Informatics 2

Dmitriy Dunaev, M.Sc.

Structured Query Language

LAB TUTORIAL

for Laboratory Exercises 2-3

Table of Contents

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.

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 tutorial 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

In this Tutorial we will focus on Microsoft SQL Server. 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.

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:

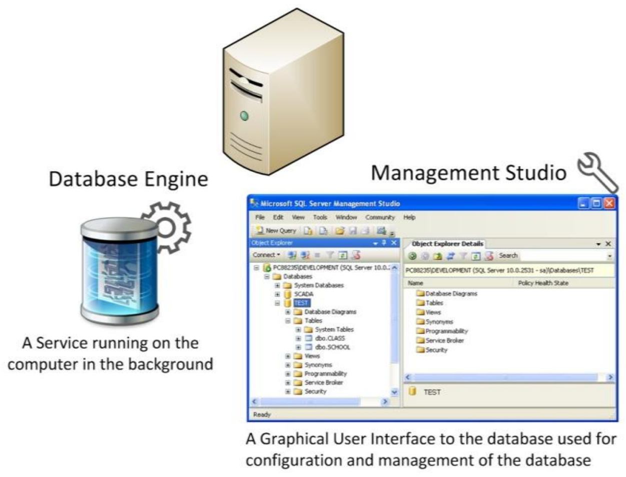
Operation	**SQL**	**Description**
Create	INSERT INTO	inserts new data into a database
Read (Retrieve)	SELECT	extracts data from a database
Update	UPDATE	updates data in a database
Delete (Destroy)	DELETE	deletes data from a database

2. *Introduction to SQL Server*

Microsoft is the vendor of SQL Server. There exist different editions of SQL Server, where SQL Server Express is free to download and use.

SQL Server uses T-SQL (Transact-SQL) that 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, data processing, mathematics, etc.

SQL Server consists of a **Database Engine** and a **Management Studio** (and lots of other programs, which we will not mention here). The Database engine has no graphical interface – it is just a service running in the background. The Management Studio is a graphical tool for configuring and viewing the information in the database.


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

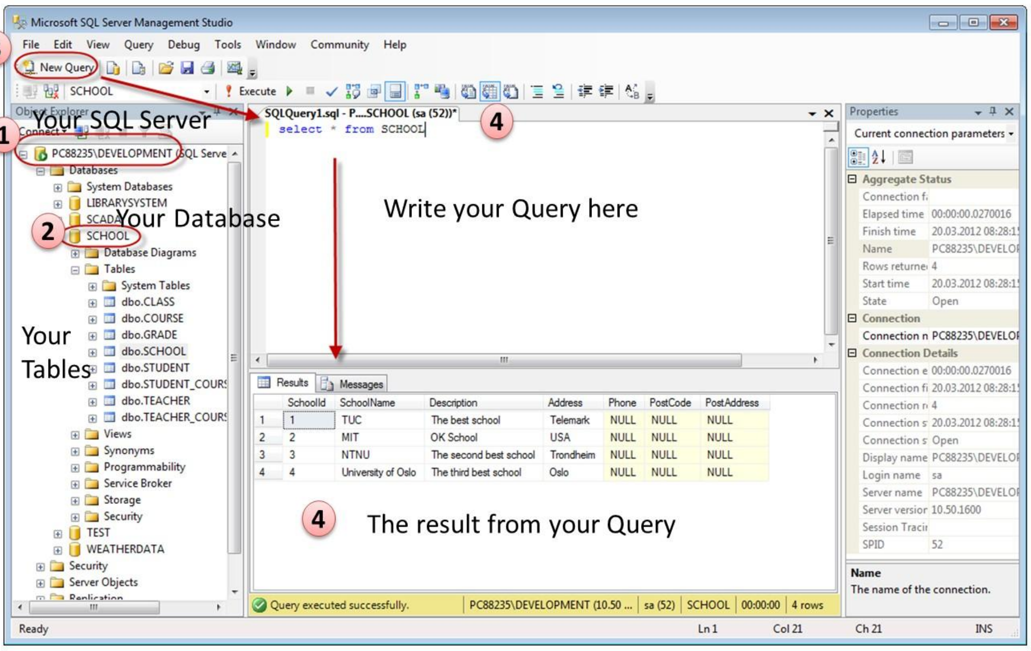
2.1 SQL Server Management Studio

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.

A central feature of SQL Server Management Studio is the Object Explorer, which allows 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 (see the figure below).

With SQL and the “Query Editor” we can do almost everything by writing scripts, however sometimes it can be more efficient to use special Designer tools in Management Studio.



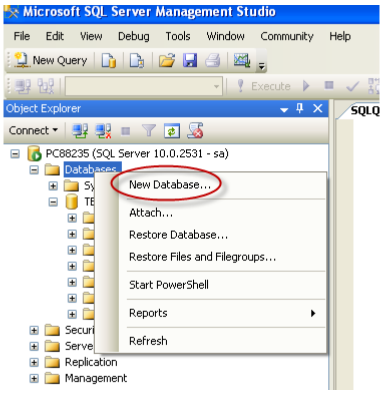
```
select * from SCHOOL
```

	SchoolId	SchoolName	Description	Address	Phone	PostCode	PostAddress
1	1	TUC	The best school	Telemark	NULL	NULL	NULL
2	2	MIT	OK School	USA	NULL	NULL	NULL
3	3	NTNU	The second best school	Trondheim	NULL	NULL	NULL
4	4	University of Oslo	The third best school	Oslo	NULL	NULL	NULL

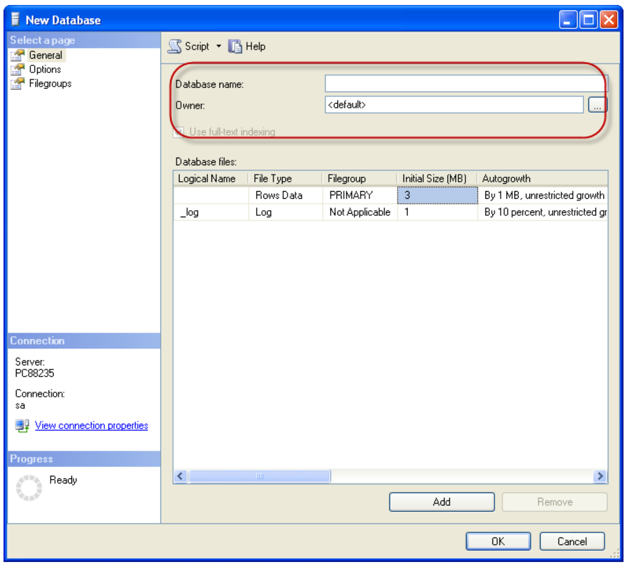
```
Query executed successfully.    PC88235\DEVELOPMENT (10.50 ...   sa (52)   SCHOOL   00:00:00   4 rows
```

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 on the Lab is the name of your database:



Database name:

Owner: <default>

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth
	Rows Data	PRIMARY	3	By 1 MB, unrestricted growth
_log	Log	Not Applicable	1	By 10 percent, unrestricted gr

