MySql Learning:

1) open the .bashrc or .bash\_profile on the linux terminal, /home/ubuntu/ path add env variable as below:

export PATH=${PATH}:/usr/local/mysql/bin/

2) login into the data base as below:

mysql -u root -p

3) CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),

species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);

[VARCHAR](https://dev.mysql.com/doc/refman/5.7/en/char.html) is a good choice for the name, owner, and species columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be 20. You can normally pick any length from 1 to 65535, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an [ALTER TABLE](https://dev.mysql.com/doc/refman/5.7/en/alter-table.html) statement.

3) Load the db (\*.sql) file as below:

source c:\temp\mysqlsampledatabase.sql

4) mysql> **show** **databases**; (to show all the dabases) or “show SCHEMAS;”,

5) To exit mysql terminal, type “quit;”

6) To select a particular db out of multiple DBs, “mysql->USE “dbname”; Or if the db does not exist, it will create one. Each time we login into the DB, we need to execute “USE” command to use a particular db.

7) “show tables” to list tables.

8) 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. **Fields** are nothing but the **“Columns”.**

9) SELECT \* FROM Customers; (Select all the records from the table).( SQL keywords are NOT case sensitive: select is the same as SELECT).

10)

* **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

11) To select on few columns or fields; (select customerName,phone from customers;)

12) Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Country FROM Customers;

13) SELECT COUNT(DISTINCT Country) FROM Customers;

14) The **WHERE** clause is used to filter records.

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
WHERE *condition*;

15) select \* from customers where state='CA' && contactLastName='Young';

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_equal_to) |
| > | Greater than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than) |
| < | Less than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than) |
| >= | Greater than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than2) |
| <= | Less than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than2) |
| <> | Not equal. **Note:** In some versions of SQL this operator may be written as != | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not_equal_to) |
| BETWEEN | Between a certain range | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| LIKE | Search for a pattern | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| IN | To specify multiple possible values for a column |  |

Instead of \*, use “%” to select multiple characters. (select \* from customers where contactLastName LIKE 'Ber%';) all conactnames starting with Ber

16) select \* from orders where orderNumber **<>** 10111;/\***not equal** \*/

17) select \* from orders where orderNumber **between** 10110 AND 10117;

18) select \* from orders where orderNumber <> 10111;/\*not equal \*/

select \* from orders where orderNumber between 10110 AND 10117;

select \* from orders where status **IN** ('On Hold', 'Disputed'); (match only selected key words).

19) The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

*select \* from orders where status='Shipped' AND shippedDate='2003-11-20'*

***select \* from orders where NOT status='Shipped';***

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

21) **ORDER BY:** The ORDER BY keyword is used to sort the result-set in ascending or descending order.

22) **INSERT INTO:** Used to insert new records into the table, 2 ways to insert:

a) It specifies both the column names and values to be inserted.

*INSERT INTO <tablename> (column1, column2,….) VALUES (value1,value2,value3…).*

b) If we are adding all the values into the columns, no need to specify the column names, ensure that all values is in the same order as the columns in the table.

*INSERT INTO <tablename> VALUES (value1,value2,value3…).*

**Error**: Cannot add or update a child row, a foreign key constraint fails

**Foreign Key Constraints**. A **foreign key** (FK): is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the **foreign key** table.

select \* from orders ORDER BY orderNumber ASC;

**23) The IS NULL Operator:**

INSERT INTO orders (orderNumber, orderDate,requiredDate,shippedDate,status,comments,customerNumber) values ('1', '2003-01-10', '2003-01-10', '2003-01-10', 'Shipped',"new order created", '103');

The IS NULL operator is used to test for empty values (NULL values).

**24) “IS NOT NULL” Operator:** The IS NOT NULL operator is used to test for non-empty values.

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

* select \* from orders where shippedDate IS NOT NULL;
* select \* from orders where shippedDate IS NULL;

25) **UPDATE Statement**: Is used to modify the existing records in a table.

Syntax: “update <tablename> set column1= value1, column2=value2,.. where condition”.

UPDATE orders set status='Shipped' where orderNumber='10406';

select \* from orders where orderNumber='10406';

**Note:** Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

Updating of all records will be done when the condition **where** is omitted or not specified.

