Create Table

When creating a table, we need to specify the table name, its columns, and the backing data types for the columns. There are other constraints that we can specify when creating a table but for now we'll demonstrate creating the table in the simplest possible manner.

```
CREATE TABLE tableName (

col1 <dataType> <Restrictions>,

col2 <dataType> <Restrictions>,

col3 <dataType> <Restrictions>,

<Primary Key or Index definitions>);
```

```
CREATE TABLE Actors (
FirstName VARCHAR(20),
SecondName VARCHAR(20),
DoB DATE,
Gender ENUM('Male','Female','Transgender'),
MaritalStatus ENUM('Married', 'Divorced', 'Single'),
NetWorthInMillions DECIMAL);
```

```
mysql> DESC Actors;
+--------------------+-----------------------------------+------+-----+---------+-------+
| Field              | Type                              | Null | Key | Default | Extra |
+--------------------+-----------------------------------+------+-----+---------+-------+
| FirstName          | varchar(20)                       | YES  |     | NULL    |       |
| SecondName         | varchar(20)                       | YES  |     | NULL    |       |
| DoB                | date                              | YES  |     | NULL    |       |
| Gender             | enum('Male','Female','Transgender') | YES  |     | NULL    |       |
| MaritalStatus      | enum('Married','Divorced','Single') | YES  |     | NULL    |       |
| NetWorthInMillions | decimal(10,0)                     | YES  |     | NULL    |       |
+--------------------+-----------------------------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Inserting Data

In the previous lessons we created our example table, **Actors**. But a table without any data is not very useful. In this lesson we'll learn how to add data into a table using the **INSERT** statement. We'll retrieve the added rows using the **SELECT** keyword. We'll learn more about using **SELECT** in the next lesson, but for now, it suffices to know that it is used for retrieving rows from a table.

```
mysql> INSERT INTO Actors (
    -> FirstName, SecondName,DoB, Gender, MaritalStatus, NetworthInMillions)
    -> VALUES ("Brad", "Pitt", "1963-12-18","Male", "Single", 240.00);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Actors (
    -> FirstName, SecondName,DoB, Gender, MaritalStatus, NetworthInMillions)
    -> VALUES
    ->
    -> ("Jennifer", "Aniston", "1969-11-02","Female", "Single", 240.00),
    ->
    -> ("Angelina", "Jolie", "1975-06-04","Female", "Single", 100.00),
    ->
    -> ("Johnny", "Depp", "1963-06-09","Male", "Single", 200.00);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Operator	Purpose
>	Greater than operator
>=	Greater than or equal to operator
<	Less than operator
<=	Less than or equal to operator
!=	Not equal operator
<>	Not equal operator
<=>	NULL-safe equal to operator
=	Equal to operator
BETWEEN ... AND ...	Whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
GREATEST()	Return the largest argument
IN	Whether a value is within a set of values
INTERVAL	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

Next, say we are required to retrieve the next 4 richest actors after the top three. We can do so by specifying the number of rows we want after the top three rows using the **OFFSET** keyword.

```
SELECT FirstName, SecondName from Actors ORDER BY NetWorthInMillions DESC LIMIT 4 O
FFSET 3;
```

Aggregate Methods

```sql
-- Query 1
SELECT COUNT(*) FROM Actors;

-- Query 2
SELECT SUM(NetworthInMillions) FROM Actors;

-- Query 3
SELECT AVG(NetWorthInMillions) FROM Actors;

-- Query 4
SELECT MIN(NetWorthInMillions) FROM Actors;

-- Query 5
SELECT MAX(NetWorthInMillions) FROM Actors;

-- Query 6
SELECT STDDEV(NetWorthInMillions) FROM Actors;
```