Server:
PC88235

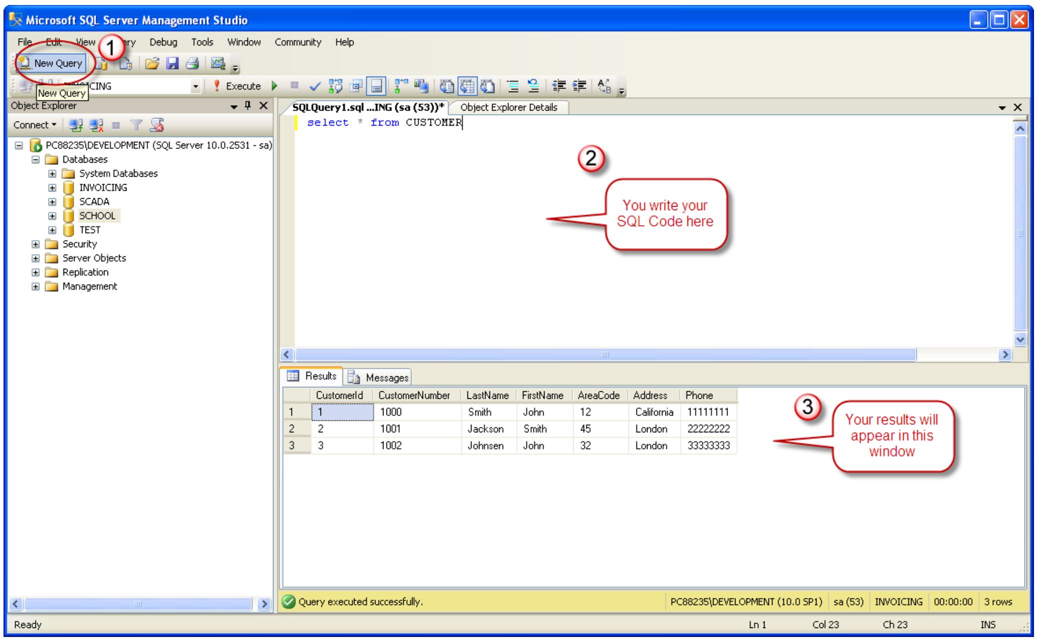
Connection:
sa

Ready

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

2.1.2 Queries

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


Microsoft SQL Server Management Studio
File Edit View Query Debug Tools Window Community Help
New Query
1
Execute
Object Explorer
Connect
PC88235\DEVELOPMENT (SQL Server 10.0.2531 - sa)
Databases
System Databases
INVOICING
SCADA
SCHOOL
TEST
Security
Server Objects
Replication
Management
SQLQuery1.sql ...ING (sa (53))*
Object Explorer Details
2
You write your SQL Code here
Results
Messages
3
Your results will appear in this window
Ready
Ln 1
Col 23
Ch 23
INS

```
select * from CUSTOMER
```

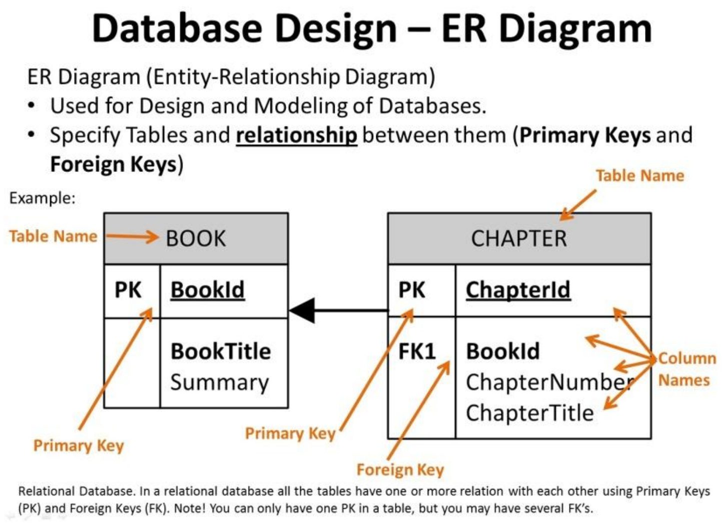
	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

Query executed successfully. PC88235\DEVELOPMENT (10.0 SP1) sa (53) INVOICING 00:00:00 3 rows

Here we can write any query that is supported by the SQL language.

3. *Creating Tables*

Before you start creating tables in your database, you should always spend some time with designing your tables. This step 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:

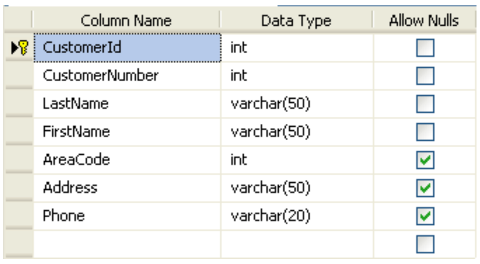
```
CREATE TABLE table_name
(
column_name1  data_type,
column_name2  data_type,
column_name3 data_type,
....
)
```

The data type specifies what type of data the column can hold. There are special data types for numbers, text dates, etc. Example:

- 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	☐
CustomerNumber	int	☐
LastName	varchar(50)	☐
FirstName	varchar(50)	☐
AreaCode	int	✓
Address	varchar(50)	✓
Phone	varchar(20)	✓
		☐

```
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,
AreaCode int NULL,
Address varchar(50) NULL,
Phone varchar(50) NULL,
)
```

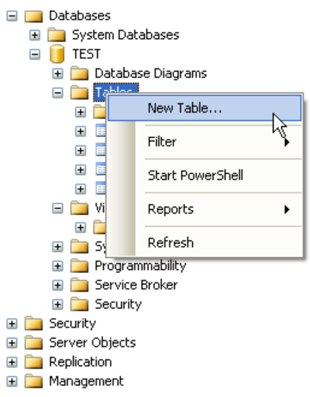
Best practice. When creating tables you should consider the following:

- Tables: Use uppercase 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.
    - Consider using 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
- Use English language for table and column names
- Avoid abbreviations! (Use RoomNumber – not RoomNo, RoomNr, RN, ...)

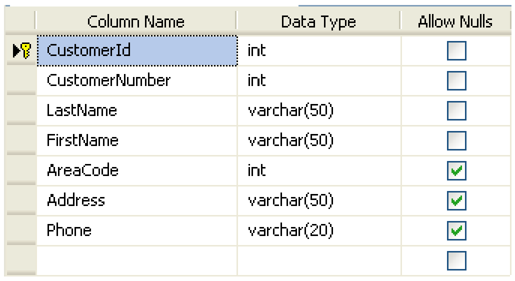
3.1 Creating Tables using the Designer Tools

Instead of writing scripts you may use the designer for creating tables.

Step 1: Select "New Table ...":



Step 2: The table designer pops up, in which you can add columns, data types, etc.



Column Name	Data Type	Allow Nulls
CustomerId	int	☐
CustomerNumber	int	☐
LastName	varchar(50)	☐
FirstName	varchar(50)	☐
AreaCode	int	✓
Address	varchar(50)	✓
Phone	varchar(20)	✓
		☐

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

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

3.2 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 details.

3.2.1 PRIMARY KEY

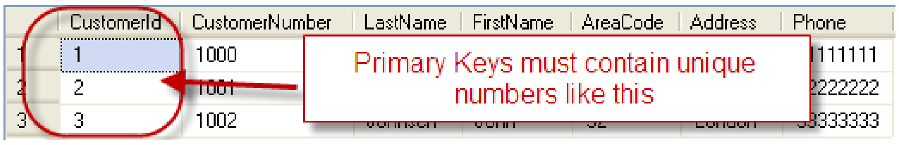
The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain unique values. It is normal to 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 will be incremented by 1 at each step.

