EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

CSE301 – DATABASE

SQL - Structured Query Language

INTRODUCTION TO SQL

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases

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.

SQL CREATE DATABASE STATEMENT

The CREATE DATABASE statement is used to create a database.

SQL CREATE DATABASE Syntax: *CREATE DATABASE database_name*

CREATE DATABASE Example

Now we want to create a database called "my_db". We use the following CREATE DATABASE statement:

CREATE DATABASE my_db

The DROP DATABASE Statement
DROP DATABASE database_name

ALTER DATABASE - modifies a database

ALTER DATABASE mydb READ ONLY = 1;

USE DATABASE: - change the current database;

Use mydb;

SQL CREATE TABLE STATEMENT

In relational database systems (DBS) data are represented using tables(relations).

The CREATE TABLE statement is used to create a table in a database.

SQL CREATE TABLE Syntax

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

SQL CREATE TABLE STATEMENT

CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: P_Id, LastName, FirstName, Address, and City.

```
CREATE TABLE Persons (
    P_Id int,
    LastName varchar(255), FirstName varchar(255),
    Address varchar(255), City varchar(255)
)
```

P_Id	LastName	FirstName	Address	City

SHOW TABLES; Show all table in database

SQL CREATE TABLE STATEMENT

CREATE TABLE Example

```
CREATE TABLE IF NOT EXISTS employees (
    employee_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    birth_date DATE,
    hire_date DATE NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    phone_number VARCHAR(15),
    job_id INT,
    salary DECIMAL(10, 2) DEFAULT 0.00,
    department_id INT,
    CONSTRAINT fk_department
        FOREIGN KEY (department_id)
        REFERENCES departments(department_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    CONSTRAINT chk_salary CHECK (salary >= 0)
)
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

MYSQL DATA TYPES

In MySQL there are three main types : text, number, and Date/Time types.

Text types:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. **Note:** The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

MYSQL DATA TYPES

Number types:

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

MYSQL DATA TYPES

Date types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format.

SQL TABLE(DROP/ALTER) INTO STATEMENT

Drop table TableName; Delete a table.

Drop table Employee;

Alter table: Modify a table

ALTER TABLE table_name [alter_specification] [, ...] [table_options];

SQL TABLE(DROP/ALTER) INTO STATEMENT

```
-- Add a new column 'middle_name' after 'first_name'
ALTER TABLE employees
ADD COLUMN middle_name VARCHAR(50) AFTER first_name;

-- Drop the column 'phone_number'
ALTER TABLE employees
DROP COLUMN phone_number;

-- Modify the 'email' column to allow NULL values
ALTER TABLE employees
MODIFY COLUMN email VARCHAR(100);

-- Change the 'hire_date' column to 'joining_date' and update its definition
ALTER TABLE employees
CHANGE COLUMN hire_date joining_date DATE NOT NULL;

-- Add a unique constraint on 'email' column
ALTER TABLE employees
ADD CONSTRAINT uq_email UNIQUE (email);
```

```
-- Add a foreign key constraint on 'department_id'
ALTER TABLE employees
ADD CONSTRAINT fk_department
FOREIGN KEY (department_id)
REFERENCES departments(department_id)
ON DELETE SET NULL
ON UPDATE CASCADE;

-- Drop the unique constraint on 'email' column
ALTER TABLE employees
DROP INDEX uq_email;

-- Drop the foreign key constraint
ALTER TABLE employees
DROP FOREIGN KEY fk_department;

-- Set auto-increment value to 1000
ALTER TABLE staff
AUTO_INCREMENT=1000;

-- Change the default character set and collation
ALTER TABLE staff
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

SQL INSERT INTO STATEMENT

The INSERT INTO statement is used to insert a new row in a table.

SQL INSERT INTO Syntax

INSERT INTO table_name

VALUES (value1, value2, value3,...)

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,...)

SQL INSERT INTO STATEMENT

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

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

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

SQL INSERT INTO STATEMENT

SQL INSERT INTO Example

Insert Data Only in Specified Columns

INSERT INTO Persons (P_Id, LastName, FirstName)

VALUES (5, 'Tjessem', 'Jakob')

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			

SQL CONSTRAINTS

Constraints are used to limit the type of data that can go into a table. 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).

We will focus on the following constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

SQL NOT NULL CONSTRAINT

By default, a table column can hold NULL values.
The NOT NULL constraint enforces a column to NOT accept NULL values.

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

Example:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE CONSTRAINT

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.

Example:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
UNIQUE (P_Id)
)
```

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
)
```

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

Example: *ALTER TABLE Persons ADD UNIQUE (P_Id)*

ALTER TABLE Persons ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)

To DROP a UNIQUE Constraint

ALTER TABLE Persons DROP INDEX uc_PersonID

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot contain NULL values.
- Each table should have a primary key, and each table can have only ONE primary key.