26) **DELETE Statement:** is used to delete existing records in a table.

* Delete from table\_name where condition;

**Note:** Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

* DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste'; deletes complete row where customerName is set as above.

**DELETE all records**: “delete from customers”;

27) **SELECT TOP:**  is used to specify the number of records to return. The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

**Note:** Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses ROWNUM.

**SQL Server / MS Access Syntax:**

SELECT TOP number|*percent* column\_name(s)  
FROM table\_nameWHERE condition;

**MYSQL Syntax:**

SELECT column\_name(s)

From table\_name

Where condition

Limit number;

**ORACLE SYNTAX:**

SELECT column\_name(s)

From table\_name

Where rownum <= number;

* select \* from orders limit 3; displays first 3 lines.

28) selects the first 50% of the records from the "Customers" table.

SELECT TOP 50 PERCENT \* FROM Customers;

select \* from orders where status='Shipped' limit 5; (To list 5 rows where status is shipped);(for mysql).

SELECT \* FROM Customers  
 WHERE Country='Germany' AND ROWNUM <= 5; (for Oracle syntax).

## 29) **The SQL MIN() and MAX() Functions:**

**Syntax**: select MIN(column\_name) from table\_name where condition;

SELECT MAX(orderNumber) AS maxordernumber FROM orders; //output “maxordernumber” 10115

SELECT MIN(orderNumber) AS minordernumber FROM orders; // output “minordernumber” 1;

29) **The SQL COUNT(), AVG() and SUM() Functions:**

SELECT COUNT(column\_name)

FROM table\_name;

WHERE condition;

* select count(orderNumber) from orders; (give total no. of rows in a column).

SELECT COUNT(column\_name)

FROM table\_name;

WHERE condition;

* select sum(orderNumber) from orders where orderNumber between 10110 AND 10117;

SELECT SUM(*column\_name*)  
 FROM *table\_name*  
 WHERE *condition*;

* select avg(orderNumber) from orders where orderNumber between 10110 AND 10117; (10113.5000)

**Note:** NULL values are ignored.

30) **The SQL LIKE Operator:** this operator is used in a “WHERE” clause to search for a specified pattern in a column.

2 Wild Cards often used in conjunction with the “LIKE” operator:

% or \* -- percentage sign represents zero, one or multiple characters.

\_ or ? -- the underscore represents a single character.

**Note:** MS Access uses an asterisk (\*) instead of the percent sign (%), and a question mark (?) instead of the underscore (\_).

SELECT *column1, column2, ...*  
FROM *table\_name*  
WHERE *columnN* LIKE *pattern*;

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that start with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that end with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_%' | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE CustomerName LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

**Tip**: You can also combine any number of conditions using AND or OR operators.

31) select \* from customers where contactLastName LIKE 'M%'; (here output prints contactLastName starts with M).

32) Select all records where the value of the City column does NOT start with the letter "a".

SELECT \* FROM Customers WHERE City NOT LIKE 'a%'

33) SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the [SQL LIKE](https://www.w3schools.com/sql/sql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Wildcard Characters in MS Access

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| \* | Represents zero or more characters | bl\* finds bl, black, blue, and blob |
| ? | Represents a single character | h?t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ! | Represents any character not in the brackets | h[!oa]t finds hit, but not hot and hat |
| - | Represents a range of characters | c[a-b]t finds cat and cbt |
| # | Represents any single numeric character | 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295 |

Wildcard Characters in SQL Server

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| \_ | Represents a single character | h\_t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ^ | Represents any character not in the brackets | h[^oa]t finds hit, but not hot and hat |
| - | Represents a range of characters | c[a-b]t finds cat and cbt |

All the wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_%\_%' | Finds any values that starts with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a" and ends with "o" |

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

SELECT \* FROM Customers  
WHERE City LIKE '[bsp]%';

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

SELECT \* FROM Customers  
WHERE City LIKE '[a-c]%';

The two following SQL statements select all customers with a City NOT starting with "b", "s", or "p":

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';

34) The SQL IN Operator

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

The IN operator is a shorthand for multiple OR conditions.