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,
CustomerNumber int NOT NULL UNIQUE,
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(50) NULL,
Phone varchar(50) NULL,
)
```

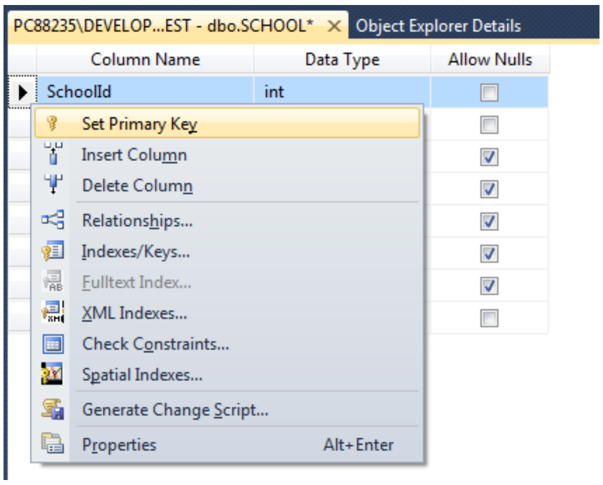
As you see we use the "Primary Key" keyword to specify that a column should be the Primary Key.



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

Setting Primary Keys in the Designer Toolю

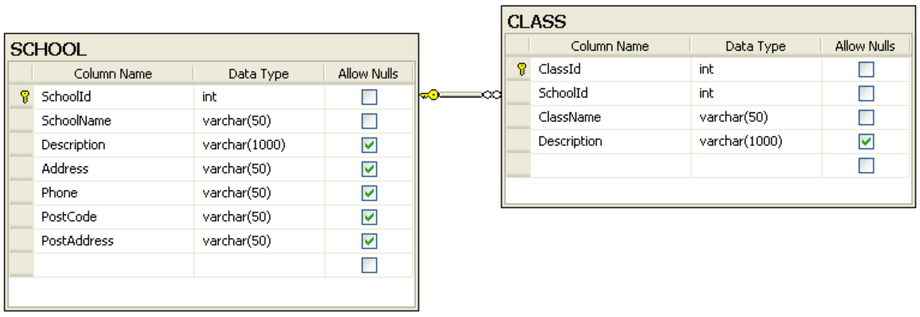
If you use the Designer tools in SQL Server you can easily set the primary Key in a table just by right-click and select "Set primary Key".



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

3.2.2 FOREIGN KEY

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



SCHOOL

	Column Name	Data Type	Allow Nulls
🔑	SchoolId	int	☐
	SchoolName	varchar(50)	☐
	Description	varchar(1000)	✓
	Address	varchar(50)	✓
	Phone	varchar(50)	✓
	PostCode	varchar(50)	✓
	PostAddress	varchar(50)	✓
			☐

CLASS

	Column Name	Data Type	Allow Nulls
🔑	ClassId	int	☐
	SchoolId	int	☐
	ClassName	varchar(50)	☐
	Description	varchar(1000)	✓
			☐

We will create a CREATE TABLE script for these tables:

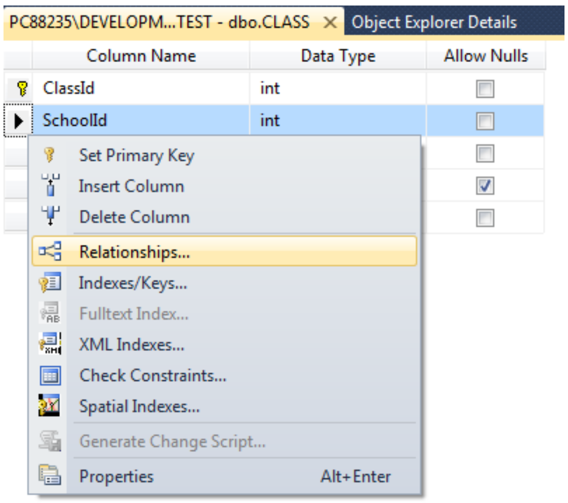
```
CREATE TABLE SCHOOL (
SchoolId int IDENTITY(1,1) PRIMARY KEY,
SchoolName varchar(50) NOT NULL UNIQUE,
Description varchar(1000) NULL,
Address varchar(50) NULL,
Phone varchar(50) NULL,
PostCode varchar(50) NULL,
PostAddress varchar(50) NULL
)
```

