**Mysql 5.6.35 installation:**

* rpm –qa | grep maria
  + we will get the result
    - postfix
    - maria-libs
* rpm –e postfix\_\_\_\_
* rpm –e maria-libs
* create a folder mysql and copy the packges

i.e mysql-community-client-5.6.35

mysql-community-common-5.6.35

mysql-community-libs-5.6.35

mysql-community-server-5.6.35

**Note: you can find the packages in https://repo.mysql.com/yum/mysql-5.6-community/el/7/x86\_64/**

* within that folder
* yum localinstall \*.rpm
* mysql\_secure\_installation

it will us to set the password (‘Admin@123’)

* mysql -u root -p

**Normalisation :**

**Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

**Anomalies in DBMS:**

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

**Example**: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp\_id, emp\_name, emp\_address and emp\_dept for storing the department details in which the employee works. At some point of time the table looks like this:



The above table is not normalized.

**Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp\_dept field doesn’t allow nulls.

**Delete anomaly**: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp\_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

## First normal form (1NF):

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

## Second normal form (2NF):

A table is said to be in 2NF if both the following conditions hold:

* Table is in 1NF (First normal form)
* No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

**Third Normal form (3NF):**

A table design is said to be in 3NF if both the following conditions hold:

* Table must be in 2NF
* [**Transitive functional dependency**](http://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

An attribute that is not part of any [**candidate key**](http://beginnersbook.com/2015/04/candidate-key-in-dbms/) is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

* X is a [**super key**](http://beginnersbook.com/2015/04/super-key-in-dbms/) of table
* Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

## Boyce Codd normal form (BCNF):

It is an advance version of 3NF that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every [**functional dependency**](http://beginnersbook.com/2015/04/functional-dependency-in-dbms/) X->Y, X should be the super key of the table.

### Types of keys in DBMS

[**Primary Key**](http://beginnersbook.com/2015/04/primary-key-in-dbms/) – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

[**Super Key**](http://beginnersbook.com/2015/04/super-key-in-dbms/) – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.

[**Candidate Key**](http://beginnersbook.com/2015/04/candidate-key-in-dbms/) – A super key with no redundant attribute is known as candidate key

[**Alternate Key**](http://beginnersbook.com/2015/04/alternate-key-in-dbms/) – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

[**Composite Key**](http://beginnersbook.com/2015/04/composite-key-in-dbms/) – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

[**Foreign Key**](http://beginnersbook.com/2015/04/foreign-key-in-dbms/) – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

## SQL CREATE INDEX Statement:

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

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

### **CREATE INDEX Syntax**

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

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

### **CREATE UNIQUE INDEX Syntax**

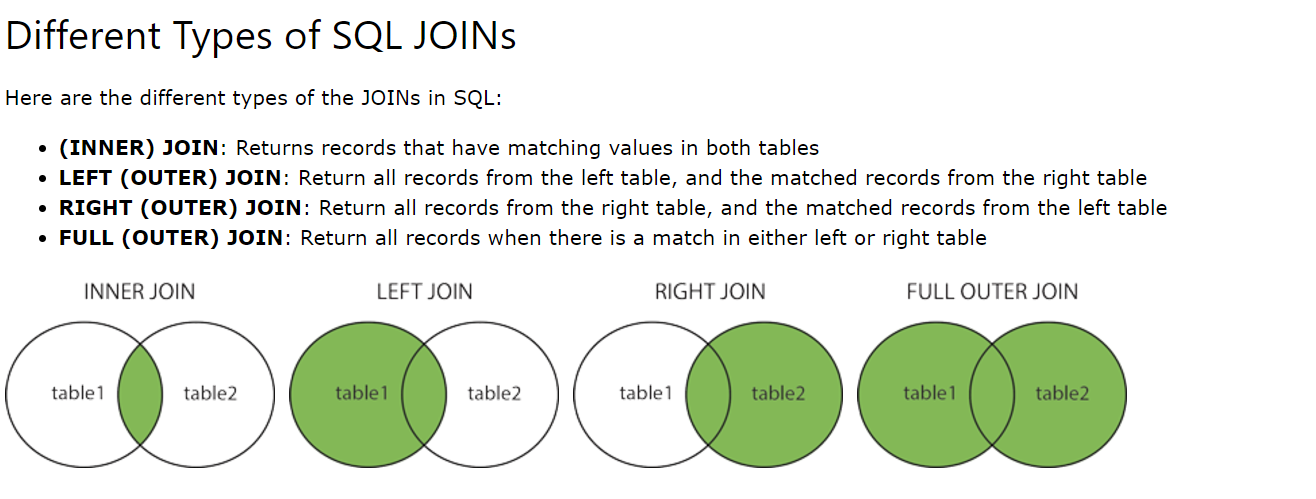
Creates a unique index on a table. Duplicate values are not allowed:

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

## DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

ALTER TABLE table\_nameDROP INDEX index\_name;



## The SQL UNION Operator

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

* Each SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in each SELECT statement must also be in the same order

### **UNION Syntax**

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

### **UNION ALL Syntax**

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

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

----------------------------------------------------------------------------------------------------------------------------------------

#### **Cursor CLOSE Syntax:**

CLOSE ***cursor\_name***

This statement closes a previously opened cursor.

