Data Base (SQL)

Mehrarvand University

Computer Engineering Department

Bachelor Degree

Lecturer: S. Rasoul Mousavi

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

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

Some of The Most Important SQL Commands

- **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
- **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

- **SQL SELECT Syntax**

SELECT *column_name*,*column_name*

FROM *table_name*;

And

SELECT * FROM table_name;

SQL SELECT DISTINCT Statement

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- SQL SELECT DISTINCT Syntax

SELECT DISTINCT *column_name*,*column_name*

FROM *table_name*;

SQL WHERE Clause

- The WHERE clause is used to filter records.
- SQL WHERE Syntax

SELECT *column_name*,*column_name*

FROM *table_name*

WHERE *column_name operator value*;

Operators in The WHERE Clause

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	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

SQL AND & OR Operators

- **The AND & OR operators are used to filter records based on more than one condition.**

```
SELECT * FROM Customers
WHERE Country='Germany'
AND City='Berlin';
```

```
SELECT * FROM Customers
WHERE Country='Germany'
AND (City='Berlin' OR City='München');
```

SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.
- SQL ORDER BY Syntax

SELECT *column_name*, *column_name*

FROM *table_name*

ORDER BY *column_name* ASC|DESC, *column_name* ASC|DESC;

SQL INSERT INTO Statement

- The INSERT INTO statement is used to insert new records in a table.
- SQL INSERT INTO Syntax

INSERT INTO *table_name*
VALUES (*value1*,*value2*,*value3*,...);

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

SQL UPDATE Statement

- The UPDATE statement is used to update records in a table.
- SQL UPDATE Syntax

UPDATE *table_name*

SET *column1=value1*,*column2=value2*,...

WHERE *some_column=some_value*;

Notice the WHERE clause in the SQL UPDATE statement!
The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

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*;

Notice the WHERE clause in the SQL DELETE statement!
The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

SQL SELECT TOP Clause

- The SELECT TOP clause is used to specify the number of records to return.
- The SELECT TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.
- **Note:** Not all database systems support the SELECT TOP clause.

SQL SELECT TOP Clause (2)

- SQL Server / MS Access Syntax

SELECT TOP *number|percent column_name(s)*

FROM *table_name*;

- MySQL Syntax

SELECT *column_name(s)*

FROM *table_name*

LIMIT *number*;

- Oracle Syntax

SELECT *column_name(s)*

FROM *table_name*

WHERE ROWNUM <= *number*;

SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- SQL LIKE Syntax

SELECT *column_name(s)*

FROM *table_name*

WHERE *column_name* LIKE *pattern*;

Example:

SELECT * FROM Customers

WHERE Country LIKE '%land%';

SQL Wildcards

- A wildcard character can be used to substitute for any other character(s) in a string.
- In SQL, wildcard characters are used with the SQL LIKE operator.
- SQL wildcards are used to search for data within a table.

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for a single character
[*charlist*]	Sets and ranges of characters to match
[^*charlist*] or [!*charlist*]	Matches only a character NOT specified within the brackets

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*,...);

SELECT * FROM Customers

WHERE City IN ('Paris','London');

SQL BETWEEN Operator

- The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.
- SQL BETWEEN Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name [NOT]* BETWEEN *value1* AND *value2;*

- Example

```
SELECT * FROM Products
WHERE (Price BETWEEN 10 AND 20)
AND NOT CategoryID IN (1,2,3);
```

SQL Aliases (1)

- SQL aliases are used to give a database table, or a column in a table, a temporary name.
- Basically aliases are created to make column names more readable.
- SQL Alias Syntax for Columns

SELECT *column_name* AS *alias_name*

FROM *table_name;*

- SQL Alias Syntax for Tables

SELECT *column_name(s)*

FROM *table_name* AS *alias_name;*

SQL Aliases (2)

- In the following SQL statement we combine four columns (Address, City, PostalCode, and Country) and create an alias named "Address":
- Example

```
SELECT CustomerName, Address+', '+City+', '+PostalCode+', '+Country AS Address
FROM Customers;
```

Note: To get the SQL statement above to work in MySQL use the following:

```
SELECT CustomerName, CONCAT(Address,', ',City,', ',PostalCode,', ',Country) AS Address
FROM Customers;
```

SQL Aliases (3)

- Example

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;
```

- Aliases can be useful when:
- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

SQL Joins

- SQL joins are used to combine rows from two or more tables.
- An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.
- The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

SQL Joins

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

```
SELECT Orders.OrderID, Customers.CustomerName,
Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different SQL JOINsDifferent SQL JOINs

