# Practical 3 Dr. Haider A. Haddad

 **Class3 CS &**

**SQL DML and DDL**

SQL can be divided into two parts: The Data Manipulation Language (DML) and the Data Definition Language (DDL).

The DDL part of SQL permits database tables to be created or deleted. It also defines indexes (keys), specifies links between tables, and imposes constraints between tables.

The most important DDL statements in SQL are:

* **CREATE DATABASE** - creates a new database
* **ALTER DATABASE** - modifies a database
* **CREATE TABLE** - creates a new table
* **ALTER TABLE** - modifies a table
* **DROP TABLE** - deletes a table
* **CREATE INDEX** - creates an index (search key)
* **DROP INDEX** - deletes an index

The query and update commands form the DML part of SQL:

* **SELECT** - extracts data from a database
* **UPDATE** - updates data in a database
* **DELETE** - deletes data from a database
* **INSERT INTO** - inserts new data into a database

## **1- The SQL SELECT Statement**

## **The SQL SELECT \***

**SELECT \* FROM Persons**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

**SELECT LastName,FirstName FROM Persons**

The result-set will look like this:

|  |  |
| --- | --- |
| **LastName** | **FirstName** |
| Hansen | Ola |
| Svendson | Tove |
| Pettersen | Kari |

**The SQL SELECT DISTINCT Statement**

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 "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select only the distinct values from the column named "City" from the table above.

We use the following SELECT statement:

**SELECT DISTINCT City FROM Persons**

The result-set will look like this:

|  |
| --- |
| **City** |
| Sandnes |
| Stavanger |

## **Operators Allowed in the WHERE Clause**

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 | To specify multiple possible values for a column |

**Combining AND & OR**

We use the following SELECT statement:

**SELECT \* FROM Persons WHERE
LastName='Svendson'
AND (FirstName='Tove' OR FirstName='Ola')**

*The result-set will look like this:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

## **2- SQL INSERT INTO Example**

We have the following "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to insert a new row in the "Persons" table.

We use the following SQL statement:

**INSERT INTO Persons
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')**

The "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |

## **Insert Data Only in Specified Columns**

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

The following SQL statement will add a new row, but only add data in the "P\_Id", "LastName" and the "FirstName" columns:

INSERT INTO Persons (P\_Id, LastName, FirstName)
VALUES (5, 'Tjessem', 'Jakob')

The "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Jassem | Jakob |  |  |

## **3-The UPDATE Statement**

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

UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'

The "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

## SQL UPDATE Warning

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

**UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'**

The "Persons" table would have looked like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Nissestien 67 | Sandnes |
| 2 | Svendson | Tove | Nissestien 67 | Sandnes |
| 3 | Pettersen | Kari | Nissestien 67 | Sandnes |
| 4 | Nilsen | Johan | Nissestien 67 | Sandnes |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

## **4-The DELETE Statement**

## SQL DELETE Example

The "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

**DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob'**

The "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |

## **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 Persons

## **4- The TOP Clause**

The "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Tom | Vingvn 23 | Stavanger |

Now we want to select only the two first records in the table above.

We use the following SELECT statement:

SELECT TOP 2 \* FROM Persons

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

## SQL TOP PERCENT Example

The "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Tom | Vingvn 23 | Stavanger |

Now we want to select only 50% of the records in the table above.

We use the following SELECT statement:

**SELECT TOP 50 PERCENT \* FROM Persons**

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

## **5- The LIKE Operator**

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

### SQL LIKE Syntax

SELECT column\_name(s)
FROM table\_name
WHERE column\_name LIKE pattern

## LIKE Operator Example

The "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select the persons living in a city that starts with "s" from the table above.

We use the following SELECT statement:

SELECT \* FROM Persons
WHERE City LIKE 's%'

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

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Next, we want to select the persons living in a city that ends with an "s" from the "Persons" table.

We use the following SELECT statement:

SELECT \* FROM Persons
WHERE City LIKE '%s'

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

Next, we want to select the persons living in a city that contains the pattern "tav" from the "Persons" table.

We use the following SELECT statement:

SELECT \* FROM Persons
WHERE City LIKE '%tav%'

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

It is also possible to select the persons living in a city that does NOT contain the pattern "tav" from the "Persons" table, by using the NOT keyword.

We use the following SELECT statement:

SELECT \* FROM Persons
WHERE City NOT LIKE '%tav%'

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

SQL wildcards can be used when searching for data in a database.

**6- SQL Wildcards**

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

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 |

SQL Wildcard Examples

We have the following "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Using the % Wildcard

Now we want to select the persons living in a city that starts with "sa" from the "Persons" table.

We use the following SELECT statement:

SELECT \* FROM Persons
WHERE City LIKE 'sa%'

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

Next, we want to select the persons living in a city that contains the pattern "nes" from the "Persons" table.

We use the following SELECT statement:

SELECT \* FROM Persons
WHERE City LIKE '%nes%'

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

Using the \_ Wildcard

Now we want to select the persons with a first name that starts with any character, followed by "la" from the "Persons" table.

We use the following SELECT statement:

**SELECT \* FROM Persons
WHERE FirstName LIKE '\_la'**

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |

Next, we want to select the persons with a last name that starts with "S", followed by any character, followed by "end", followed by any character, followed by "on" from the "Persons" table.

We use the following SELECT statement:

**SELECT \* FROM Persons
WHERE LastName LIKE 'S\_end\_on'**

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |

Using the [charlist] Wildcard

Now we want to select the persons with a last name that starts with "b" or "s" or "p" from the "Persons" table.

We use the following SELECT statement:

**SELECT \* FROM Persons
WHERE LastName LIKE '[bsp]%'**

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Next, we want to select the persons with a last name that do not start with "b" or "s" or "p" from the "Persons" table.

We use the following SELECT statement:

**SELECT \* FROM Persons
WHERE LastName LIKE '[!bsp]%'**

The result-set will look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |

**7- The COUNT() function returns the number of rows that matches a specified criteria.**

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to count the number of orders from "Customer Nilsen".

We use the following SQL statement:

**SELECT COUNT(Customer) AS CustomerNilsen FROM Orders
WHERE Customer='Nilsen'**

The result of the SQL statement above will be 2, because the customer Nilsen has made 2 orders in total:

|  |
| --- |
| **CustomerNilsen** |
| 2 |

## SQL COUNT(\*) Example

If we omit the WHERE clause, like this:

**SELECT COUNT(\*) AS NumberOfOrders FROM Orders**

The result-set will look like this:

|  |
| --- |
| **NumberOfOrders** |
| 6 |

which is the total number of rows in the table.

## SQL COUNT(DISTINCT column\_name) Example

Now we want to count the number of unique customers in the "Orders" table.

We use the following SQL statement:

**SELECT COUNT(DISTINCT Customer) AS NumberOfCustomers FROM Orders**

The result-set will look like this:

|  |
| --- |
| **NumberOfCustomers** |
| 3 |

which is the number of unique customers (Hansen, Nilsen, and Jensen) in the "Orders" table.

## 8-The FIRST() and Last() Function

The FIRST() function returns the first value of the selected column.

## SQL FIRST() Example

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find the first value of the "OrderPrice" column.

We use the following SQL statement:

**SELECT FIRST(OrderPrice) AS FirstOrderPrice FROM Orders**

|  |
| --- |
| **FirstOrderPrice** |
| 1000 |

## The LAST() Function

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find the last value of the "OrderPrice" column.

We use the following SQL statement:

**SELECT LAST(OrderPrice) AS LastOrderPrice FROM Orders**

|  |
| --- |
| **LastOrderPrice** |
| 100 |