* Select column\_name(s) from <table> where column\_name in (value1, value2)
* Select column\_name(s) from <table> where column\_name in (select statement)

*SELECT \* FROM Customers  
WHERE Country IN (SELECT Country FROM Suppliers);*

select \* from customers where phone in ('2967 555', '7675-3555');

select \* from customers where phone not in ('2967 555', '7675-3555');

**SQL Aliases:**

Aliases used to give a table or column in a table, a temporary name. Aliases are often used to make column names more readable.

An alias only exists for the duration of the query.

Column syntax:

* Select column\_name AS alias\_name From table\_name;
* Select column\_name (s) from table\_name AS alias\_name;
* SELECT CustomerID AS ID, CustomerName AS Customer  
  FROM Customers;
* SELECT CustomerName AS Customer, ContactName AS [Contact Person]  
  FROM Customers;
* SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS Address  
  FROM Customers;
* select customerNumber, addressLine1 + ',' + addressLine2 + ',' + city + ',' + state + ',' + postalCode + ',' + country AS CompAddress from customers;
* SELECT customerNumber, CONCAT(addressLine1,', ',addressLine2,', ',city,', ',state, ',', country) AS Address

FROM Customers;

* SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName  
  FROM Customers, Orders  
  WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;

**SQL JOIN:** is used to combine rows from 2 or more tables based on a related column between them.

select orders.orderNumber, orders.orderDate, orders.customerNumber from orders **INNER JOIN orderDetails ON orders.orderNumber=orderdetails.orderNumber;**

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table (The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

Select column\_name(s)

From table1

LEFT Join table2

ON table1.column\_name= table2.column.name;

**Note**: In some databases left join is called as Left outer join.

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

* **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table

Select column\_name(s)

From table1

RIGHT JOIN table2

ON table1.column\_name = table2.column\_name;

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
ORDER BY Orders.OrderID;

* **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

SELECT *column\_name(s)*  
FROM *table1*  
FULL OUTER JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*  
WHERE *condition*;

* The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
* **Note:** FULL OUTER JOIN can potentially return very large result-sets!
* **Tip:** FULL OUTER JOIN and FULL JOIN are the same.

SELECT *column\_name(s)*  
FROM *table1*  
FULL OUTER JOIN *table2*ON *table1.column\_name*=*table2.column\_name*WHERE *condition*;

**SQL self join:** A self join is a regular join, but the table is joined with itself.

Select column\_name(s)

From table T1, table T2

Where condition

T1 and T2 are different table aliases for the same table.

**DATABASE:**

1) Create a Database (CREATE DATABASE db\_name).

2) Drop or delete a dabase ( DROP DATABASE db\_name)

**Note:** Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

**Tip:** Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

3) Backup a database :

BACKUP DATABASE *databasename*  
TO DISK = '*filepath*'  
WITH DIFFERENTIAL;

Ex:

*BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak';*

***Tip: Always back up the database to a different drive than the actual database. Then, if you get a disk crash, you will not lose your backup file along with the database.***

4) BACKUP WITH DIFFERENTIAL Example

BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak'  
WITH DIFFERENTIAL;

**Tip:** A differential back up reduces the back up time (since only the changes are backed up).

5) Crete Table is used to create a table in database.

CREATE TABLE *table\_name*(  
*column1 datatype*,  
*column2 datatype*,  
*column3 datatype*,  
   ....  
);

CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);

After that insert data into above table using (INSERT INTO)

**6) Create Table Using Another Table:**

**CREATE TABLE *new\_table\_name* AS  
    SELECT *column1, column2,...*  
    FROM *existing\_table\_name*  
    WHERE .**

**7) DROP TABLE: is used to drop an existing table in a database. (DROP TABLE table\_name).**

Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

8) **TRUNCATE TABLE**: is used to delete all data inside a table, but not the table.

Ex: Truncate table table\_name;

9) **SQL ALTER TABLE: “**Alter table” statement is used to add, modify and delete columns in an existing table. It is also used to add and drop various constraints on an existing table.