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (P_Id)
)
```

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

SQL PRIMARY KEY Constraint on ALTER TABLE

To create a PRIMARY KEY constraint on the "P_Id" column when the table is already created, use the following SQL

ALTER TABLE Persons ADD PRIMARY KEY (P_Id)

ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)

To DROP a PRIMARY KEY Constraint

ALTER TABLE Persons DROP PRIMARY KEY

ALTER TABLE Persons
DROP CONSTRAINT pk_PersonID

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another 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

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
)
```

SQL FOREIGN KEY Constraint on ALTER TABLE

To create a FOREIGN KEY constraint on the "P_Id" column when the "Orders" table is already created, use the following SQL

ALTER TABLE Orders ADD FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)

ALTER TABLE Orders ADD CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)

To DROP a FOREIGN KEY Constraint

ALTER TABLE Orders DROP FOREIGN KEY fk_PerOrders

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column

SQL CHECK Constraint on CREATE TABLE

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,  FirstName varchar(255),
Address varchar(255),
City varchar(255),
CHECK (P_Id>0)
)
```

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
)
```

SQL CHECK Constraint

SQL CHECK Constraint on ALTER TABLE

ALTER TABLE Persons ADD CHECK (P_Id>0)

ALTER TABLE Persons ADD CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')

To DROP a CHECK Constraint

ALTER TABLE Persons DROP CONSTRAINT chk_Person

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

SQL DEFAULT Constraint on CREATE TABLE

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
)
```

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
OrderDate date DEFAULT GETDATE()
)
```

SQL DEFAULT Constraint

SQL DEFAULT Constraint on ALTER TABLE

ALTER TABLE Persons ALTER City SET DEFAULT 'SANDNES'

To DROP a DEFAULT Constraint

ALTER TABLE Persons ALTER City DROP DEFAULT

SQL SYNTAX

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.
Below is an example of a table called "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

SQL SYNTAX

SQL DML and DDL

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

The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose 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

SQL SELECT STATEMENT

The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set.
Most of the actions you need to perform on a database are done with SQL statements.
The following SQL statement will select all the records in the "Persons" table:

SQL SELECT Syntax

SELECT column_name(s) FROM table_name
*SELECT * FROM table_name*

SQL SELECT STATEMENT

SQL SELECT Example

*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

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

SQL SELECT STATEMENT

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 DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax.

SELECT DISTINCT column_name(s) FROM table_name

SELECT DISTINCT Example

SELECT DISTINCT City 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

City
Sandnes
Stavanger

SQL SELECT STATEMENT

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 DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax.

SELECT DISTINCT column_name(s) FROM table_name

SELECT DISTINCT Example

SELECT DISTINCT City 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

City
Sandnes
Stavanger

SQL SELECT STATEMENT

SQL WHERE CLAUSE

The WHERE clause is used to filter records.

SQL WHERE Syntax

SELECT column_name(s)
FROM table_name
WHERE column_name operator value

WHERE Clause Example

*SELECT * FROM Persons WHERE City='Sandnes'*

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

SQL SELECT STATEMENT

Operators Allowed in the WHERE Clause

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	

SQL SELECT STATEMENT

SQL 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.

AND Operator Example

*SELECT * FROM Persons WHERE FirstName='Tove' AND LastName='Svendson'*

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

OR Operator Example

*SELECT * FROM Persons WHERE FirstName='Tove' OR FirstName='Ola'*

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

SQL SELECT STATEMENT

SQL ORDER BY KEYWORD

The ORDER BY keyword is used to sort the result-set.

SQL ORDER BY Syntax

SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC

ORDER BY Example

*SELECT * FROM Persons ORDER BY LastName*

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

SQL SELECT STATEMENT

ORDER BY DESC Example

*SELECT * FROM Persons ORDER BY LastName DESC*

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

SQL UPDATE STATEMENT

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

SQL UPDATE Syntax

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

SQL UPDATE Example

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

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		

Address	City
Timoteivn 10	Sandnes
Borgvn 23	Sandnes
Storgt 20	Stavanger
Bakken 2	Stavanger
Nissestien 67	Sandnes

SQL DELETE STATEMENT

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

SQL DELETE Syntax

DELETE FROM table_name
WHERE some_column=some_value

SQL DELETE Example

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

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

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

SQL LIMIT CLAUSE

SQL LIMIT Syntax:

SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;

SQL LIMIT Example

SELECT *
FROM Persons
LIMIT 2

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

P_Id	LastName	FirstName		Address	City
1	Hansen	Ola		Timoteivn 10	Sandnes
2	Svendson	Tove		Borgvn 23	Sandnes

SQL LIKE OPERATOR

SQL WILDCARDS

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

SQL wildcards must be used with the SQL LIKE operator.

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character

SQL LIKE OPERATOR

SQL WILDCARDS

SQL Wildcard Examples

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

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

*SELECT * FROM Persons*
WHERE LastName LIKE '%e%'

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

*SELECT * FROM Persons*
WHERE LastName LIKE 'h%'

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

SQL IN OPERATOR

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

SQL IN Syntax

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

IN Operator Example

*SELECT * FROM Persons*
WHERE LastName IN ('Hansen','Pettersen')

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

SQL BETWEEN OPERATOR

The BETWEEN operator is used in a WHERE clause to select a range of data between two values.