```
CREATE TABLE CLASS (
ClassId int IDENTITY(1,1) PRIMARY KEY,
SchoolId int NOT NULL FOREIGN KEY REFERENCES SCHOOL (SchoolId),
ClassName varchar(50) NOT NULL UNIQUE,
Description varchar(1000) NULL,
)
```

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

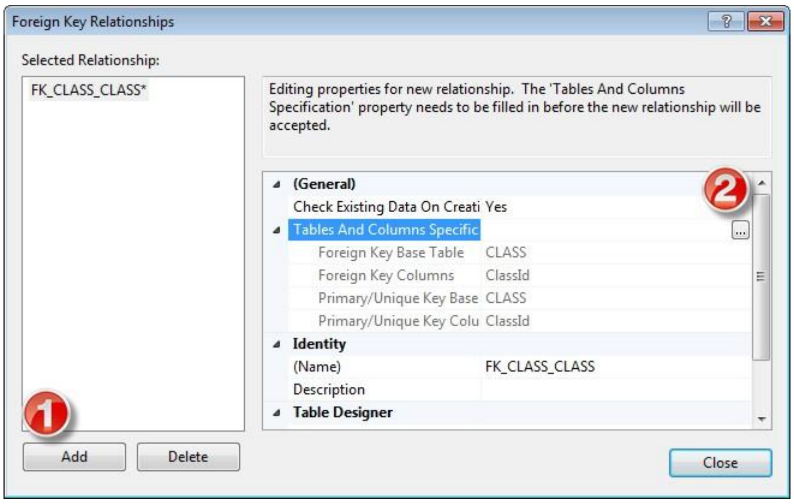
Setting Foreign Keys in the Designer Tools:

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



Column Name	Data Type	Allow Nulls
ClassId	int	☐
SchoolId	int	☐

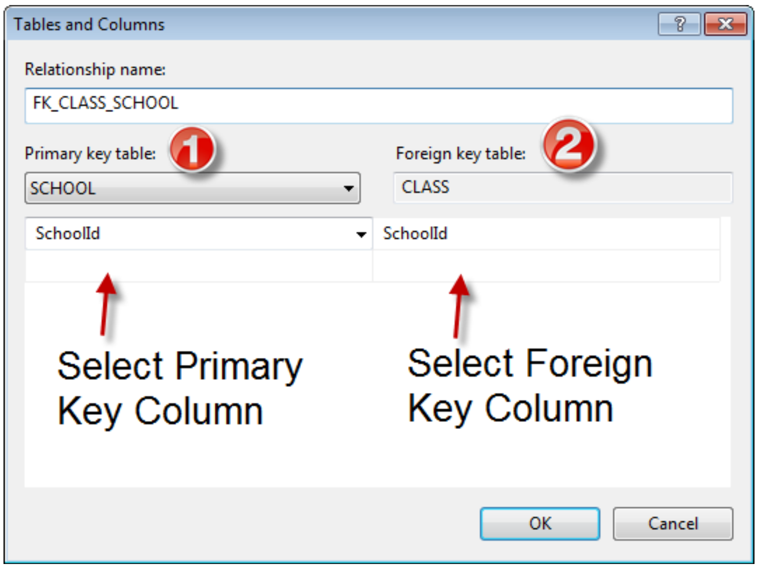
The following window pops up (Foreign Key Relationships):



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 Creati	Yes
Tables And Columns Specific	
Foreign Key Base Table	CLASS
Foreign Key Columns	ClassId
Primary/Unique Key Base	CLASS
Primary/Unique Key Colu	ClassId
Identity	
(Name)	FK_CLASS_CLASS
Description	
Table Designer	

Click on the “Add” button and then click on the small “…” button. Then the following window pops up (Tables and Columns):



Relationship name: FK_CLASS_SCHOOL

Primary key table:	Foreign key table:
SCHOOL	CLASS
SchoolId	SchoolId

Here you specify the primary Key Column in the Primary Key table and the Foreign Key Column in the Foreign Key table.

3.2.3 NOT NULL / Required Columns

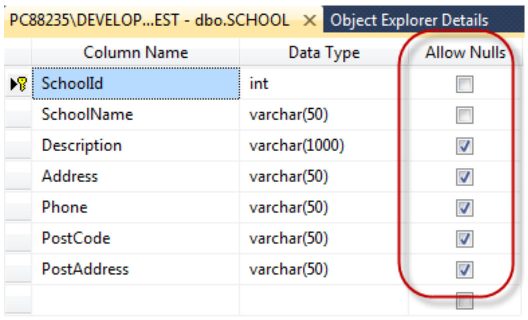
The NOT NULL constraint enforces a column to NOT accept NULL values, that is it enforces a field to always contain a value. Therefore, you cannot insert a new record, or update a record without adding a value to this field.

If we take a closer look at the CUSTOMER table created earlier, we see that “CustomerNumber”, “LastName” and “FirstName” are set to “NOT NULL”, this means these columns needs to contain data. While “AreaCode”, “Address” and “Phone” may be left empty, i.e, they don’t need to be filled out.

Note! A primary key column cannot contain NULL values.

Setting NULL/NOT NULL in the Designer Tools

In the Table Designer you can easily set which columns should allow NULL as a value:



PC88235\DEVELOP...EST - dbo.SCHOOL | Object Explorer Details

Column Name	Data Type	Allow Nulls
SchoolId	int	☐
SchoolName	varchar(50)	☐
Description	varchar(1000)	☑
Address	varchar(50)	☑
Phone	varchar(50)	☑
PostCode	varchar(50)	☑
PostAddress	varchar(50)	☑
		☐

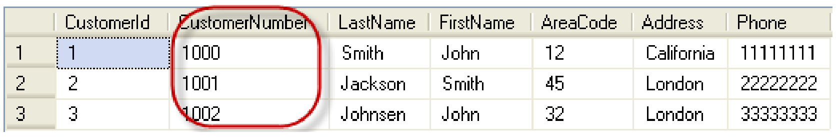
3.2.4 UNIQUE

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.

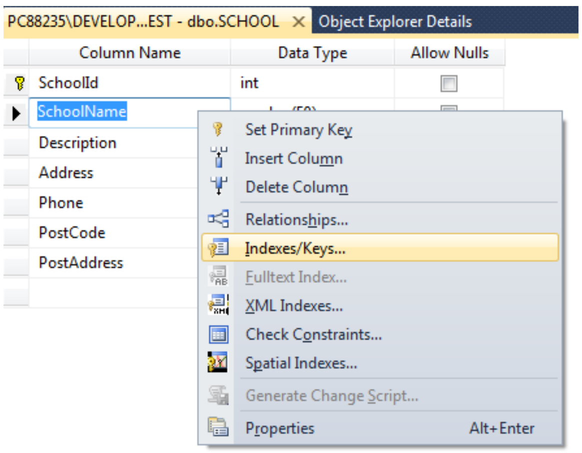
If we take a closer look at the CUSTOMER table created earlier, we see that the "CustomerNumber" is set to UNIQUE, meaning each customer must have a unique Customer Number.



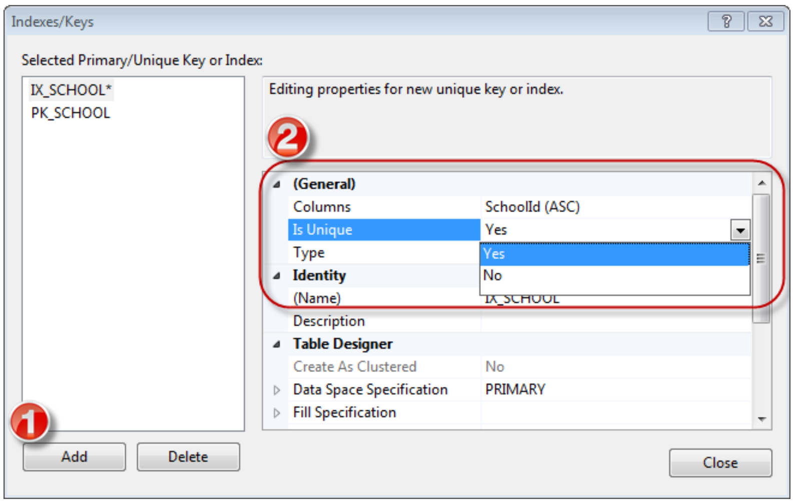
	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

Setting UNIQUE in the Designer Tools:

If you want to use the designer, right-click on the column that you want to be UNIQUE and select "Indexes/Keys…":



Then click "Add" and set the "Is Unique" property to "Yes":



Selected Primary/Unique Key or Index:

IX_SCHOOL*
PK_SCHOOL

Editing properties for new unique key or index.

(General)	
Columns	SchoolId (ASC)
Is Unique	Yes
Type	
Identity	
(Name)	IX_SCHOOL
Description	
Table Designer	
Create As Clustered	No
Data Space Specification	PRIMARY
Fill Specification	

Yes
No

3.2.5 CHECK

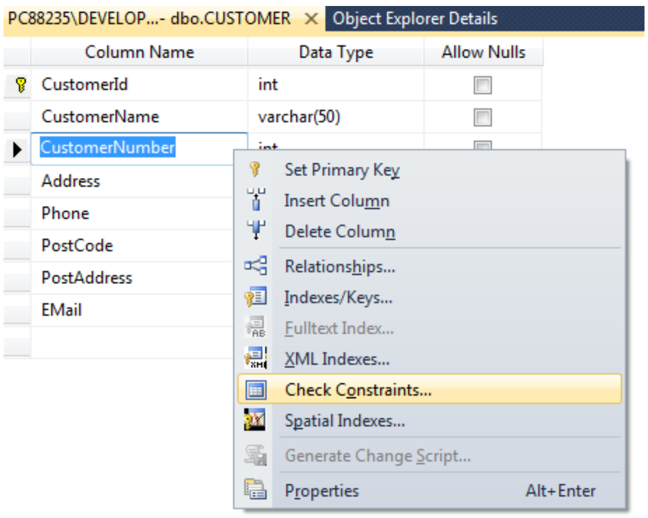
The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

```
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(50) NULL,
Phone varchar(50) NULL,
)
```

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

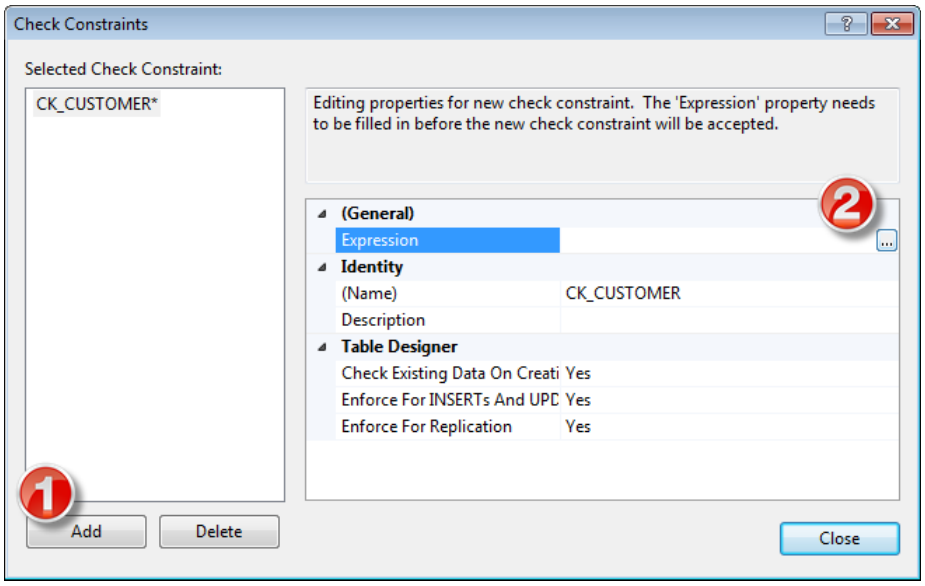
Setting CHECK constraints in the Designer Tools:

If you want to use the designer, right-click on the column where you want to set the constraints and select "Check Constraints…":



Column Name	Data Type	Allow Nulls
CustomerId	int	☐
CustomerName	varchar(50)	☐
CustomerNumber	int	☐
Address		
Phone		
PostCode		
PostAddress		
EMail		

Then click "Add" and then click "…" in order to open the Expression window:



Check Constraints

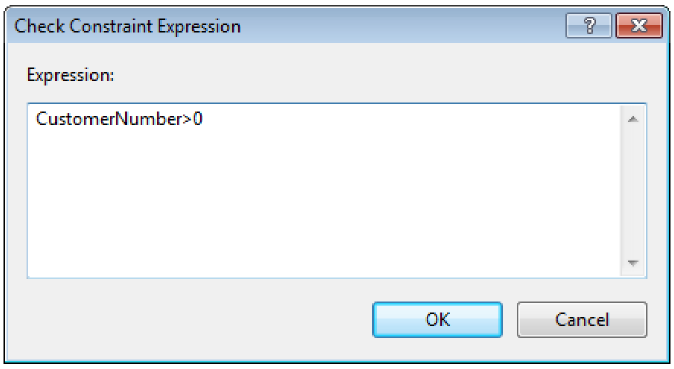
Selected Check Constraint:

CK_CUSTOMER*

Editing properties for new check constraint. The 'Expression' property needs to be filled in before the new check constraint will be accepted.

(General)	
Expression	
Identity	
(Name)	CK_CUSTOMER
Description	
Table Designer	
Check Existing Data On Creati	Yes
Enforce For INSERTs And UPD	Yes
Enforce For Replication	Yes

In the Expression window you can type in the expression you want to use:



Check Constraint Expression

Expression:

```
CustomerNumber>0
```

3.2.6 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.

```
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,
Country varchar(10) DEFAULT 'Hungary',
AreaCode int NULL,
Address varchar(50) NULL,
Phone varchar(50) NULL,
)
```

3.2.7 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,
CustomerNumber int NOT NULL UNIQUE,
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(50) NULL,
Phone varchar(50) NULL
)
```

IDENTITY(1,1) means the first CustomerID value will be 1 and then it will incremented by 1.

4. *INSERT command*

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 ('1000', 'Smith', 'John', 12, 'California',
'11111111')
```

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

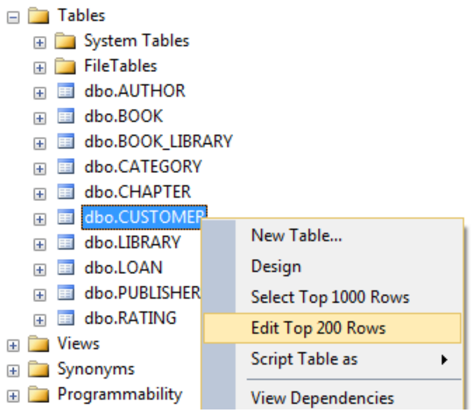
```
INSERT INTO table_name (column1, column2, column3,...) VALUES (value1,
value2, value3,...)
```

The second form is recommended!

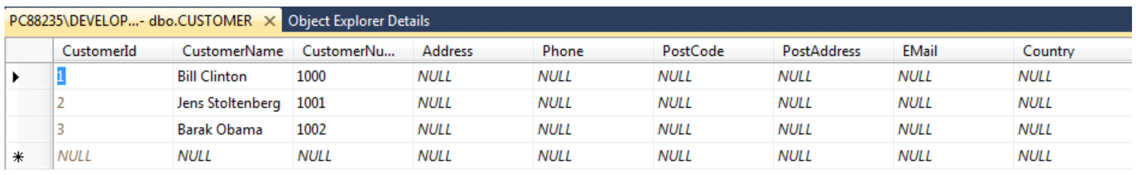
Example:

```
INSERT INTO CUSTOMER (CustomerNumber, LastName, FirstName, AreaCode,
Address, Phone)
VALUES ('1000', 'Smith', 'John', 12, 'California', '11111111')
```

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:



CustomerId	CustomerName	CustomerNu...	Address	Phone	PostCode	PostAddress	EMail	Country
1	Bill Clinton	1000	NULL	NULL	NULL	NULL	NULL	NULL
2	Jens Stoltenberg	1001	NULL	NULL	NULL	NULL	NULL	NULL
3	Barak Obama	1002	NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

5. *UPDATE command*

The UPDATE statement is used to update existing records in a table. The syntax is as follows:

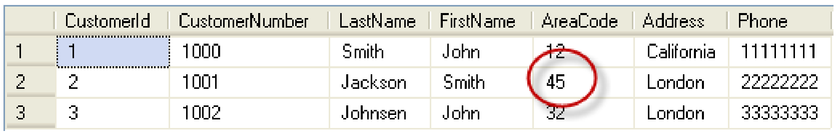
```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

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

Example:

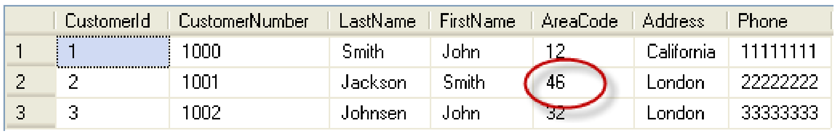
```
update CUSTOMER set AreaCode= where CustomerId=2
```

Before update:



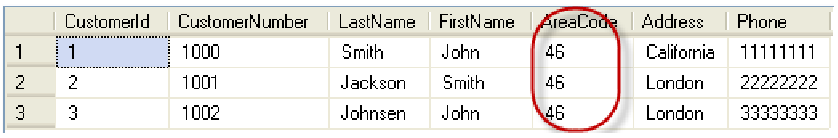
	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

After update:



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	46	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

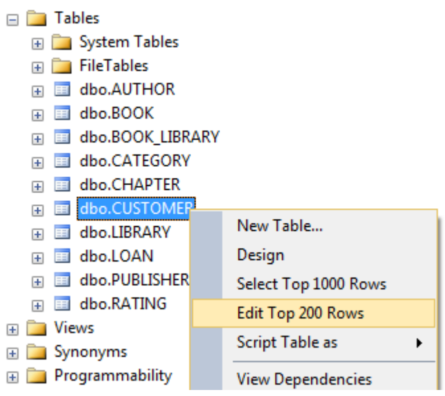
If you don't include the WHERE clause the result becomes:



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	46	California	11111111
2	2	1001	Jackson	Smith	46	London	22222222
3	3	1002	Johnsen	John	46	London	33333333

Update Data in the Designer Tools:

The same way you insert data you can also update the data. Right-click on the specific table and select "Edit Top 200 Rows":



Then you can change your data in table view.

6. *DELETE command*

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

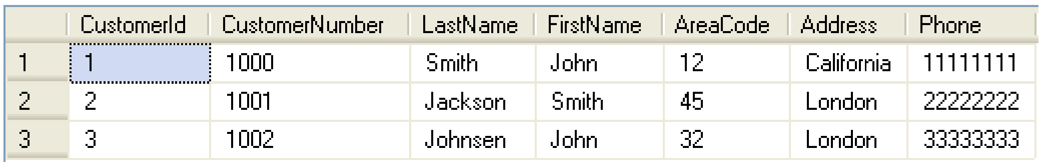
```
DELETE FROM table_name
WHERE some_column=some_value
```

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

Example:

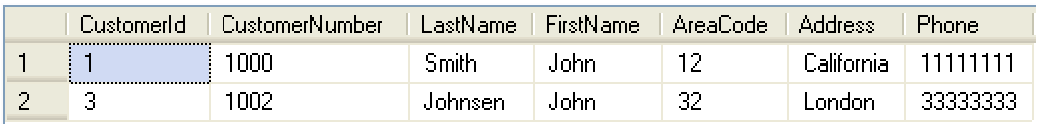
```
delete from CUSTOMER where CustomerId=2
```

Before delete:



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

After delete:



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	3	1002	Johnsen	John	32	London	33333333

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 need it! You cannot UNDO this command!

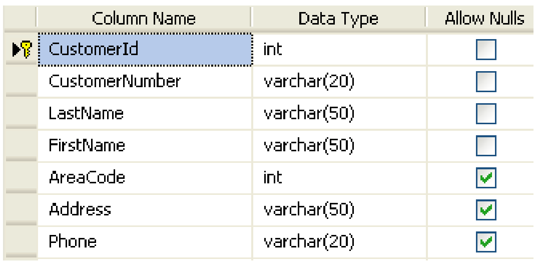
Delete Data in the Designer Tools.