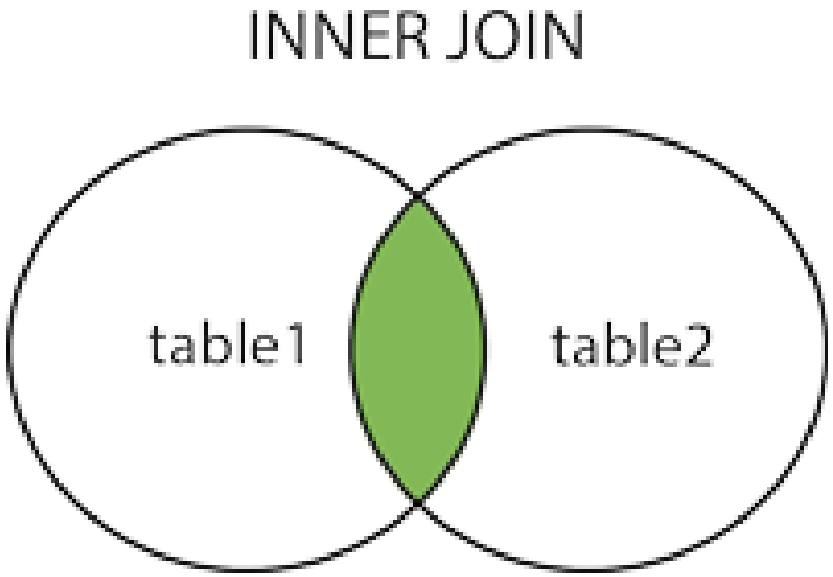
- **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables
- **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table
- **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table
- **FULL JOIN**: Return all rows when there is a match in ONE of the tables

SQL INNER JOIN Keyword

- The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.
- SQL INNER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name=table2.column_name*;

SELECT *column_name(s)*
FROM *table1*
JOIN *table2*
ON *table1.column_name=table2.column_name*;

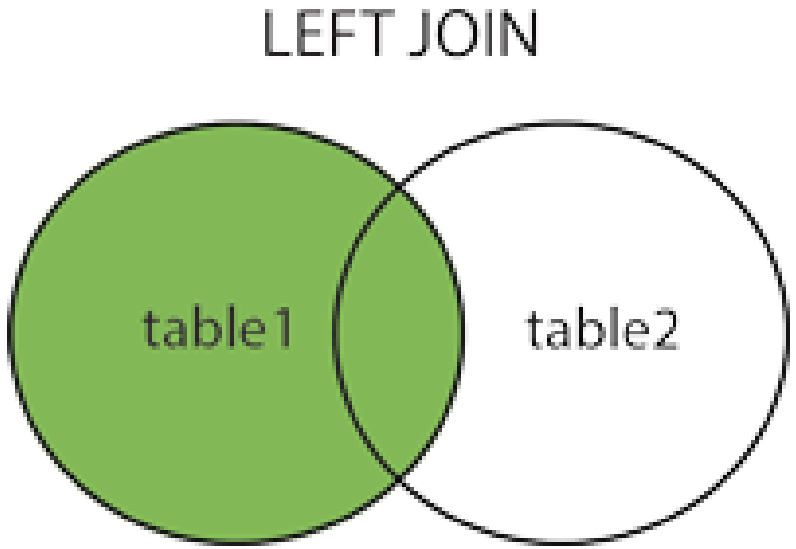


SQL LEFT JOIN Keyword

- The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.
- SQL LEFT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name=table2.column_name*;

SELECT *column_name(s)*
FROM *table1*
LEFT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;

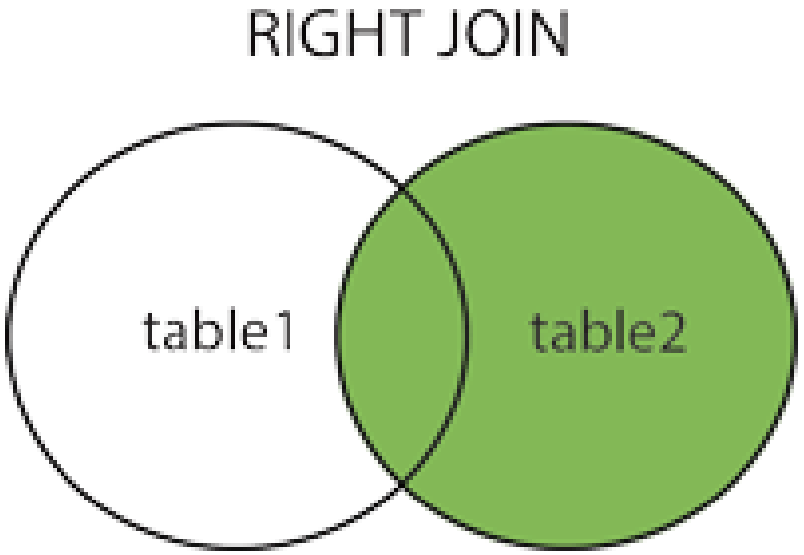


SQL RIGHT JOIN Keyword

- The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.
- SQL RIGHT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name=table2.column_name*;