SQL BETWEEN Syntax

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

BETWEEN Operator Example

*SELECT * FROM Persons*
WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

SQL BETWEEN OPERATOR

To display the persons outside the range in the previous example, use NOT BETWEEN:

*SELECT **
FROM Persons
WHERE LastName NOT BETWEEN 'Hansen' AND 'Pettersen'

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

SQL ALIAS

With SQL, an alias name can be given to a table or to a column.

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

Alias Example

SELECT po.OrderID, p.LastName, p.FirstName FROM Persons AS p,
Product_Orders AS po
WHERE p.LastName='Hansen' AND p.FirstName='Ola'

SQL JOINS

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

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

The "Orders" table

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

SQL JOINS

Different SQL JOINs

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

SQL INNER JOIN

The INNER JOIN keyword return rows when there is at least one match in both tables.

SQL INNER JOIN Syntax

SELECT column_name(s)
FROM table_name1 INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name

SQL INNER JOIN Example

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons
INNER JOIN Orders ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName

SQL INNER JOIN

SQL INNER JOIN Example

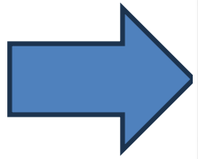
The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons
INNER JOIN Orders
ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName
```



LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

SQL LEFT JOIN

The LEFT JOIN keyword returns all rows from the left table (table_name1), even if there are no matches in the right table (table_name2).

SQL LEFT JOIN Syntax

SELECT column_name(s)
FROM table_name1 LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name

SQL LEFT JOIN Example

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons
LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName

SQL LEFT JOIN

SQL LEFT JOIN Example

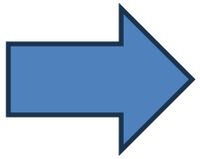
The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons
LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName
```



LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

SQL RIGHT JOIN

The RIGHT JOIN keyword Return all rows from the right table (table_name2), even if there are no matches in the left table (table_name1).

SQL RIGHT JOIN Syntax

SELECT column_name(s)
FROM table_name1 RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name

