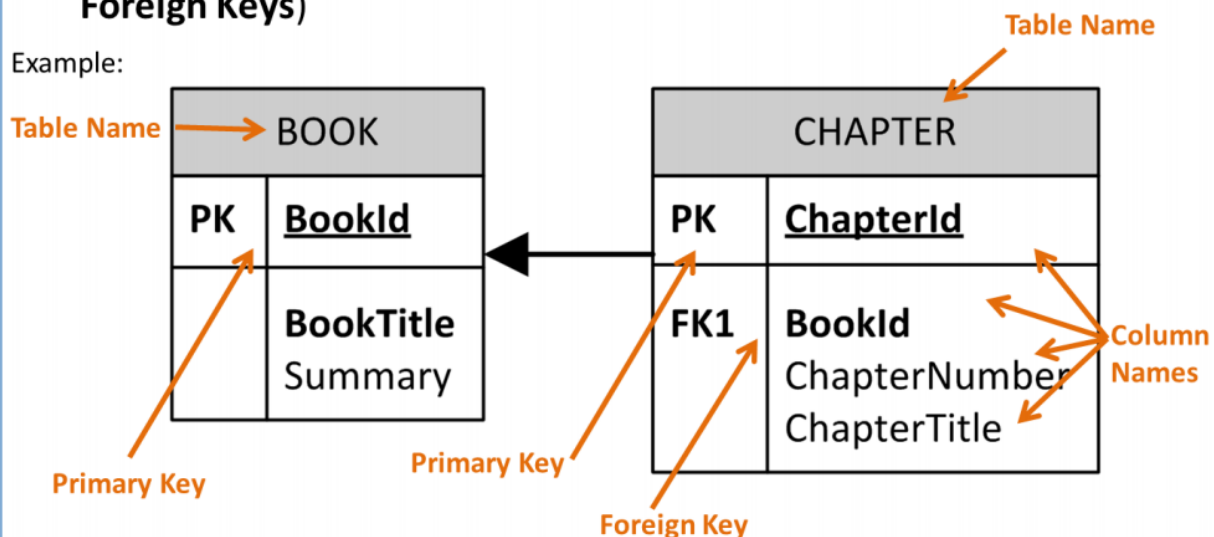
Before you start implementing your tables in the database, you should always spend some time design your tables properly using a design tool like, e.g., ERwin, Toad Data Modeler, PowerDesigner, Visio, etc. This is called Database Modeling.

Database Design – ER Diagram

ER Diagram (Entity-Relationship Diagram)

- Used for Design and Modeling of Databases.
- Specify Tables and relationship between them (**Primary Keys** and **Foreign Keys**)

Example:



Relational Database. In a relational database all the tables have one or more relation with each other using Primary Keys (PK) and Foreign Keys (FK). Note! You can only have one PK in a table, but you may have several FK's.

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	<input type="checkbox"/>
	LastName	varchar(50)	<input type="checkbox"/>
	FirstName	varchar(50)	<input type="checkbox"/>
	AreaCode	int	<input checked="" type="checkbox"/>
	Address	varchar(200)	<input checked="" type="checkbox"/>
	Phone	varchar(11)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

```
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, ...)

2. 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!

3. 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
```

4. 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.

4.1 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.

	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000					1111111
2	2	1001					2222222
3	3	1002	vanish	John	32	London	3333333

Primary Keys must contain unique numbers like this

4.2 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.

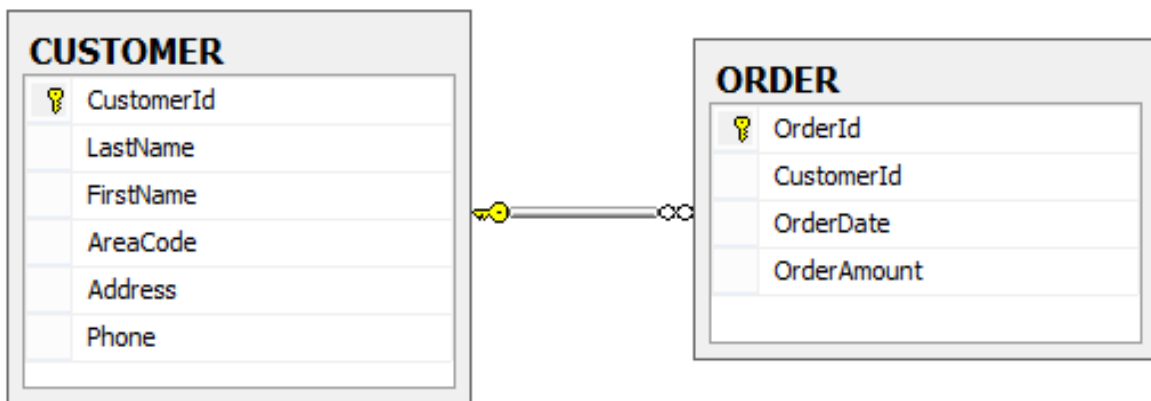
```
CREATE TABLE CUSTOMER
(
    CustomerId int IDENTITY(1,1) PRIMARY KEY,
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(200) NULL,
    Phone varchar(11) NULL,
)
```

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.

4.3 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.

4.4 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.

	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000					111111
2	2	1001					222222
3	3	1002					333333

Primary Keys must contain unique numbers like this

You can insert **NULL** values into columns with the **UNIQUE** constraint because **NULL** is the absence of a value, so it is never equal to other NULL values and **not considered a duplicate value**.

4.5 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.

4.6 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>0),
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(200) NULL DEFAULT 'Dhaka',
    Phone varchar(11) NULL,
)
```

5. 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 is updated to all records. So make sure to include the WHERE clause while using the UPDATE command!

6. 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!

7. 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.

8. 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
```

9. 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.

10. 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
```

11. 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
```