#### **Cursor DECLARE Syntax:**

DECLARE ***cursor\_name*** CURSOR FOR ***select\_statement***

This statement declares a cursor and associates it with a [SELECT](https://dev.mysql.com/doc/refman/5.6/en/select.html) statement that retrieves the rows to be traversed by the cursor. To fetch the rows later, use a [FETCH](https://dev.mysql.com/doc/refman/5.6/en/fetch.html) statement. The number of columns retrieved by the [SELECT](https://dev.mysql.com/doc/refman/5.6/en/select.html) statement must match the number of output variables specified in the [FETCH](https://dev.mysql.com/doc/refman/5.6/en/fetch.html) statement.

#### **Cursor FETCH Syntax:**

FETCH [[NEXT] FROM] ***cursor\_name*** INTO ***var\_name*** [, ***var\_name***] ...

This statement fetches the next row for the [SELECT](https://dev.mysql.com/doc/refman/5.6/en/select.html) statement associated with the specified cursor (which must be open), and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the [SELECT](https://dev.mysql.com/doc/refman/5.6/en/select.html)statement must match the number of output variables specified in the [FETCH](https://dev.mysql.com/doc/refman/5.6/en/fetch.html) statement.

#### **Cursor OPEN Syntax:**

OPEN ***cursor\_name***

### **CREATE VIEW Syntax:**

CREATE

[OR REPLACE]

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]

[DEFINER = { ***user*** | CURRENT\_USER }]

[SQL SECURITY { DEFINER | INVOKER }]

VIEW ***view\_name*** [(***column\_list***)]

AS ***select\_statement***

[WITH [CASCADED | LOCAL] CHECK OPTION]

The [CREATE VIEW](https://dev.mysql.com/doc/refman/5.6/en/create-view.html) statement creates a new view, or replaces an existing view if the OR REPLACE clause is given. If the view does not exist,[CREATE OR REPLACE VIEW](https://dev.mysql.com/doc/refman/5.6/en/create-view.html) is the same as [CREATE VIEW](https://dev.mysql.com/doc/refman/5.6/en/create-view.html). If the view does exist, [CREATE OR REPLACE VIEW](https://dev.mysql.com/doc/refman/5.6/en/create-view.html) is the same as [ALTER VIEW](https://dev.mysql.com/doc/refman/5.6/en/alter-view.html).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**procedures:**

CREATE PROCEDURE curdemo6()

BEGIN

SET @s='select \*from address where addressid=\'nirmal\'';

SET @l='insert into test values(?)';

PREPARE stn FROM @l;

EXECUTE stn USING @s;

END;

**Sample Code:**

**CREATE PROCEDURE USER\_DET**

**(IN CHAR(15))**

**LANGUAGE SQL MODIFIES SQL DATA**

**SELECT email FROM user WHERE username=USER;**

**SELECT MAX(idd) FROM Employee**

**WHERE Salary NOT IN (SELECT MAX(Salary) FROM Employee )**

**....**

**SELECT \* FROM samplee Emp1 WHERE (3-1) = (SELECT COUNT(DISTINCT(Emp2.idd)) FROM samplee Emp2 WHERE Emp2.idd > Emp1.idd)**

**...................**

**CREATE PROCEDURE curdemo6()**

**BEGIN**

**SET @s='select \*from address where addressid=\'nirmal\'';**

**SET @l='insert into test values(?)';**

**PREPARE stn FROM @l;**

**EXECUTE stn USING @s;**

**END;**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CREATE PROCEDURE curdemo7()**

**BEGIN**

**SET autocommit=0;**

**START TRANSACTION;**

**UPDATE address SET pin=50801 WHERE addressid='nalgonda';**

**ROLLBACK;**

**END;**

**-----------------**

**CREATE TRIGGER beforeaddress\_update**

**BEFORE UPDATE ON address**

**FOR EACH ROW**

**BEGIN**

**call Sam();**

**END;**

**DELIMITER ;**

**-----------**

**CREATE TRIGGER sum BEFORE INSERT ON expence**

**FOR EACH ROW SET @amount = @amount + NEW.amount;**

**----------------------**

**SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';**

**PREPARE stmt2 FROM @s;**

**SET @a = 6;**

**SET @b = 8;**

**mysql> EXECUTE stmt2 USING @a, @b;**

**...........................**

**SET TRANSACTION ISOLATION LEVEL**

**{ READ UNCOMMITTED**

**| READ COMMITTED**

**| REPEATABLE READ**

**| SNAPSHOT**

**| SERIALIZABLE**

**}**

**[ ; ]**

**------**

**SELECT COUNT(amount),username FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5 ORDER BY COUNT(CustomerID) DESC;**