SQL RIGHT JOIN Example

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName

SQL RIGHT JOIN

SQL RIGHT JOIN Example

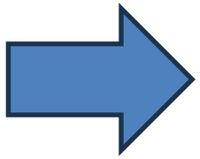
The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id  ORDER BY Persons.LastName
```



LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
		34764

SQL FULL JOIN

The FULL JOIN keyword return rows when there is a match in one of the tables. **(SQL Server/Oracle)**

SQL FULL JOIN Syntax

SELECT column_name(s)
FROM table_name1 FULL JOIN table_name2
ON table_name1.column_name=table_name2.column_name

SQL FULL JOIN Example

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons FULL JOIN Orders
ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName

MySQL

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons LEFT JOIN Orders
ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName
UNION ALL
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons RIGHT JOIN Orders
ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName

SQL FULL JOIN

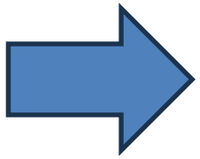
SQL FULL JOIN Example

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

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15



LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	
		34764

SQL UNION OPERATOR

The UNION operator is used to combine the result-set of two or more SELECT statements.

SQL UNION Syntax

SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2

Note: The column names in the result-set of a UNION are always equal to the column names in the first SELECT statement in the UNION.

SQL UNION OPERATOR

SQL UNION Example

"Employees_Norway":

E_ID	E_Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

"Employees_USA":

E_ID	E_Name
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

SQL UNION OPERATOR

SQL UNION Example

SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

Note: This command cannot be used to list all employees in Norway and USA. In the example above we have two employees with equal names, and only one of them will be listed. The UNION command selects only **distinct** values.

SQL UNION OPERATOR

SQL UNION ALL Example

SELECT E_Name FROM Employees_Norway
UNION ALL
SELECT E_Name FROM Employees_USA

E_Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Svendson, Stephen
Scott, Stephen

SQL SELECT INTO STATEMENT

SQL SELECT INTO Syntax

SELECT * INTO @variable [IN externaldatabase]
FROM old_tablename

Example

SELECT E_id, E_name INTO @id, @name
FROM Employees
LIMIT 1;

SELECT @id, @name

SQL CREATE INDEX STATEMENT

The **CREATE INDEX** statement is used to create indexes in tables
An index can be created in a table to find data more quickly and efficiently.
The users cannot see the indexes, they are just used to speed up searches/queries.

SQL CREATE INDEX Syntax
Creates an index on a table. **Duplicate values** are allowed:

CREATE INDEX index_name ON table_name (column_name)

SQL CREATE UNIQUE INDEX Syntax
Creates a unique index on a table. **Duplicate values are not allowed**:

CREATE UNIQUE INDEX index_name ON table_name (column_name)

SQL CREATE INDEX STATEMENT

CREATE INDEX Example

CREATE INDEX Pindex ON Persons (LastName)

CREATE INDEX Pindex ON Persons (LastName, FirstName)

SQL DROP INDEX, DROP TABLE, and DROP DATABASE

The DROP INDEX Statement

ALTER TABLE table_name DROP INDEX index_name

The DROP TABLE Statement

DROP TABLE table_name

The DROP DATABASE Statement

DROP DATABASE database_name

SQL ALTER TABLE STATEMENT

SQL ALTER TABLE Syntax

To add a column in a table, use the following syntax:

ALTER TABLE table_name ADD column_name datatype

To delete a column in a table, use the following syntax :

ALTER TABLE table_name DROP COLUMN column_name

To change the data type of a column in a table, use the following syntax:

ALTER TABLE table_name MODIFY COLUMN column_name datatype

SQL ALTER TABLE STATEMENT

SQL ALTER TABLE Example

ALTER TABLE Persons ADD DateOfBirth date

Change Data Type Example

ALTER TABLE Persons ALTER COLUMN DateOfBirth year

DROP COLUMN Example

ALTER TABLE Persons DROP COLUMN DateOfBirth

SQL AUTO INCREMENT FIELD

The following SQL statement defines the "P_Id" column to be **an auto-increment primary key** field in the "Persons" table:

CREATE TABLE Persons (
P_Id int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT NULL, FirstName varchar(255),
Address varchar(255),
City varchar(255), PRIMARY KEY (P_Id)
)

The starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

ALTER TABLE Persons AUTO_INCREMENT=100 (Start)

SQL VIEWS

A view is a virtual table.

SQL CREATE VIEW Statement

A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

SQL CREATE VIEW Syntax

CREATE VIEW view_name

AS

SELECT column_name(s) FROM table_name

WHERE condition

SQL VIEWS

SQL CREATE VIEW Examples

CREATE VIEW [Current Product List]
AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No

We can query the view:
*SELECT * FROM [Current Product List]*

SQL VIEWS

SQL CREATE VIEW Examples

CREATE VIEW [Products Above Average Price]
AS
SELECT ProductName,UnitPrice
FROM Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)

We can query the view:
*SELECT * FROM [Products Above Average Price]*

SQL VIEWS

SQL Updating a View

You can update a view by using the following syntax:

CREATE OR REPLACE VIEW view_name

AS

SELECT column_name(s)
FROM table_name WHERE condition

Example

CREATE VIEW [Current Product List]
AS
SELECT ProductID,ProductName,Category FROM Products
WHERE Discontinued=No

SQL VIEWS

SQL Dropping a View

SQL DROP VIEW Syntax

DROP VIEW view_name

SQL DATE FUNCTIONS

Function	Description
NOW()	Returns the current date and time
CURDATE()	Returns the current date
CURTIME()	Returns the current time
DATE()	Extracts the date part of a date or date/time expression
EXTRACT()	Returns a single part of a date/time
DATE_ADD()	Adds a specified time interval to a date
DATE_SUB()	Subtracts a specified time interval from a date
DATEDIFF()	Returns the number of days between two dates
DATE_FORMAT()	Displays date/time data in different formats

SQL NULL VALUES

NULL values represent missing unknown data. By default, a table column can hold NULL values

SQL IS NULL

SELECT LastName,FirstName,Address
FROM Persons
WHERE Address IS NULL

SQL IS NOT NULL

SELECT LastName,FirstName,Address
FROM Persons
WHERE Address IS NOT NULL

SQL FUNCTIONS

SQL Aggregate Functions

- **AVG**() - Returns the average value
- **COUNT**() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- **MAX**() - Returns the largest value
- **MIN**() - Returns the smallest value
- **SUM**() - Returns the sum

SQL Scalar functions

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

SQL FUNCTIONS

SQL AVG() Function

SELECT AVG(column_name) FROM table_name

SQL AVG() Example

SELECT AVG(OrderPrice) AS OrderAverage
FROM Orders

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

OrderAverage
950

SQL FUNCTIONS

SQL COUNT() Function

SELECT COUNT(column_name) FROM table_name

SQL COUNT(column_name) Example

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

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

CustomerNilsen
2

SQL FUNCTIONS

SQL COUNT() Function

SELECT COUNT(column_name) FROM table_name

SQL COUNT(DISTINCT column_name) Example

SELECT COUNT(DISTINCT Customer) AS NumberOfCustomers
FROM Orders

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

NumberOfCustomers	
3	

SQL FUNCTIONS

SQL FIRST() Function

SELECT FIRST(column_name) FROM table_name

SQL FIRST() Example

SELECT FIRST(OrderPrice) AS FirstOrderPrice
FROM Orders

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

FirstOrderPrice
1000

SQL FUNCTIONS

SQL LAST() Function

SELECT LAST(column_name) FROM table_name

SQL LAST() Example

SELECT LAST(OrderPrice) AS LastOrderPrice
FROM Orders

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

LastOrderPrice
100

SQL FUNCTIONS

SQL MAX(), MIN() Function

SELECT MAX(column_name) FROM table_name
SELECT MIN(column_name) FROM table_name

SQL MAX(), MIN() Example

SELECT MAX(OrderPrice) AS LargestOrderPrice
FROM Orders

SELECT MIN(OrderPrice) AS SmallestOrderPrice
FROM Orders

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

LargestOrderPrice
2000

SmallestOrderPrice
100

SQL FUNCTIONS

SQL SUM Function

SELECT SUM(column_name) FROM table_name

SQL SUM() Example

SELECT SUM(OrderPrice) AS OrderTotal
FROM Orders

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

OrderTotal
5700

SQL FUNCTIONS

SQL MID() Function

SELECT MID(column_name,start[,length]) FROM table_name

Parameter	Description
column_name	Required. The field to extract characters from
Start	Required. Specifies the starting position (starts at 1)
Length	Optional. The number of characters to return. If omitted, the MID() function returns the rest of the text

SQL MID() Example

SELECT MID(City,1,4) as SmallCity
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

SmallCity
Sand
Sand
Stav

SQL FUNCTIONS

SQL LEN() Function

SELECT LEN (column_name,start[,length]) FROM table_name

SQL LEN () Example

SELECT LEN(Address) as LengthOfAddress
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

LengthOfAddress
12
9
9

SQL FUNCTIONS

SQL ROUND() Function

SELECT ROUND(column_name,decimals) FROM table_name

SQL ROUND() Example

SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice
FROM Products

Prod_Id	ProductName	Unit	UnitPrice
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	Gorgonzola	1000 g	15.67

ProductName	UnitPrice
Jarlsberg	10
Mascarpone	33
Gorgonzola	16

SQL FUNCTIONS

SQL NOW() Function

SELECT NOW() FROM table_name

SQL NOW() Example

SELECT ProductName, UnitPrice, Now() as PerDate
FROM Products

Prod_Id	ProductName		Unit	UnitPrice
1	Jarlsberg		1000 g	10.45
2	Mascarpone		1000 g	32.56
3	Gorgonzola		1000 g	15.67

ProductName	UnitPrice		PerDate
Jarlsberg	10.45		10/7/2008 11:25:02 AM
Mascarpone	32.56		10/7/2008 11:25:02 AM
Gorgonzola	15.67		10/7/2008 11:25:02 AM

SQL GROUP BY Statement

Aggregate functions often need an added GROUP BY statement

SQL GROUP BY Syntax

SELECT column_name, aggregate_function(column_name)

FROM table_name

WHERE column_name operator value

GROUP BY column_name

SQL GROUP BY Example

SELECT Customer, SUM(OrderPrice)
FROM Orders
GROUP BY Customer

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

Customer	SUM(OrderPrice)
Hansen	2000
Nilsen	1700
Jensen	2000

SQL GROUP BY Statement

GROUP BY More Than One Column

Example

SELECT Customer,OrderDate,SUM(OrderPrice)
FROM Orders
GROUP BY Customer,OrderDate

SQL HAVING CLAUSE

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

SQL HAVING 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

SQL HAVING CLAUSE

SQL HAVING Example

SELECT Customer,SUM(OrderPrice)
FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000

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

Customer	SUM(OrderPrice)
Nilsen	1700

SUB QUERY

Sub query syntax:

```
SELECT
   LASTNAME, FIRSTNAME
FROM    EMPLOYEES
WHERE    OFFICECODE IN (SELECT   OFFICECODE
                        FROM     OFFICES
                        WHERE COUNTRY = 'USA');
```

Note: **IN, NOT IN, EXISTS, NOT EXISTS**

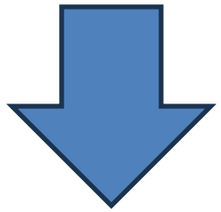
SUB QUERY

Sub query example:

```
SELECT *
FROM PERSONS
WHERE P_ID NOT IN (SELECT P_ID
                   FROM orders)
```

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	NULL	NULL

O_id	OrderNo	P_id
1	77895	3
2	44678	3
3	22456	2
4	22562	1



P_Id	LastName	FirstName	Address	City
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	NULL	NULL

SUB QUERY

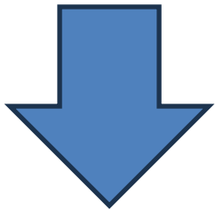
Sub query example:

Get all people who don't have any orders.

```
SELECT *
FROM PERSONS
WHERE P_ID NOT IN (SELECT P_ID
                   FROM orders)
```

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	NULL	NULL

O_id	OrderNo	P_id
1	77895	3
2	44678	3
3	22456	2
4	22562	1



P_Id	LastName	FirstName	Address	City
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	NULL	NULL

SUB QUERY

Sub query example:

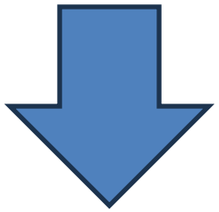
Get all people who have any orders.

```
SELECT *
FROM PERSONS as p
WHERE exists (SELECT P_ID
                FROM orders as o
               WHERE o.P_id = p.P_id);
```

```
SELECT *
FROM PERSONS as p
WHERE  p_id in (SELECT P_ID
                  FROM orders );
```

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	NULL	NULL

O_id	OrderNo	P_id
1	77895	3
2	44678	3
3	22456	2
4	22562	1



P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

STORE PROCEDURE

A stored procedure is a set of SQL statements that can be stored and executed on the database server.
Stored procedures can help improve the performance, security, and maintainability of the database applications.

Create store procedure

CREATE PROCEDURE procedure_name ([parameter_list])
BEGIN
    procedure_body
END

Drop store procedure
Drop store Procedure_name

STORE PROCEDURE

Store procedure Example