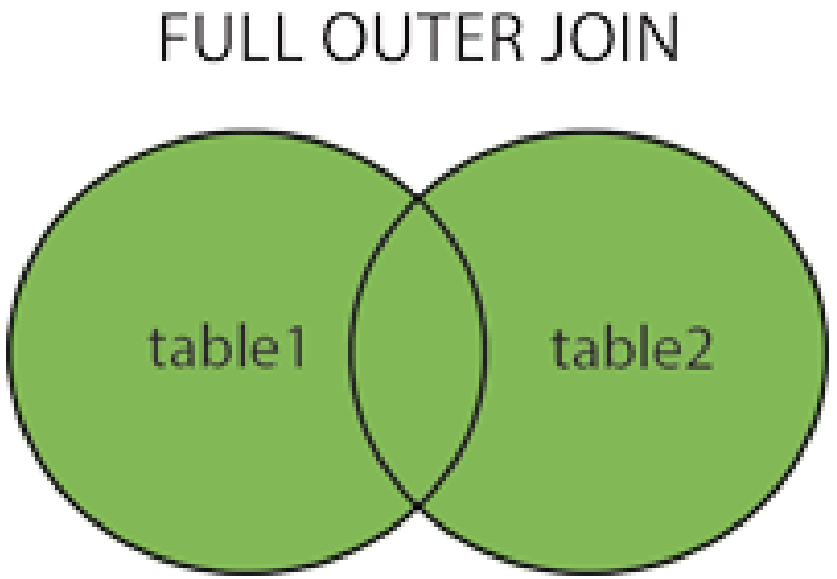
SELECT *column_name(s)*
FROM *table1*
RIGHT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;



SQL FULL OUTER JOIN Keyword

- The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).
- The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.
- SQL FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```



The SQL UNION Operator

- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*;

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

SELECT *column_name(s)* FROM *table1*
UNION ALL
SELECT *column_name(s)* FROM *table2*;

SQL UNION ALL With WHERE

- The following SQL statement uses UNION ALL to select **all** (duplicate values also) **German** cities from the "Customers" and "Suppliers" tables:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

SQL SELECT INTO Statement

- With SQL, you can copy information from one table into another.
- The SELECT INTO statement copies data from one table and inserts it into a new table.

SELECT *
INTO *newtable* [IN *externaldb*]
FROM *table1;*

SELECT *column_name(s)*
INTO *newtable* [IN *externaldb*]
FROM *table1;*

SQL SELECT INTO Examples

- SELECT *
  INTO CustomersBackup2013
  FROM Customers;
- SELECT *
  INTO CustomersBackup2013 IN 'Backup.mdb'
  FROM Customers;
- SELECT *
  INTO CustomersBackup2013
  FROM Customers
  WHERE Country='Germany';
- SELECT Customers.CustomerName, Orders.OrderID
  INTO CustomersOrderBackup2013
  FROM Customers
  LEFT JOIN Orders
  ON Customers.CustomerID=Orders.CustomerID;

SQL CREATE DATABASE Statement

CREATE DATABASE *dbname*;

SQL CREATE TABLE Statement

CREATE TABLE *table_name*

(

column_name1 data_type(*size*),

column_name2 data_type(*size*),

column_name3 data_type(*size*),

....

);

SQL Constraints

- SQL constraints are used to specify rules for the data in a table.
- If there is any violation between the constraint and the data action, the action is aborted by the constraint.
- Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

CREATE TABLE *table_name*
(
column_name1 data_type(*size*) *constraint_name*,
column_name2 data_type(*size*) *constraint_name*,
column_name3 data_type(*size*) *constraint_name*,
....
);

SQL Constraints

- In SQL, we have the following constraints:
- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK** - Ensures that the value in a column meets a specific condition
- **DEFAULT** - Specifies a default value when specified none for this column

SQL NOT NULL Constraint

- The NOT NULL constraint enforces a column to NOT accept NULL values.

```
CREATE TABLE PersonsNotNull
(
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.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.
- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint

- **SQL Server / Oracle / MS Access:**

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

SQL UNIQUE Constraint

- **MySQL:**

```
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)
)
```

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.
- Most tables should have a primary key, and each table can have only ONE primary key.

SQL PRIMARY KEY Constraint

- **MySQL:**