Window Functions

Types of Joins

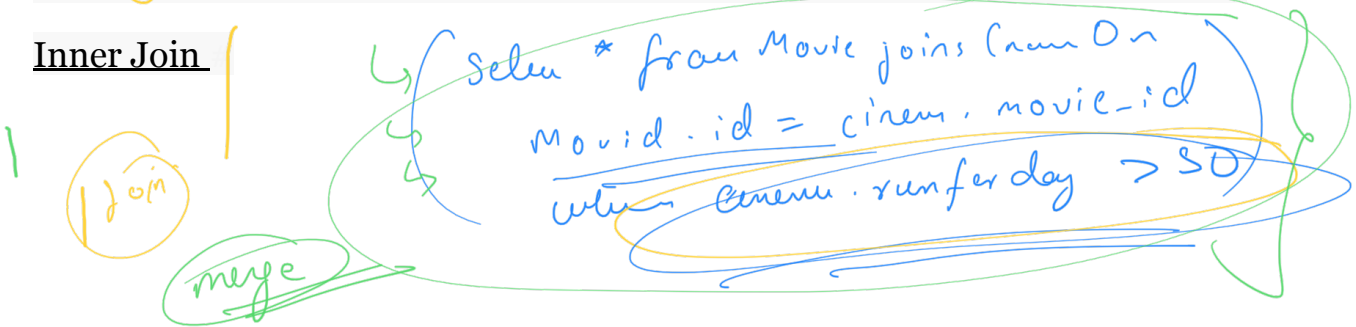
Movie Table

MovieID	MovieName
1	Star Wars
2	Sholay
3	The Italian Job

Cinema Table

MovieID	Cinema-Name	RunForDays
2	Naz Cinema	101
5	Apollo Theater	45

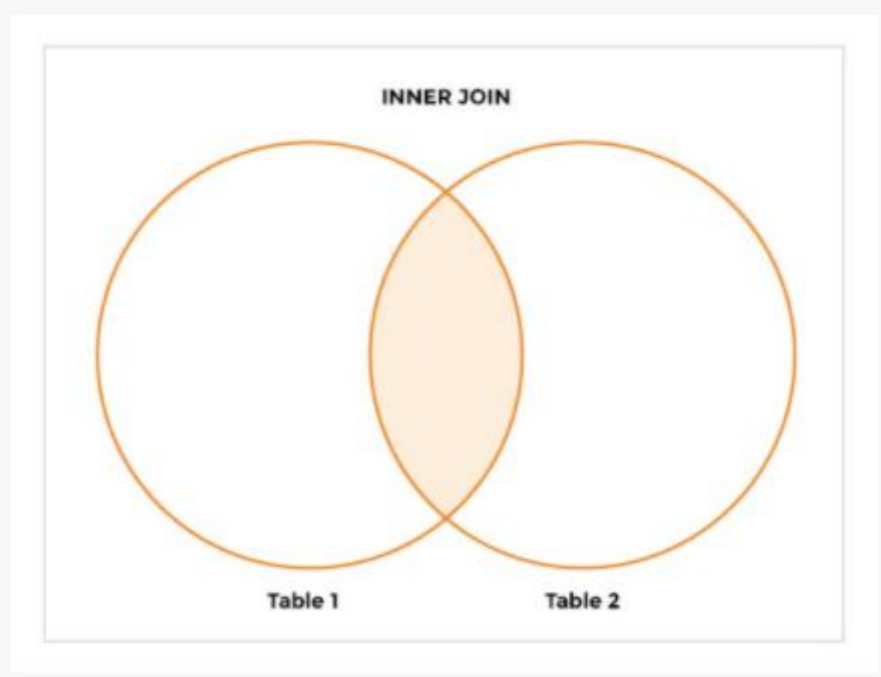
Inner Join



In case of an inner join, a condition, or multiple conditions, are tested to determine if a row from Table A should be joined with a row from Table B. This condition is called the **join predicate**. In the case of our example, the two tables share the movie iD column as the common value between them. The movie iD column establishes a relation between the two tables. Using the common column, we can determine if a given movie was screened on any of the theaters we have in our database and if so, then for how many days.