You can delete data in the designer by right-clicking on the row and selecting "Delete".

7. SELECT command

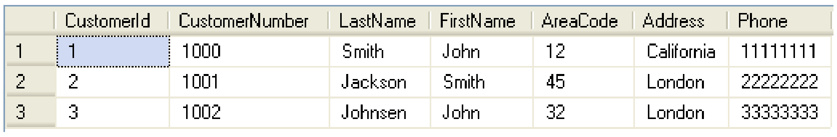
The SELECT statement is probably the most widely used SQL command. The SELECT statement is used for retrieving rows from the database and enables the selection of one or many rows or columns from one or many tables in the database.

We will use the CUSTOMER table with the following columns:



Column Name	Data Type	Allow Nulls
CustomerId	int	☐
CustomerNumber	varchar(20)	☐
LastName	varchar(50)	☐
FirstName	varchar(50)	☐
AreaCode	int	✓
Address	varchar(50)	✓
Phone	varchar(20)	✓

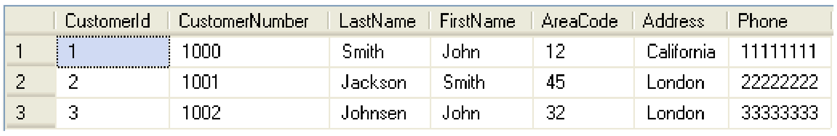
The CUSTOMER table contains the following data:



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

Example:

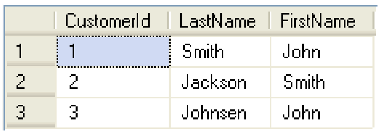
```
select * from CUSTOMER
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

This simple example gets all the data in the table CUSTOMER. The asterisk 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:

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



	CustomerId	LastName	FirstName
1	1	Smith	John
2	2	Jackson	Smith
3	3	Johnsen	John

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

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

Note! SQL is not case sensitive. "SELECT" will result in the same as "select".

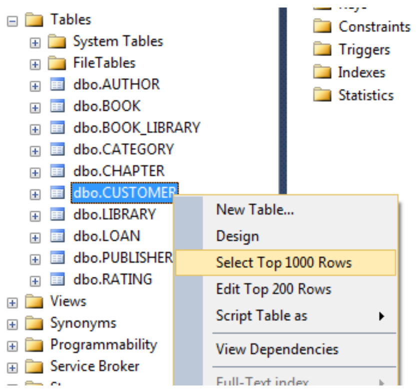
The full syntax of the SELECT statement is somewhat complex, but the main clauses can be summarized as:

```
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 ] ]
```

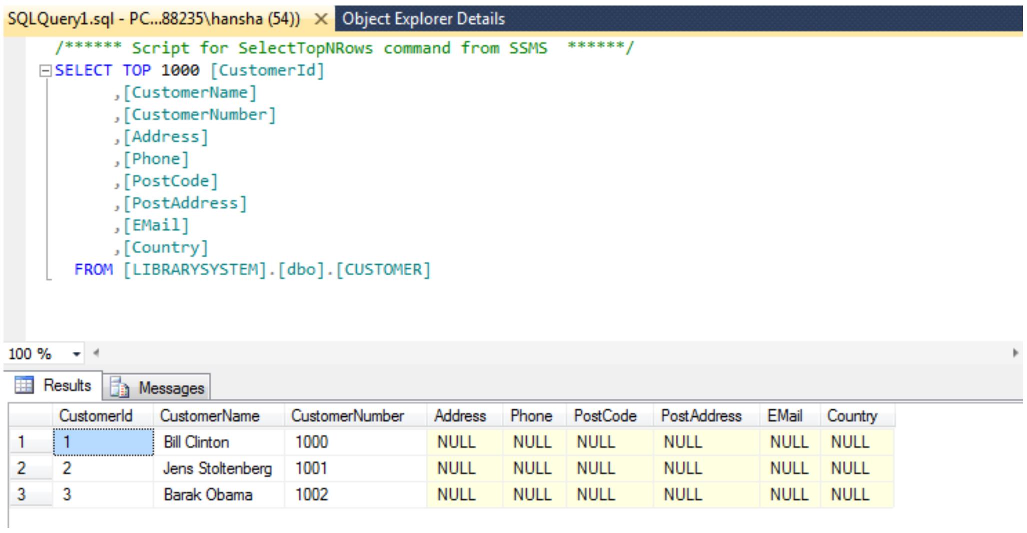
It seems complex, but we will take the different parts step by step in the next sections.

Select Data in the Designer Tools:

Right-click on a table and select “Select Top 1000 Rows”:



The following result will appear:



```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [CustomerId]
      ,[CustomerName]
      ,[CustomerNumber]
      ,[Address]
      ,[Phone]
      ,[PostCode]
      ,[PostAddress]
      ,[EMail]
      ,[Country]
  FROM [LIBRARYSYSTEM].[dbo].[CUSTOMER]
```

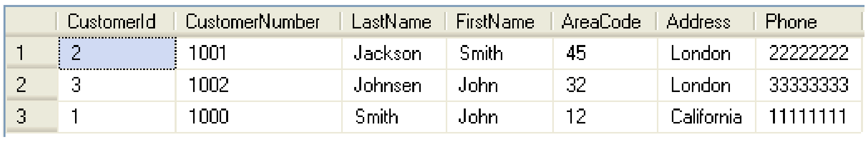
	CustomerId	CustomerName	CustomerNumber	Address	Phone	PostCode	PostAddress	EMail	Country
1	1	Bill Clinton	1000	NULL	NULL	NULL	NULL	NULL	NULL
2	2	Jens Stoltenberg	1001	NULL	NULL	NULL	NULL	NULL	NULL
3	3	Barak Obama	1002	NULL	NULL	NULL	NULL	NULL	NULL

7.1 The ORDER BY Keyword

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

Example:

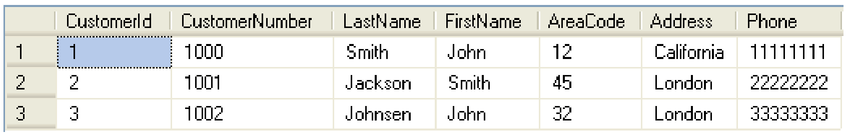
```
select * from CUSTOMER order by LastName
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	2	1001	Jackson	Smith	45	London	22222222
2	3	1002	Johnsen	John	32	London	33333333
3	1	1000	Smith	John	12	California	11111111

You may also sort by several columns:

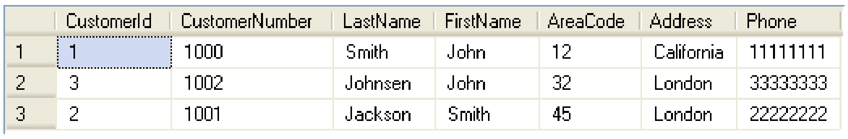
```
select * from CUSTOMER order by Address, LastName
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

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
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	3	1002	Johnsen	John	32	London	33333333
3	2	1001	Jackson	Smith	45	London	22222222

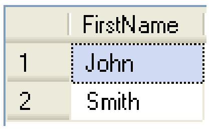
7.2 The DISTINCT Keyword

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

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

Example:

```
select distinct FirstName from CUSTOMER
```



	FirstName
1	John
2	Smith

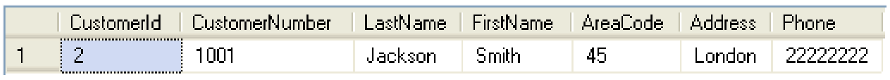
7.3 The WHERE Clause

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

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

Example:

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



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	2	1001	Jackson	Smith	45	London	22222222

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

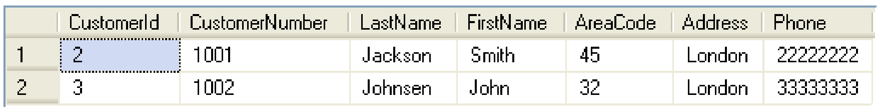
7.3.1 Operators

Within 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
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	2	1001	Jackson	Smith	45	London	22222222
2	3	1002	Johnsen	John	32	London	33333333

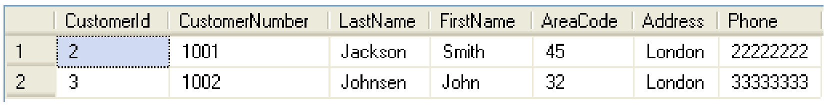
7.3.2 LIKE Operator

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

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

Example:

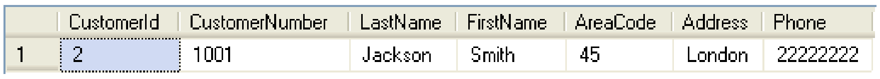
```
select * from CUSTOMER where LastName like 'J%'
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	2	1001	Jackson	Smith	45	London	22222222
2	3	1002	Johnsen	John	32	London	33333333

Note! The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

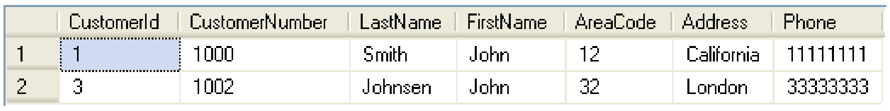
```
select * from CUSTOMER where LastName like '%a%'
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	2	1001	Jackson	Smith	45	London	22222222

You may also combine with the NOT keyword, example:

```
select * from CUSTOMER where LastName not like '%a%'
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	3	1002	Johnsen	John	32	London	33333333

7.3.3 IN Operator

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

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

7.3.4 BETWEEN Operator

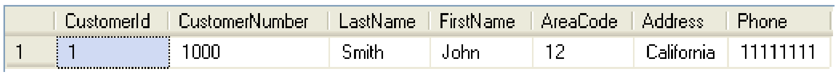
The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

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

7.4 AND & OR Operators

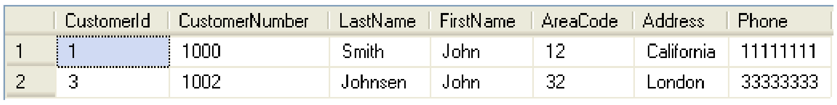
The AND operator displays a record if both the first condition and the second condition is true.
The OR operator displays a record if either the first condition or the second condition is true.

```
select * from CUSTOMER where LastName='Smith' and FirstName='John'
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111

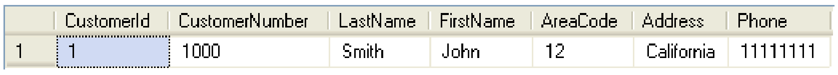
```
select * from CUSTOMER where LastName='Smith' or FirstName='John'
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	3	1002	Johnsen	John	32	London	33333333

Combining AND & OR:

```
select * from CUSTOMER
where LastName='Smith' and (FirstName='John' or FirstName='Smith')
```



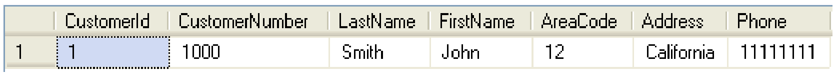
	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111

7.5 SELECT TOP Clause

The TOP clause is used to specify the number of records to return. It can be very useful on large tables with thousands of records. Returning a large number of records can have impact on performance.

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

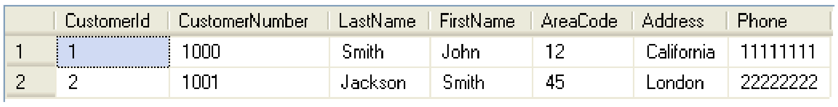
```
select TOP 1 * from CUSTOMER
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111

You can also specify it in percent:

```
select TOP 60 percent * from CUSTOMER
```



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222

7.6 Alias

You can give a table or a column another name by using an alias. This can be a good thing to do if you have very long or complex table names or column names. An alias name could be anything, but usually it is some short name that speaks for itself.

SQL Alias Syntax for Tables:

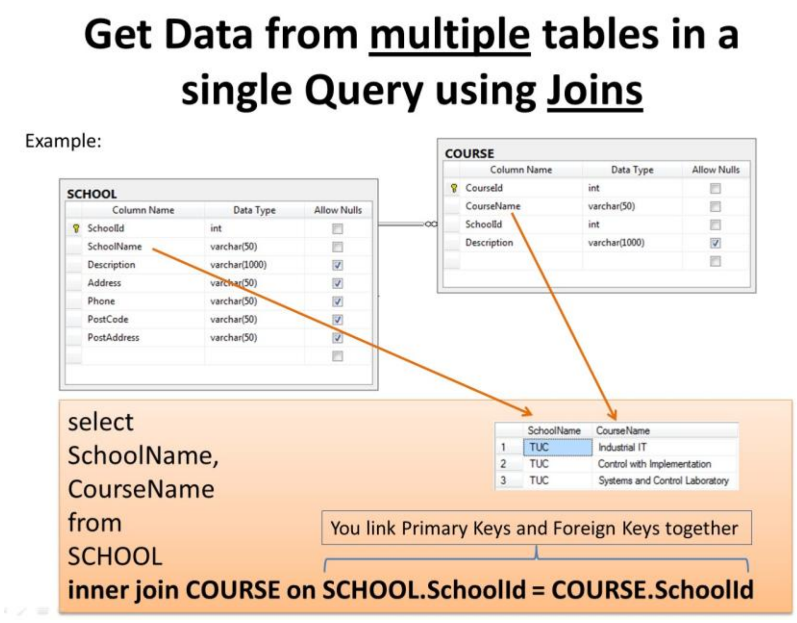
```
SELECT column_name(s)
FROM table_name
AS alias_name
```

SQL Alias Syntax for Columns:

```
SELECT column_name AS alias_name
FROM table_name
```

7.7 Joins

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



Get Data from multiple tables in a single Query using Joins

Example:

SCHOOL

	Column Name	Data Type	Allow Nulls
🔑	SchoolId	int	☐
	SchoolName	varchar(50)	☐
	Description	varchar(1000)	✓
	Address	varchar(50)	✓
	Phone	varchar(50)	✓
	PostCode	varchar(50)	✓
	PostAddress	varchar(50)	✓
			☐

COURSE

	Column Name	Data Type	Allow Nulls
🔑	CourseId	int	☐
	CourseName	varchar(50)	☐
	SchoolId	int	☐
	Description	varchar(1000)	✓
			☐

