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.

The CUSTOMER table has 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 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
```

	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>
```

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

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

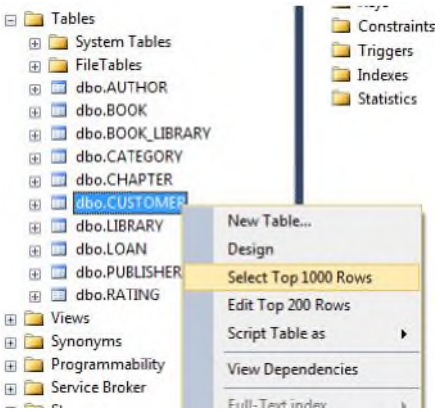
The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

```
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 ] ]
```

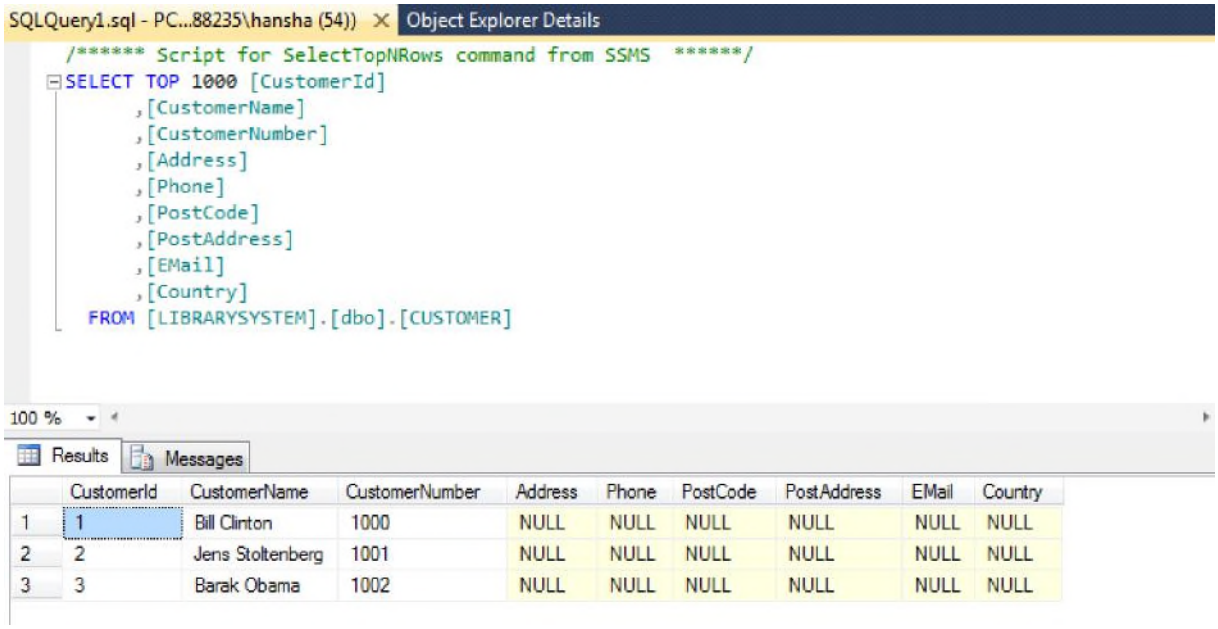
It seems complex, but we will take the different parts step by step in the next sections.

Select Data in the Designer Tools:

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



The following will appear:



	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

A Select query is automatically created for you which you can edit if you want to.

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, e.g. like this:

```
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

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
```

	FirstName
1	John
2	Smith

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'
```

	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.

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
```

	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.

Syntax:

```
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.

Syntax:

```
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.

Syntax:

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

7.4 Wildcards

SQL wildcards can substitute for one or more characters when searching for data in a database.

Note! SQL wildcards must be used with the SQL LIKE operator.

With SQL, the following wildcards can be used:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character
[charlist]	Any single character in charlist
[^charlist]or **[!charlist]**	Any single character not in charlist

Examples:

```
SELECT * FROM CUSTOMER WHERE LastName LIKE 'J_cks_n'
```

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

```
SELECT * FROM CUSTOMER WHERE CustomerNumber LIKE '[10]%'
```

	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

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 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:

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

Example:

```
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

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
```

Examples:

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

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

You can also specify 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

This is very useful for large tables with thousands of records

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 short.

SQL Alias Syntax for Tables:

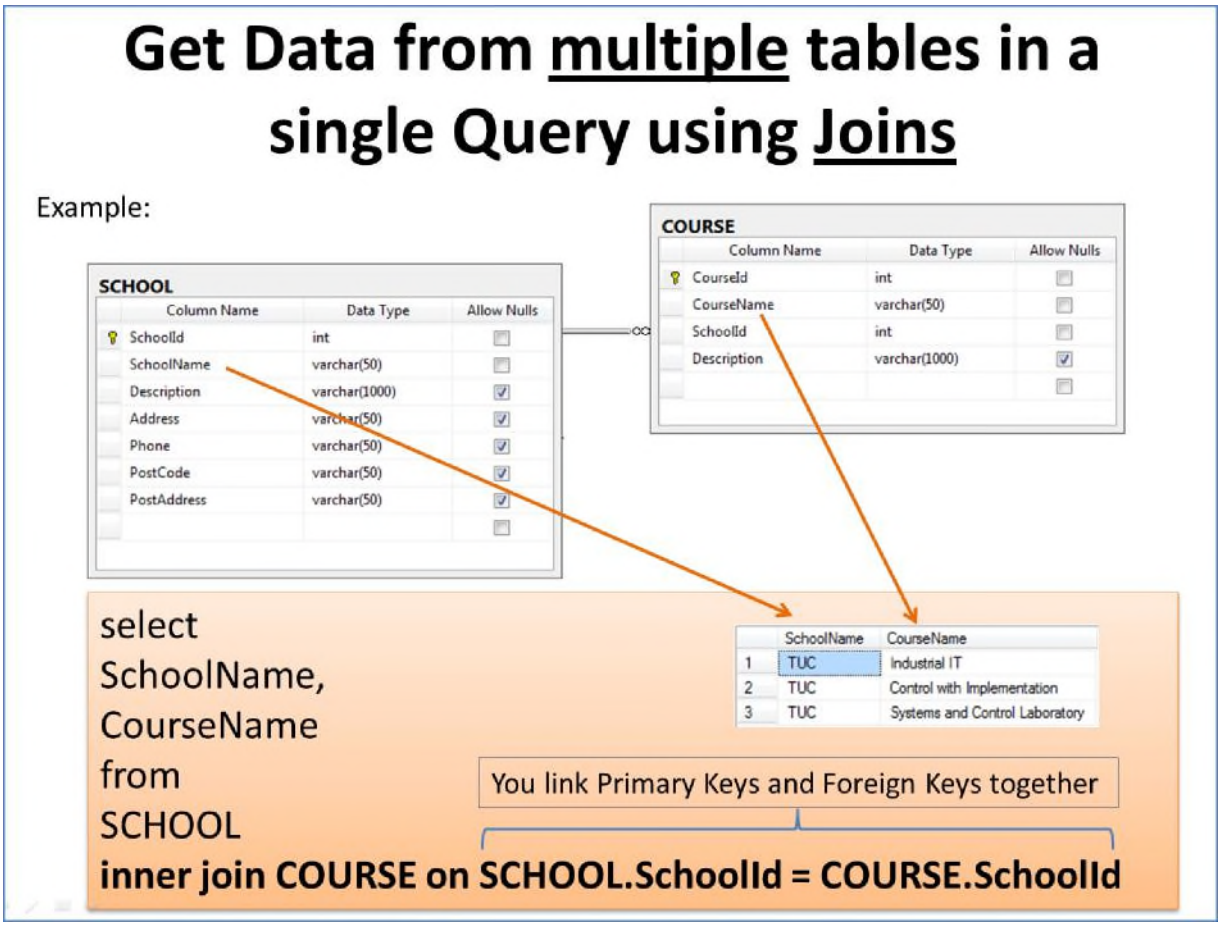
```
SELECT column_name(s)
FROM table_name
AS alias_name
```

SQL Alias Syntax for Columns:

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

Joins

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



Different SQL JOINs

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

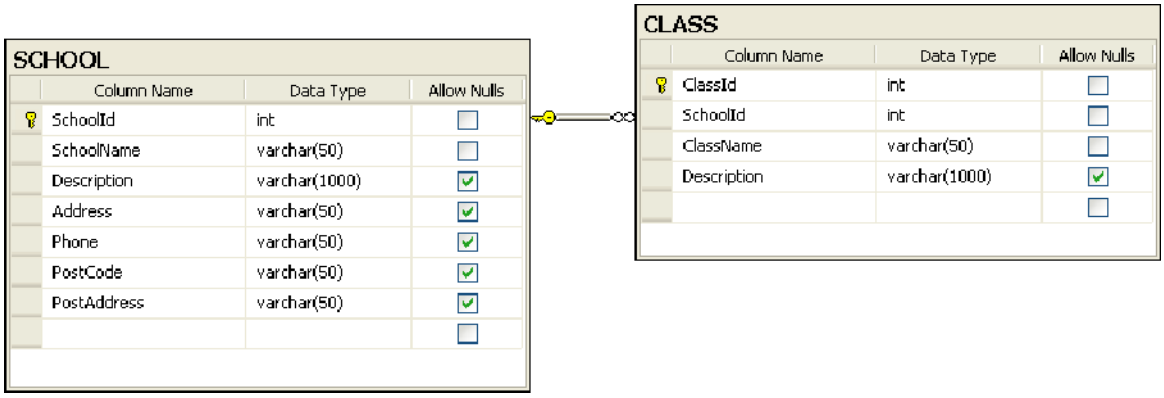
- JOIN: Return rows when there is at least one match in both tables
- LEFT JOIN: Return all rows from the left table, even if there are no matches in the right table
- RIGHT JOIN: Return all rows from the right table, even if there are no matches in the left table
- FULL JOIN: Return rows when there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

Example:

Given 2 tables:

- SCHOOL
- CLASS

The diagram is shown below:



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

CASE

The CASE statement evaluates a list of conditions and returns one of multiple possible result expressions.

Example:

We have a "GRADE" table that contains the grades for each student in different courses:

```
select GradeId, StudentId, CourseId, Grade from GRADE
```

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

In the "GRADE" table is the grades stored as numbers, but since the students get grades with the letters A..F (A=5, B=4, C=3, D=2, E=1, F=0), we want to convert the values in the table into letters using a CASE statement:

```
select
GradeId,
StudentId,
CourseId,
case Grade
      when 5 then 'A'
      when 4 then 'B'
      when 3 then 'C'
      when 2 then 'D'
      when 1 then 'E'
      when 0 then 'F'
      else '-'
end as Grade
from
```

```
GRADE
```

	GradeId	StudentId	CourseId	Grade
1	1	1	1	B
2	2	2	1	A
3	3	3	3	F
4	4	4	3	C
5	5	1	3	A

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 of the most used built-in functions and in addition we will create our own function.

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.

String Functions

Here are some useful functions used to manipulate with 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.

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.

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.

AVG()

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

Syntax:

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

Example:

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

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
```

We 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

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
```

Example:

We 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
```

We 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

The HAVING Clause

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

Syntax:

```
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
```

We 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

While the following query:

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

	CourseId	(No column name)
1	1	4,5