ORMs

MovieID	MovieName	MovieID	CinemaName	RunForDays
2	Sholay	2	Naz Cinema	101

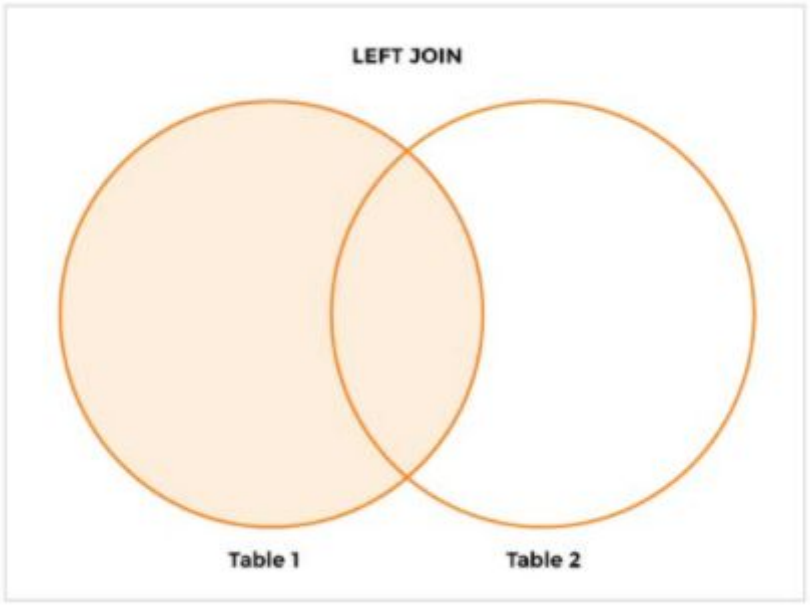


Left Outer Join

In the case of left join, the result set consists of rows that match the join predicate and also rows from the table specified on the left of the left join clause that doesn't match the join predicate. Null is inserted for columns from Table B and for rows from table A that didn't satisfy the join predicate. Said a different way, all rows from the left are always included in the result

set and rows from the right are only included if they match the join predicate.

MovieID	MovieName	MovieID	CinemaName	RunForDays
1	Star Wars	**NULL**	**NULL**	**NULL**
2	Sholay	2	Naz Cinema	101
3	The Italian Job	**NULL**	**NULL**	**NULL**