```
delimiter //
create procedure GetPersonsInOrders()
begin
      SELECT *
      FROM PERSONS as p
      WHERE  p_id in (SELECT P_ID
                  FROM orders);
End
//

call GetPersonsInOrders();

drop procedure GetPersonsInOrders;
```

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

FUNCTION STATEMENT

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

The syntax the CREATE FUNCTION statement

CREATE FUNCTION function_Name(input_arguments)
statements
RETURNS output_parameter

FUNCTION STATEMENT

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

The syntax the CREATE FUNCTION statement

CREATE FUNCTION function_Name(input_arguments)
RETURNS output_parameter
DETERMINISTIC
statements

Drop function Function_Name;

FUNCTION STATEMENT

FUNCTION statement example:

Create function

```
delimiter //
create function CountOrders()
returns int
DETERMINISTIC
begin
    declare totalOrder int;
    select count(*) into totalOrder
    from orders;
    return totalOrder;
end//
```

call function

```
select CountOrders();
```

	CountOrders()
▶	4

Drop function

```
drop function CountOrders;
```

User-Defined Variables

User-defined variables are session-specific and are defined and used within a session.

Example:

SET *@myVariable = value;*

SELECT @myVariable = value;

set @c = 10;

Set @c = (select count(*) as total from product);

select @c;

VARIABLES

System Variables .

Most of them can be changed dynamically at runtime using the SET statement, which enables you to modify operation of the server without having to stop and restart it. Some variables are read-only, and their values are determined by the system environment

Global Variables:

SET GLOBAL variable_name = value;

Example:

#Set the global max_connections variable

SET GLOBAL max_connections = 200;

#Verify the change

SHOW VARIABLES LIKE 'max_connections';

```
SHOW VARIABLES ;
```

Variable_name	Value
activate_all_roles_on_login	OFF
admin_address	
admin_port	33062
admin_ssl_ca	
admin_ssl_capath	
admin_ssl_cert	
admin_ssl_cipher	
admin_ssl_crl	
admin_ssl_crlpath	
admin_ssl_key	
admin_tls_ciphersuites	
admin_tls_version	TLSv1.2,TLSv1.3
authentication_policy	*,,
auto_generate_certs	ON
auto_increment_increment	1
auto_increment_offset	1
autocommit	ON
automatic_sp_privileges	ON
avoid_temporal_upgrade	OFF
back_log	151
basedir	C:\Program

VARIABLES

System Variables

Session Variables:

SET SESSION variable_name = value;

Example:

SET SESSION sql_mode = 'STRICT_TRANS_TABLES';

SHOW VARIABLES LIKE 'sql_mode’;

VARIABLES

Local Variables

Local variables are used within stored routines (procedures and functions) and are defined using the

DECLARE variable_name datatype [DEFAULT value];

Example:

DECLARE myLocalVar INT DEFAULT 0;

VARIABLES

```
DELIMITER //
CREATE PROCEDURE GetProductDetails(
    IN in_product_id INT,
    OUT out_product_name VARCHAR(100),
    OUT out_price DECIMAL(10, 2),
    OUT out_quantity INT
)
BEGIN
    DECLARE temp_product_name VARCHAR(100);
    DECLARE temp_price DECIMAL(10, 2);
    DECLARE temp_quantity INT;
    -- Select the product details into local variables
    SELECT product_name, price, quantity
    INTO temp_product_name, temp_price, temp_quantity
    FROM product
    WHERE product_id = in_product_id;

    -- Set the output parameters
    SET out_product_name = temp_product_name;
    SET out_price = temp_price;
    SET out_quantity = temp_quantity;
END //

DELIMITER ;
```

```
-- Declare variables to hold the output values
SET @product_name = '';
SET @price = 0.00;
SET @quantity = 0;

-- Call the stored procedure
CALL GetProductDetails(1, @product_name, @price,
@quantity);

-- Display the output values
SELECT @product_name AS product_name, @price AS price,
@quantity AS quantity;
```

IF Statement

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

Example:

```
DELIMITER //

CREATE PROCEDURE GetProductDetails(
    IN in_product_id INT,
    OUT out_product_name VARCHAR(100),
    OUT out_price DECIMAL(10, 2),
    OUT out_quantity INT
)
BEGIN
    DECLARE temp_product_name VARCHAR(100);
    DECLARE temp_price DECIMAL(10, 2);
    DECLARE temp_quantity INT;
    DECLARE product_exists INT;

    -- Check if the product exists
    SELECT COUNT(*)
    INTO product_exists
    FROM product
    WHERE product_id = in_product_id;

    -- Conditional logic to set output variables based on product existence
    IF product_exists > 0 THEN
        -- Select the product details into local variables
        SELECT product_name, price, quantity
        INTO temp_product_name, temp_price, temp_quantity
        FROM product
        WHERE product_id = in_product_id;

        -- Set the output parameters
        SET out_product_name = temp_product_name;
        SET out_price = temp_price;
        SET out_quantity = temp_quantity;
    ELSE
        -- Set output parameters to NULL if product does not exist
        SET out_product_name = NULL;
        SET out_price = NULL;
        SET out_quantity = NULL;
    END IF;
END //

