# 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
```

| | 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.

Kvrity Solutions

# 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 |

For Kritya Solutions

# 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 |

# 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,...)
```

Kitty Solutions

## 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
```

# 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

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.

Kritya Solutions

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

for Kriya Solutions

# 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.

## 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

# Different SQL JOINs

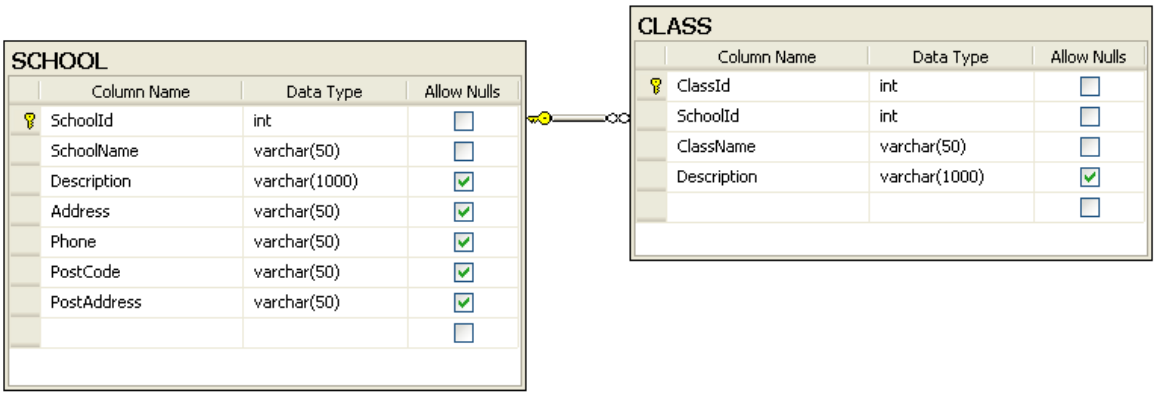
Before we continue with examples, we will list the types of JOIN you can use, and the differences between them.

- 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:




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 |

For Kritya