**ALTER TABLE *table\_name*  
ADD *column\_name datatype*;**

**ADD a column:**

*ALTER TABLE Customers //Adds an email column to the existing “Customers table”  
ADD Email varchar(255);*

**Drop a column**: “Alter table customers DROP column email”;

**Alter/Modify a column**: Alter table <tablename> Modify column column\_name data type:

ALTER TABLE table\_name  
MODIFY column\_name datatype;

*select \* from Student;*

*CREATE TABLE Student (studentId int, studentName varchar(255));*

*INSERT INTO Student (studentId, studentName) values (123, 'Shankar');*

*INSERT INTO Student (studentId, studentName) values (124, 'Krishna');*

*INSERT INTO Student (studentId, studentName) values (123, 'Vasudeva');*

*INSERT INTO Student (studentId, studentName) values (125, 'Vasudeva');*

*ALTER TABLE Student ADD studentClass varchar(255);*

*ALTER TABLE Student DROP COLUMN studentClass;*

*ALTER TABLE Student Modify studentName int;*

*Can we alter the data type of a column which is varchar to an int or other types?*

**SQL CONSTRAINTS:** SQL constraints are used to specify rules for data in a table.

SQL CREATE constraints:

Constraints can be specified when the table is created with the “CREATE TABLE” statement or after a table is created with “ALTER TABLE” statement.

* CREATE TABLE table\_name ( column1 datatype constraint, column2 datatype constraint, column3 datatype constraint…);
* Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
* Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
* The following constraints are commonly used in SQL:

a) **NOT NULL:** Ensures that a column cannot have a NULL value.

**b) UNIQUE:** Ensures that all values in a column are different.

c) **PRIMARY KEY:**  A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.

d) **FOREIGN KEY:**  Uniquely identifies a row /record in another table

e) **CHECK:** Ensures that all values in a column satisfies specific condition.

f) **DEFAULT:** Set a default value for a column when no value is specified.

g) **INDEX:**  Used to create and retrieve data from a database very quickly.

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

This enforces a field to always contain a value, which means that you cannot insert a new record or update a record without adding a value to the field.

**NOTE**: Most critically, all existing NULL values within the column **must** be updated to a non-null value before the ALTER command can be successfully used and the column made NOT NULL. Any attempt to set the column to NOT NULL while actual NULL data remains in the column will result in an error and no change will occur.

b) UNIQUE:

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

Note: However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL Server

CREATE TABLE StudentData (

ID int NOT NULL UNIQUE,

lastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int );

**MySQL:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    UNIQUE (ID)  
);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT UC\_Person UNIQUE (ID,LastName)  
);

**SQL UNIQUE Constraint on ALTER TABLE**

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

ALTER TABLE students

ADD UNIQUE (ID);

To name a unique constraint and to define unique constraint on multiple columns, use following syntax

ALTER TABLE students

ADD CONSTRAINT UC\_Student UNIQUE (ID, Last\_name);

DROP Unique Constraints:

**To drop unique constraint:**

**Mysql:**

ALTER TABLE Students

DROP Index UC\_Person

**SQL Server/Oracle/Ms Access:**

ALTER TABLE Students

DROP Constraints UC\_Person

**PRIMARY KEY Constraint:**

The primary key constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL Values.

A table can have only ONE primary key; and in the table, this primary key consists of single or multiple columns(fields).

**SQL Primary Key on CREATE TABLE:**

**MySQL:**

CREATE TABLE Persons (

ID int NOT NULL,

Lastname varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

PRIMARY KEY (ID)

);

SQL SERVER / Oracle /Ms Access:

CREATE TABLE Persons (

ID int NOT NULL PRIMARY KEY,

Lastname varchar(255) NOT NULL,

Firstname varchar(255),

Age int

);

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 (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)  
);

**Note:** In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

**SQL Foreign Key:**

a) A Foreign key is a key used to link 2 tables together. A foreign Key is a field ( or collection of fields) in one table that refers to the Primary Key in another table.

b) The table containing the Foregin key is called the child table and the table containing candidate key is called the referenced or parent table.