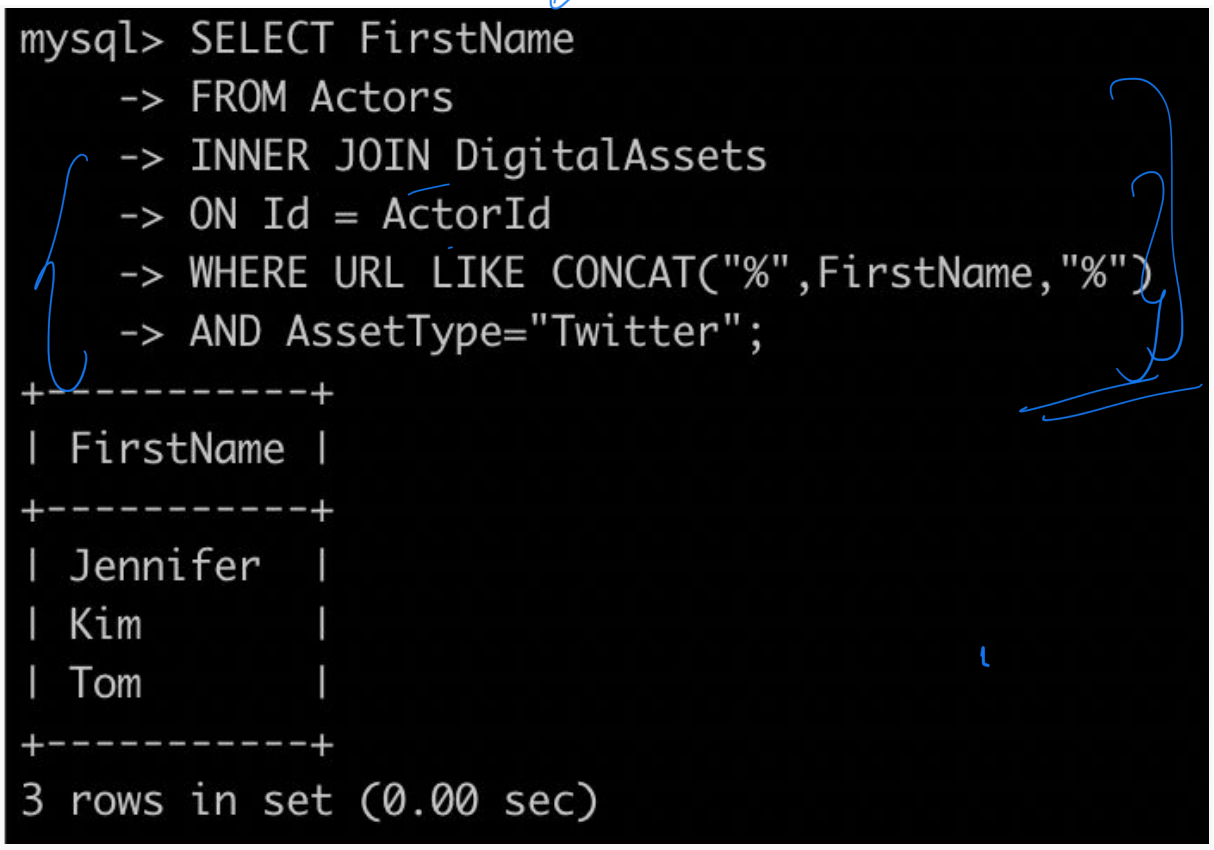
Right Outer Join

The right join is the reverse of the left join. In this case, all rows from the right table are always included in the result set and only those rows from the left table make it to the result set that satisfies the join condition. With left and outer joins, we specify which side of the join is allowed to have a row in the result when the join predicate isn't satisfied.

MovieID	MovieName	MovieID	CinemaName	RunForDays
2	Sholay	2	Naz Cinema	101
NULL	**NULL**	5	Apollo Theater	45

Correlated Queries

Correlated queries are a type of nested queries. The distinguishing feature about them is that the inner query references a table or a column from the outer query.



```
mysql> SELECT FirstName
    -> FROM Actors
    -> INNER JOIN DigitalAssets
    -> ON Id = ActorId
    -> WHERE URL LIKE CONCAT("%",FirstName,"%")
    -> AND AssetType="Twitter";
+-----------+
| FirstName |
+-----------+
| Jennifer  |
| Kim       |
| Tom       |
+-----------+
3 rows in set (0.00 sec)
```

The inner query references the column FirstName in its WHERE clause even though FirstName is part of the **Actors** table which is referenced only in the outer query. It is legal to access a table or any of its columns referenced in the outer query inside a sub-query. The value of FirstName for each row in the **Actors** table is provided as a scalar value to the inner query.