DELIMITER ;
```

CASE Statement

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Example:

```
UPDATE product
SET price = CASE
    WHEN price < 20 THEN price * 1.10
    WHEN price BETWEEN 20 AND 30 THEN price * 1.05
    ELSE price * 1.02
END;
```

```
DELIMITER //

CREATE PROCEDURE GetPriceCategory(
    IN in_product_id INT,
    OUT out_price_category VARCHAR(100)
)
BEGIN
    SELECT
        CASE
            WHEN price < 20 THEN 'Budget'
            WHEN price BETWEEN 20 AND 30 THEN 'Mid-range'
            ELSE 'Premium'
        END AS price_category
    INTO out_price_category
    FROM product
    WHERE product_id = in_product_id;
END //

DELIMITER ;
```

WHILE Statement

```
[begin_label:] WHILE search_condition DO
   statement_list
END WHILE [end_label:]
```

Example

```
DELIMITER //

CREATE PROCEDURE CalculateFactorial(
   IN in_number INT,
   OUT out_factorial BIGINT
)
BEGIN
   DECLARE counter INT DEFAULT 1;
   DECLARE result BIGINT DEFAULT 1;

   -- Use a WHILE loop to calculate the factorial
   WHILE counter <= in_number DO
      SET result = result * counter;
      SET counter = counter + 1;
   END WHILE;

   -- Set the output parameter to the result
   SET out_factorial = result;
END //

DELIMITER ;
```

LOOP Statement

```
[begin_label:] LOOP
   statement_list
END LOOP [end_label]
```

Example

```
DELIMITER //

CREATE PROCEDURE CalculateSum(
   IN in_number INT,
   OUT out_sum INT
)
BEGIN
   DECLARE counter INT DEFAULT 1;
   DECLARE result INT DEFAULT 0;

   my_loop: LOOP
      IF counter > in_number THEN
         LEAVE my_loop;
      END IF;

      SET result = result + counter;
      SET counter = counter + 1;
   END LOOP my_loop;

   -- Set the output parameter to the result
   SET out_sum = result;
END //

DELIMITER ;
```

REPEAT Statement

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

```
DELIMITER //

CREATE PROCEDURE CalculateSumRepeat(
    IN in_number INT,
    OUT out_sum INT
)
BEGIN
    DECLARE counter INT DEFAULT 1;
    DECLARE result INT DEFAULT 0;

    REPEAT
        SET result = result + counter;
        SET counter = counter + 1;
    UNTIL counter > in_number
    END REPEAT;

    -- Set the output parameter to the result
    SET out_sum = result;
END //

DELIMITER ;
```

RETURN, LEAVE Statement

LEAVE statement

This statement is used to exit the flow control construct that has the given label

LEAVE label

LEAVE can be used within BEGIN ... END or loop constructs (LOOP, REPEAT, WHILE).

RETURN statement

The RETURN statement terminates execution of a stored function and returns the value expr to the function caller. There must be at least one RETURN statement in a stored function.

RETURN expr

Cursors

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL.
Cursors have these properties:

- *Asensitive: The server may or may not make a copy of its result table*
- *Read only: Not updatable*
- *Nonscrollable: Can be traversed only in one direction and cannot skip rows*

Cursors

1. Cursor DECLARE Statement;

```
DECLARE cursor_name CURSOR FOR select_statement
```

2. Cursor OPEN Statement

```
OPEN cursor_name
```

3. Cursor FETCH Statement

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

4. Cursor CLOSE Statement

```
CLOSE cursor_name
```

Cursors

Example

```
DELIMITER //

CREATE PROCEDURE CalculateProductValues()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE p_id INT;
    DECLARE p_name VARCHAR(100);
    DECLARE p_price DECIMAL(10, 2);
    DECLARE p_quantity INT;
    DECLARE total DECIMAL(15, 2);

    -- Declare a cursor
    DECLARE product_cursor CURSOR FOR
    SELECT product_id, product_name, price, quantity FROM product;

    -- Declare a handler for the end of the cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Open the cursor
    OPEN product_cursor;

    -- Loop through all rows in the cursor
    read_loop: LOOP
        FETCH product_cursor INTO p_id, p_name, p_price, p_quantity;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Calculate the total value
        SET total = p_price * p_quantity;

        -- Insert the result into the product_values table
        INSERT INTO product_values (product_id, product_name, total_value)
        VALUES (p_id, p_name, total);
    END LOOP read_loop;

    -- Close the cursor
    CLOSE product_cursor;
END //

DELIMITER ;
```

TRIGGER STATEMENT

A trigger is an object identified in the database and tied to an event occurring on a certain table.
These events include: INSERT, UPDATE or DELETE a table.
Trigger is executed automatically when a change action occurs in the table.
Users check data, synchronize data, ensure relationships between tables.

Using **SIGNAL** to Abort a Transaction
Syntax:

```
CREATE TRIGGER trigger_name trigger_time trigger_event
 ON table_name
 FOR EACH ROW
 BEGIN
        Statements
 END
```

Note: Trigger_time*(Before,After),* Trigger_event*(Insert,update,delete)*

Drop trigger syntax:

DROP TRIGGER table_name.trigger_name

TRIGGER STATEMENT

TRIGGER STATEMENT EXAMPLE:

```
delimiter //
create trigger priceIncrease
before insert on OrdersDetail
FOR EACH ROW
SET
        New.orderPrice = New.orderprice + New.orderprice *0.1;
//
```

```
insert into OrdersDetail values(9,'20240101',1000,'Tom');
```

O_id	OrderDate	OrderPrice	Customer
1	2008-11-12	1000.00	Hansen
2	2008-10-23	1600.00	Nilsen
3	2008-08-02	700.00	Hansen
4	2008-09-03	300.00	Hansen
5	2008-08-30	2000.00	Jensen
6	2008-10-04	100.00	Nilsen
7	2024-01-01	100.00	Marry
9	2024-01-01	1100.00	Tom

```
drop trigger priceIncrease;
```

Note: NEW: contains new data lines, OLD: contains old data lines

TRIGGER STATEMENT

TRIGGER STATEMENT EXAMPLE:

```
delimiter //
CREATE TRIGGER before_product_insert
   BEFORE INSERT ON products
   FOR EACH ROW
BEGIN
   IF NEW.sell_price <= 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Price must be positive';
   END IF;
END;
//
```

```
INSERT into product values('P1015','Apple Macbook air', 10,10,-5,10,5);
```

Output: Price must be positive

TRIGGER STATEMENT

TRIGGER STATEMENT EXAMPLE:

```
delimiter //
CREATE TRIGGER before_product_delete
BEFORE DELETE ON products
FOR EACH ROW
BEGIN
   INSERT INTO deleted_products_log (product_id, product_name, deleted_at)
   VALUES (OLD.id, OLD.name, NOW());
END;
//
```

TRIGGER STATEMENT

TRIGGER STATEMENT EXAMPLE:

```
delimiter //
CREATE TRIGGER before_product_delete
BEFORE DELETE ON product
FOR EACH ROW
BEGIN
   IF OLD.sell_price < 10 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete products with price less than $10';
   END IF;
END;
//
```

SQL QUICK REFERENCE

SQL Statement	Syntax
AND / OR	SELECT column_name(s) FROM table_name WHERE condition AND|OR condition
ALTER TABLE	ALTER TABLE table_name ADD column_name datatype or ALTER TABLE table_name DROP COLUMN column_name
AS (alias)	SELECT column_name AS column_alias FROM table_name or SELECT column_name FROM table_name AS table_alias
BETWEEN	SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2
CREATE DATABASE	CREATE DATABASE database_name
CREATE TABLE	CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, column_name2 data_type, ...)
CREATE INDEX	CREATE INDEX index_name ON table_name (column_name) or CREATE UNIQUE INDEX index_name ON table_name (column_name)
CREATE VIEW	CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition
DELETE	DELETE FROM table_name WHERE some_column=some_value

SQL QUICK REFERENCE

DROP DATABASE	DROP DATABASE database_name
DROP INDEX	DROP INDEX table_name.index_name (SQL Server) DROP INDEX index_name ON table_name (MS Access) DROP INDEX index_name (DB2/Oracle) ALTER TABLE table_name DROP INDEX index_name (MySQL)
DROP TABLE	DROP TABLE table_name
GROUP BY	SELECT column_name, aggregate_function(column_name) FROM table_name WHERE column_name operator value GROUP BY column_name
HAVING	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
IN	SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,..)
INSERT INTO	INSERT INTO table_name VALUES (value1, value2, value3,....) *or* INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,....)
INNER JOIN	SELECT column_name(s) FROM table_name1 INNER JOIN table_name2 ON table_name1.column_name=table_name2.column_name
LEFT JOIN	SELECT column_name(s) FROM table_name1 LEFT JOIN table_name2 ON table_name1.column_name=table_name2.column_name
RIGHT JOIN	SELECT column_name(s) FROM table_name1 RIGHT JOIN table_name2 ON table_name1.column_name=table_name2.column_name
FULL JOIN	SELECT column_name(s) FROM table_name1 FULL JOIN table_name2 ON table_name1.column_name=table_name2.column_name
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern
ORDER BY	SELECT column_name(s) FROM table_name ORDER BY column_name [ASC|DESC]
SELECT	SELECT column_name(s) FROM table_name
SELECT *	SELECT * FROM table_name
SELECT DISTINCT	SELECT DISTINCT column_name(s) FROM table_name
SELECT INTO	SELECT * INTO new_table_name [IN externaldatabase] FROM old_table_name *or* SELECT column_name(s) INTO new_table_name [IN externaldatabase] FROM old_table_name
SELECT TOP	SELECT TOP number|percent column_name(s) FROM table_name
TRUNCATE TABLE	TRUNCATE TABLE table_name
UNION	SELECT column_name(s) FROM table_name1 UNION SELECT column_name(s) FROM table_name2
UNION ALL	SELECT column_name(s) FROM table_name1 UNION ALL SELECT column_name(s) FROM table_name2
UPDATE	UPDATE table_name SET column1=value, column2=value,... WHERE some_column=some_value
WHERE	SELECT column_name(s) FROM table_name WHERE column_name operator value

Q&A