Persons" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **PersonID** | **LastName** | **FirstName** | **Age** |
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |
| 3 | Pettersen | Kari | 20 |

"Orders" table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

**MYSQL:**

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

**SQL SERVER/ Oracle /MS ACCESS:**

CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,  
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)  
);

**To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:**

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

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

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

ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

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

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To drop a FOREIGN KEY constraint, use the following SQL:

MYSQL:

ALTER TABLE Orders  
DROP FOREIGN KEY FK\_PersonOrder;

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

ALTER TABLE Orders  
DROP CONSTRAINT FK\_PersonOrder;

## **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 on CREATE TABLE:**

**MySQL:**

CREATE TABLE Persons (

ID int NOT NULL,

Lastname varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

CHECK (Age>=18)

);

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int CHECK (Age>=18)  
);

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255),  
    CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')  
);

SQL CHECK on ALTER TABLE:

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

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

ALTER TABLE Persons  
ADD CHECK (Age>=18);

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

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

ALTER TABLE Persons  
ADD CONSTRAINT CHK\_PersonAge CHECK (Age>=18 AND City='Sandnes');

DROP a Check Constraint:

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

ALTER TABLE Persons  
DROP CONSTRAINT CHK\_PersonAge;

**MySQL:**

ALTER TABLE Persons  
DROP CHECK CHK\_PersonAge;

**SQL DEFAULT CONSTRAINT:**

The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

**SQL DEFAULT on CREATE TABLE:**

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT GETDATE()  
);

**On Alter Table:**

**MYSQL:**

ALTER TABLE Persions

ADD City SET DEFAULT ‘Sadness’;

**SQL SERVER:**

ALTER TABLE Persons

ADD CONSTRAINT df\_city

DEFAULT ‘Sadness’ FOR City;

**MS ACCESS:**

ALTER TABLE Persons  
ALTER COLUMN City SET DEFAULT 'Sandnes';

**Oracle:**

ALTER TABLE Persons  
MODIFY City DEFAULT 'Sandnes';

**DROP a DEFAULT Constraint:**

**MySQL:**

ALTER TABLE Persons  
ALTER City DROP DEFAULT;

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

ALTER TABLE Persons  
ALTER COLUMN City DROP DEFAULT;

**SQL CREATE INDEX:**

The “CREATE INDEX” statement is used to create indexes in the tables.

Indexes are used to retrieve data from the database more quickly than otherwise. 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, only create indexes on columns that will be frequently searched against.

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX *index\_name*  
ON *table\_name* (*column1*, *column2*, ...);

**CREATE UNIQUE INDEX Syntax:**

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name  
ON table\_name (column1, column2, ...);

**Note:** The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

CREATE INDEX idx\_lastname  
ON Persons (LastName);

CREATE INDEX idx\_pname  
ON Persons (LastName, FirstName);

**DROP INDEX:**

**MS Access:**

DROP INDEX index\_name ON table\_name;

**SQL Server:**

DROP INDEX *table\_name*.*index\_name*;

**DB2/Oracle:**

DROP INDEX *index\_name*;

**MySQL:**

ALTER TABLE *table\_name*  
DROP INDEX *index\_name*;

## **SQL AUTO INCREMENT Field:**

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

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

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

CREATE TABLE Persons (  
    Personid int NOT NULL AUTO\_INCREMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (Personid)  
);

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;

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

## **Syntax for SQL Server:**

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

CREATE TABLE Persons (  
    Personid int IDENTITY(1,1) PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

The MS SQL Server uses the IDENTITY keyword to perform an auto-increment feature.

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

**Tip:** To specify that the "Personid" column should start at value 10 and increment by 5, change it to IDENTITY(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

## **Syntax for Access:**

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

CREATE TABLE Persons (  
    Personid AUTOINCREMENT PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

The MS Access uses the AUTOINCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTOINCREMENT is 1, and it will increment by 1 for each new record.

**Tip:** To specify that the "Personid" column should start at value 10 and increment by 5, change the autoincrement to AUTOINCREMENT(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

**Syntax for Oracle:**

In Oracle the code is a little bit more tricky.

You will have to create an auto-increment field with the sequence object (this object generates a number sequence).

Use the following CREATE SEQUENCE syntax:

CREATE SEQUENCE seq\_person  
MINVALUE 1  
START WITH 1  
INCREMENT BY 1  
CACHE 10;

The code above creates a sequence object called seq\_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "Persons" table, we will have to use the **nextval** function (this function retrieves the next value from seq\_person sequence):

INSERT INTO Persons (Personid,FirstName,LastName)  
VALUES (seq\_person.nextval,'Lars','Monsen');

**SQL DATES:**

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

**SQL Date Data Types:**

**MySQL** comes with the following data types for storing a date or a date/time value in the database:

* DATE - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
* YEAR - format YYYY or YY

**SQL Server** comes with the following data types for storing a date or a date/time value in the database:

* DATE - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: a unique number

**Note:** The date types are chosen for a column when you create a new table in your database!

Note: You can compare two dates easily if there is no time component involved!