```
select
SchoolName,
CourseName
from
SCHOOL
inner join COURSE on SCHOOL.SchoolId = COURSE.SchoolId
```

	SchoolName	CourseName
1	TUC	Industrial IT
2	TUC	Control with Implementation
3	TUC	Systems and Control Laboratory

You link Primary Keys and Foreign Keys together

7.7.1 Different SQL JOINs

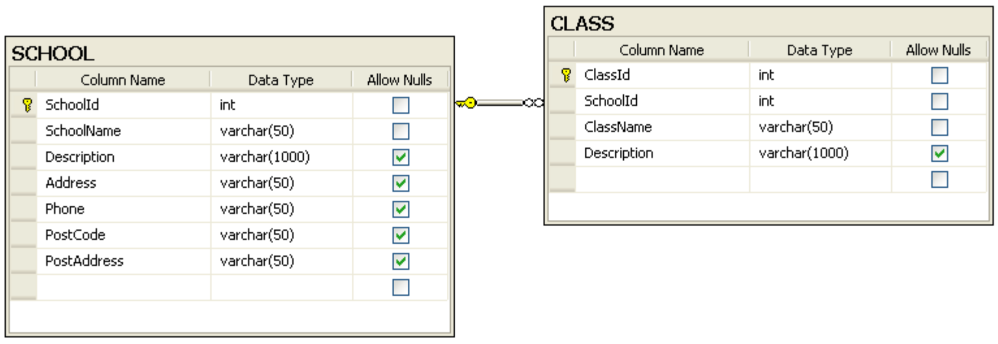
Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

- 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

Example:

Given 2 tables: SCHOOL and CLASS

The diagram is shown below:



SCHOOL

	Column Name	Data Type	Allow Nulls
🔑	SchoolId	int	☐
	SchoolName	varchar(50)	☐
	Description	varchar(1000)	✓
	Address	varchar(50)	✓
	Phone	varchar(50)	✓
	PostCode	varchar(50)	✓
	PostAddress	varchar(50)	✓
			☐

CLASS

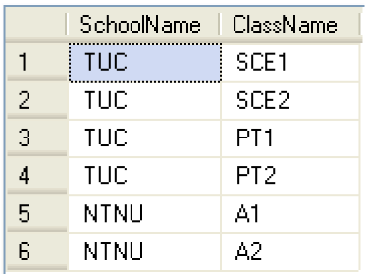
	Column Name	Data Type	Allow Nulls
🔑	ClassId	int	☐
	SchoolId	int	☐
	ClassName	varchar(50)	☐
	Description	varchar(1000)	✓
			☐

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
```



	SchoolName	ClassName
1	TUC	SCE1
2	TUC	SCE2
3	TUC	PT1
4	TUC	PT2
5	NTNU	A1
6	NTNU	A2

8. *Functions*

With SQL and SQL Server you can use lots of built-in functions or you may create your own functions. Here we will learn to use some built-in functions.

8.1 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:

- 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:

- 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.

8.1.1 String Functions

Here are some useful functions used to manipulate strings in SQL Server:

- CHAR
- CHARINDEX
- REPLACE
- SUBSTRING
- LEN
- REVERSE
- LEFT / RIGHT
- LOWER / UPPER
- LTRIM / RTRIM

Read more about these functions in the SQL Server Help.

8.1.2 Date and Time Functions

Here are some useful Date and Time functions in SQL Server:

- DATEPART
- GETDATE
- DATEADD
- DATEDIFF
- DAY
- MONTH
- YEAR
- ISDATE

Read more about these functions in the SQL Server Help.

8.1.3 Mathematics and Statistics Functions

Here are some useful functions for mathematics and statistics in SQL Server:

- COUNT
- MIN, MAX
- COS, SIN, TAN
- SQRT
- STDEV
- MEAN
- AVG

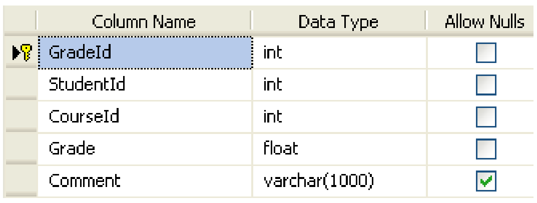
Read more about these functions in the SQL Server Help.

8.1.4 AVG()

The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name) FROM table_name
```

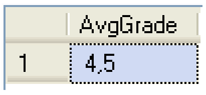
Given a GRADE table:



Column Name	Data Type	Allow Nulls
GradeId	int	☐
StudentId	int	☐
CourseId	int	☐
Grade	float	☐
Comment	varchar(1000)	✓

We want to find the average grade for a specific student:

```
select AVG(Grade) as AvgGrade from GRADE where StudentId=1
```



	AvgGrade
1	4,5

8.1.5 COUNT()

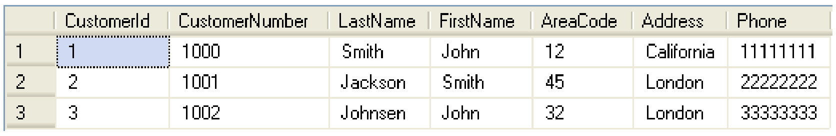
The COUNT() function returns the number of rows that matches a specified criteria. The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name
```

The COUNT(*) function returns the number of records in a table:

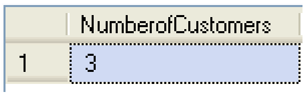
```
SELECT COUNT(*) FROM table_name
```

Let us use the CUSTOMER table as an example:



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

```
select COUNT(*) as NumbersofCustomers from CUSTOMER
```



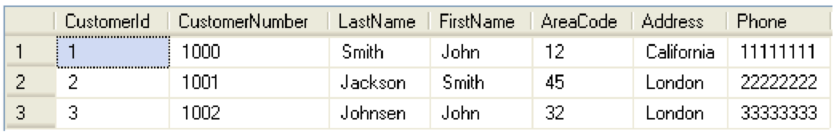
	NumberofCustomers
1	3

8.1.6 The GROUP BY Statement

Aggregate functions are often accompanied by a GROUP BY statement. It is used in conjunction with the aggregate functions to group the result-set by one or more columns.

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

Let us use the CUSTOMER table as an example:



	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

If we try the following:

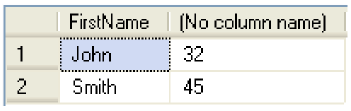
```
select FirstName, MAX(AreaCode) from CUSTOMER
```

And get the following error message:

Column 'CUSTOMER.FirstName' 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 the GROUP BY:

```
select FirstName, MAX(AreaCode) from CUSTOMER
group by FirstName
```



	FirstName	(No column name)
1	John	32
2	Smith	45

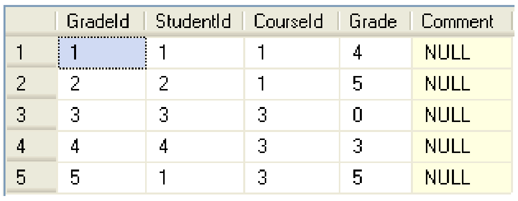
8.1.7 The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

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

Let us use the GRADE table as an example:

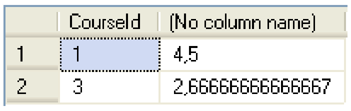
```
select * from GRADE
```



	GradeId	StudentId	CourseId	Grade	Comment
1	1	1	1	4	NULL
2	2	2	1	5	NULL
3	3	3	3	0	NULL
4	4	4	3	3	NULL
5	5	1	3	5	NULL

First we use the GROUP BY statement:

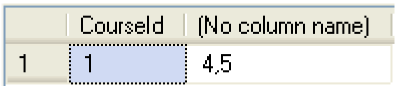
```
select CourseId, AVG(Grade) from GRADE
group by CourseId
```



	CourseId	(No column name)
1	1	4,5
2	3	2,66666666666667

Compare with the following query:

```
select CourseId, AVG(Grade) from GRADE
group by CourseId
having AVG(Grade)>3
```



	CourseId	(No column name)
1	1	4,5

9. *Sample Review Questions*

1. What is an SQL? What is DDL and DML? What is Transact-SQL?
2. List the most important constraints used at table creation. Explain in details three of them at your choice.
3. What is a PRIMARY KEY and a FOREIGN KEY? Define and provide examples.
4. What is a CHECK constraint used for? Show it on an example.
5. What is the syntax of the operation to add a new record to a table?
6. What is the syntax of the operation to modify a record in a table?
7. What is the syntax of the operation to remove a record from a table?
8. What are ORDER BY and DISTINCT clauses used for? Provide the syntax.
9. What operators can be used within the WHERE clause? Explain in details three of them at your choice.
10. What clause do we use to specify the number of records to return? Provide the syntax. Show an example.
11. List 4 types of JOINs. What is the difference between them? Explain in details two of them at your choice, providing examples.[1]
12. List at least 5 aggregate functions. Briefly describe each of them.
13. List at least 5 scalar functions. Briefly describe each of them.
14. What is GROUP BY used for? Provide example.
15. What is HAVING used for? Provide example.
16. … other similar questions may be asked as well…

[1] This question will not be asked on Lab 2.