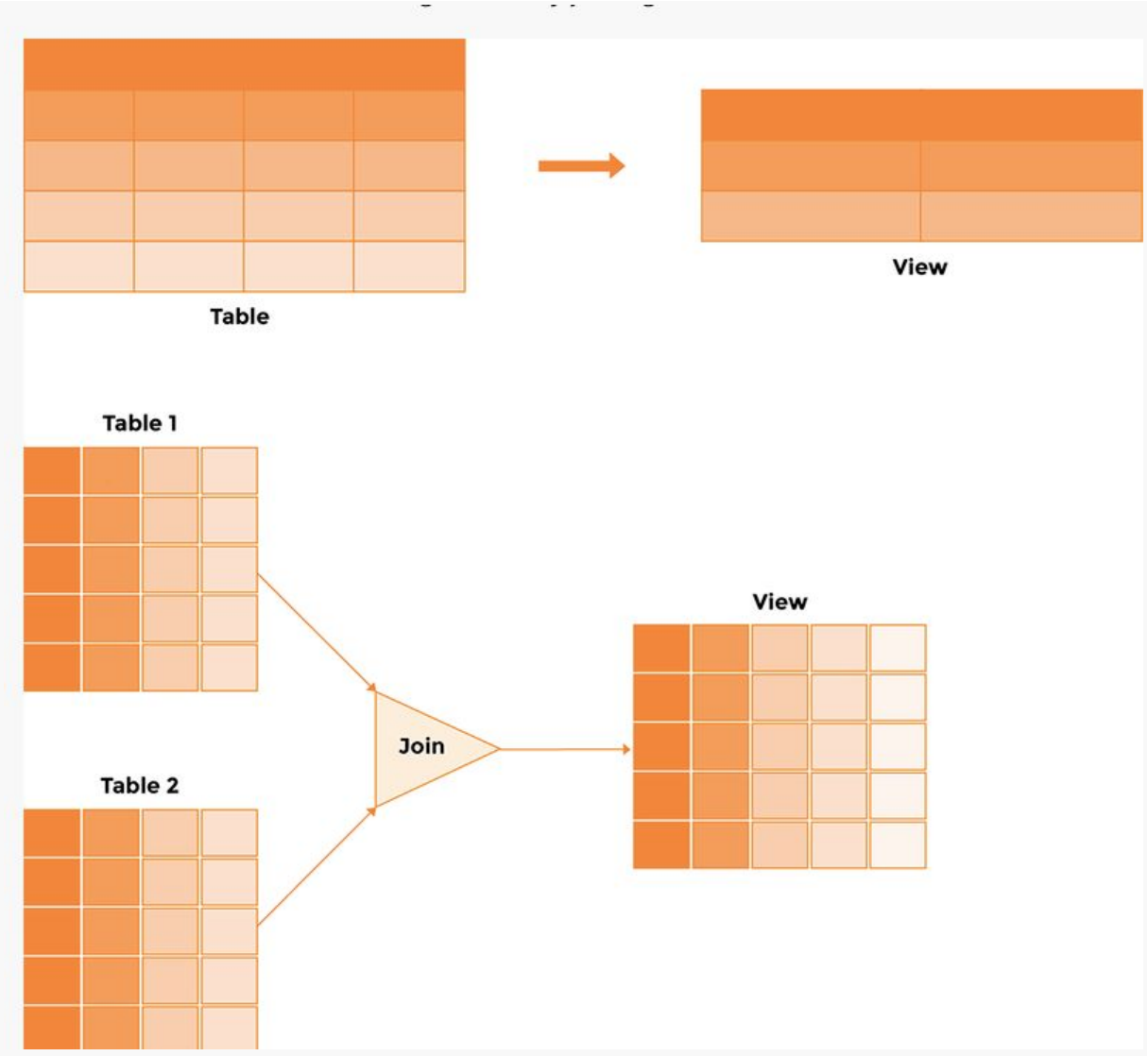
Pay attention to how we have used the **EXISTS** operator. We want to check for a condition that is true about each actor. We aren't interested in what the inner query returns for each actor, rather only if it returns anything or not. If no result is returned for an actor, then the outer query for that actor evaluates to false and the name isn't printed on the console.

Creating a View

Views are virtual tables that are created as a result of a SELECT query. They offer a number of advantages such as showing only a subset of data that is meaningful to users or restricting the number of rows and columns shown for security reasons. A view containing columns from multiple tables can

simplify queries by changing a multi-table query to a single-table query against a view. Views are stored in the database along with tables.

A view can be created from a single table, by joining two tables, or from another view.



CREATE [OR REPLACE] VIEW **view_name** AS

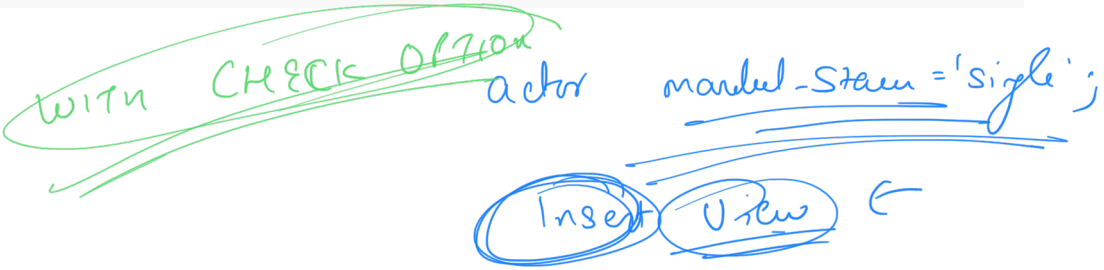
SELECT **col1, col2, ...coln**

FROM **table**

WHERE < **condition**>

```
mysql> CREATE VIEW DigitalAssetCount AS
    -> SELECT ActorId, COUNT(AssetType) AS NumberOfAssets
    -> FROM DigitalAssets
    -> GROUP BY ActorId;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