**Tip:** To keep your queries simple and easy to maintain, do not allow time components in your dates!

**SQL VIEWS:**

**CREATE VIEW:**

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.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

*CREATE VIEW view\_name AS*

*Select column1, column2,..*

*From table\_name*

*Where condition;*

**Note:** A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

CREATE VIEW USA\_customers AS

select customerName, addressLine1, city

from customers

where country = 'USA';

select \* from USA\_customers;

ex:2

CREATE VIEW [BRAZIL CUSTMOERS]

SELECT CustomerName, contactName

From Customers

Where Country = ‘Brazil’;

SELECT \* FROM [BRAZIL CUSTMOERS]

Ex:3

CREATE VIEW [Products Above Avg Price]

Select ProductName, Price

From Products

Where Price > (SELECT AVG(Price) from Products];

SELECT \* FROM [[Products Above Avg Price];

**Updating a VIEW: (Use either UPDATE or REPLACE VIEW):**

CREATE OR REPLACE <view\_name> AS

SELECT CusotmerName, ContactName, City

From Customers

Where country = ‘India”

**DROP A VIEW:**

DROP VIEW <view\_name>

**SQL INJECTION:**

a) SQL Injection is a Code injection technique that might destroy database.

b) SQL Injection is the most common web hacking technique.

c) SQL injection is the placement of malicious code into the SQL statements, via web page input.

**SQL in WebPages:**

a) SQL injection usually occurs when you ask a user for input, like their username/userId , instead of name/Id and user gives you SQL statement that you will run unknowingly on your database.

b) Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

*txtUserId = getRequestString(“UserId”);  
txtSQL = “SELECT \* FROM Users WHERE UserId = “ + txtUserId;*

C) SQL Injection Based on 1=1 is Always True:

SELECT \* FROM Users WHERE UserId = 105 OR 1=1;

d) SQL Injection Based on ""="" is Always True:

uName = getRequestString("username");  
uPass = getRequestString("userpassword");  
  
sql = 'SELECT \* FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'

e) SELECT \* FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""

f) SQL Injection Based on Batched SQL Statements

SELECT \* FROM Users; DROP TABLE Suppliers

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;

**Use SQL Parameters for Protection:**

To protect a web site from SQL injection, you can use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

**ASP.NET Razor example:**

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = @0";  
db.Execute(txtSQL,txtUserId);

Note that parameters are represented in the SQL statement by a @ marker.

The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

txtNam = getRequestString("CustomerName");  
txtAdd = getRequestString("Address");  
txtCit = getRequestString("City");  
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";  
db.Execute(txtSQL,txtNam,txtAdd,txtCit);

**SQL Hosting:**

If you want your web site to be able to store and retrieve data from a database, your web server should have access to a database-system that uses the SQL language.

If your web server is hosted by an Internet Service Provider (ISP), you will have to look for SQL hosting plans.

The most common SQL hosting databases are MS SQL Server (Micro Soft SQL Server), Oracle, MySQL, and MS Access.

**MS SQL Server: (Micro Soft SQL SERVER);**

Microsoft's SQL Server is a popular database software for database-driven web sites with high traffic.

SQL Server is a very powerful, robust and full featured SQL database system.

**Oracle:**

Oracle is also a popular database software for database-driven web sites with high traffic.

Oracle is a very powerful, robust and full featured SQL database system.

**MySQL:**

MySQL is also a popular database software for web sites.

MySQL is a very powerful, robust and full featured SQL database system.

MySQL is an inexpensive alternative to the expensive Microsoft and Oracle solutions.

**Access:**

When a web site requires only a simple database, Microsoft Access can be a solution.

Access is not well suited for very high-traffic, and not as powerful as MySQL, SQL Server, or Oracle.

**SQL References:**

**SQL KeyWords:**

a) ALTER COLUMN: is used to change the data type of a column in a Table

ALTER TABLE Employees

ALTER COLUMN BirthdDate year;

b)