```
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)
)
```

- **SQL Server / Oracle / MS Access:**

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

SQL PRIMARY KEY Constraint

- To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:
- **MySQL / SQL Server / Oracle / MS Access:**

```
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)
)
```

- **Note:** In the example above there is only ONE PRIMARY KEY (pk_PersonID). However, the VALUE of the primary key is made up of TWO COLUMNS (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:
- **MySQL / SQL Server / Oracle / MS Access:**

```
ALTER TABLE Persons
ADD PRIMARY KEY (P_Id)
```

- **MySQL / SQL Server / Oracle / MS Access:**

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

SQL PRIMARY KEY Constraint on ALTER TABLE

- To DROP a PRIMARY KEY Constraint
- To drop a PRIMARY KEY constraint, use the following SQL:
- **MySQL:**

```
ALTER TABLE Persons
DROP PRIMARY KEY
```

- **SQL Server / Oracle / MS Access:**

```
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.
- **MySQL:**

```
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)
)
```

SQL FOREIGN KEY Constraint

- **SQL Server / Oracle / MS Access:**

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

SQL FOREIGN KEY Constraint on ALTER TABLE

- **MySQL / SQL Server / Oracle / MS Access:**

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

SQL CHECK Constraint

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

SQL CHECK Constraint

- **MySQL:**

```
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)
)
```

SQL CHECK Constraint

- **SQL Server / Oracle / MS Access:**

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

SQL DEFAULT Constraint

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

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

SQL DEFAULT Constraint

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

SQL CREATE INDEX Statement

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes allow the database application to find data fast; without reading the whole table.
- The users cannot see the indexes, they are just used to speed up searches/queries.
- **Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So you should only create indexes on columns (and tables) that will be frequently searched against.

SQL CREATE INDEX Statement

- 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 DROP INDEX, DROP TABLE, and DROP DATABASE

- DROP INDEX Syntax for MS Access:

DROP INDEX index_name ON table_name

- DROP INDEX Syntax for MS SQL Server:

DROP INDEX table_name.index_name

- DROP INDEX Syntax for DB2/Oracle:

DROP INDEX index_name

- DROP INDEX Syntax for MySQL:

ALTER TABLE table_name DROP INDEX index_name

SQL DROP INDEX, DROP TABLE, and DROP DATABASE

- The DROP TABLE statement is used to delete a table.

```
DROP TABLE table_name
```

- The DROP DATABASE statement is used to delete a database.

```
DROP DATABASE database_name
```

- What if we only want to delete the data inside the table, and not the table itself? Then, use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name
```

SQL ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- 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 (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name
```

SQL ALTER TABLE Statement

- To change the data type of a column in a table, use the following syntax:
- **SQL Server / MS Access:**

ALTER TABLE table_name

ALTER COLUMN column_name datatype

- **My SQL / Oracle (prior version 10G):**

ALTER TABLE table_name

MODIFY COLUMN column_name datatype

- **Oracle 10G and later:**

ALTER TABLE table_name

MODIFY column_name datatype

SQL AUTO INCREMENT Field

- Auto-increment allows a unique number to be generated when a new record is inserted into a table.
- Syntax for MySQL

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

SQL AUTO INCREMENT Field

- MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.
- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100
```

SQL AUTO INCREMENT Field

- Syntax for SQL Server

```
CREATE TABLE Persons
(
ID int IDENTITY(1,1) PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

- In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.

SQL Views

- In SQL, 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 Updating a View)

- **SQL CREATE OR REPLACE VIEW Syntax**

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

- **SQL DROP VIEW Syntax**

```
DROP VIEW view_name
```

Data type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

MySQL Data 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

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

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:MI: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:MI: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:MI:SS**Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format.**Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

SQL Functions

- SQL has many built-in functions for performing calculations on data.
- SQL Aggregate Functions
- SQL Scalar functions

SQL Aggregate Functions

- SQL aggregate functions return a single value, calculated from values in a column.
- 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

The AVG() Function

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

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

```
SELECT AVG(Price) AS PriceAverage FROM Products;
```

```
SELECT ProductName, Price FROM Products
WHERE Price>(SELECT AVG(Price) FROM Products);
```

SQL COUNT() Function

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

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

```
SELECT COUNT(CustomerID) AS OrdersFromCustomerID7 FROM Orders
WHERE CustomerID=7;
```

The MAX() Function & The MIN() Function

- The MAX() function returns the largest value of the selected column.

SELECT MAX(column_name) FROM table_name;

- The MIN() function returns the smallest value of the selected column.

SELECT MIN(column_name) FROM table_name;

The SUM() Function

- The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name) FROM table_name;
```

The 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.

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

The HAVING Clause

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

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