```
mysql> SELECT * FROM DigitalAssetCount;
+---------+-----------------+
| ActorId | NumberOfAssets  |
+---------+-----------------+
|       1 |               2 |
|       2 |               3 |
|       3 |               3 |
|       4 |               1 |
|       5 |               4 |
|       6 |               3 |
|       8 |               3 |
|      10 |               2 |
+---------+-----------------+
8 rows in set (0.00 sec)
```



```
mysql> SHOW TABLES;
+--------------------------+
| Tables_in_MovieIndustry |
+--------------------------+
| Actors                   |
| DigitalAssetCount        |
| DigitalAssets            |
+--------------------------+
3 rows in set (0.00 sec)
```

```
mysql> SHOW FULL TABLES;
+--------------------------+------------+
| Tables_in_MovieIndustry | Table_type |
+--------------------------+------------+
| Actors                   | BASE TABLE |
| DigitalAssetCount        | VIEW       |
| DigitalAssets            | BASE TABLE |
+--------------------------+------------+
3 rows in set (0.00 sec)
```

```
mysql>
mysql> mysql> CREATE VIEW ActorsTwitterAccounts AS
    -> SELECT FirstName, SecondName, URL
    -> FROM Actors
    -> INNER JOIN DigitalAssets
    -> ON Actors.Id = DigitalAssets.ActorID
    -> WHERE AssetType = 'Twitter';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM ActorsTwitterAccounts;
+-----------+------------+-----------------------------------+
| FirstName | SecondName | URL                               |
+-----------+------------+-----------------------------------+
| Shahrukh  | Khan       | https://twitter.com/iamsrk        |
| Jennifer  | Aniston    | https://twitter.com/jenniferannistn |
| Angelina  | Jolie      | https://twitter.com/joliestweet   |
| Kim       | Kardashian | https://twitter.com/KimKardashian |
| Natalie   | Portman    | https://twitter.com/natpdotcom    |
| Tom       | Cruise     | https://twitter.com/TomCruise     |
+-----------+------------+-----------------------------------+
6 rows in set (0.00 sec)
```

Update views

```
mysql> UPDATE ActorView
    -> SET
    -> NetWorthInMillions = 250
    -> WHERE
    -> Id =1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM ActorView;
+----+-----------+------------+--------------------+
| Id | FirstName | SecondName | NetWorthInMillions |
+----+-----------+------------+--------------------+
|  1 | Brad      | Pitt       |                250 |
|  2 | Jennifer  | Aniston    |                240 |
|  3 | Angelina  | Jolie      |                100 |
|  4 | Johnny    | Depp       |                200 |
|  5 | Natalie   | Portman    |                 60 |
|  6 | Tom       | Cruise     |                570 |
|  7 | Kylie     | Jenner     |               1000 |
|  8 | Kim       | Kardashian |                370 |
```

WITH CHECK OPTION

```
mysql> SELECT * FROM Actors;
+----+-----------+------------+------------+--------+---------------+--------------------+
| Id | FirstName | SecondName | DoB        | Gender | MaritalStatus | NetWorthInMillions |
+----+-----------+------------+------------+--------+---------------+--------------------+
|  1 | Brad      | Pitt       | 1963-12-18 | Male   | Single        |                250 |
|  2 | Jennifer  | Aniston    | 1969-11-02 | Female | Single        |                240 |
|  3 | Angelina  | Jolie      | 1975-06-04 | Female | Single        |                100 |
|  4 | Johnny    | Depp       | 1963-06-09 | Male   | Single        |                200 |
|  5 | Natalie   | Portman    | 1981-06-09 | Male   | Married       |                 60 |
|  6 | Tom       | Cruise     | 1962-07-03 | Male   | Divorced      |                570 |
|  7 | Kylie     | Jenner     | 1997-08-10 | Female | Married       |               1000 |
|  8 | Kim       | Kardashian | 1980-10-21 | Female | Married       |                370 |
|  9 | Amitabh   | Bachchan   | 1942-10-11 | Male   | Married       |                400 |
| 10 | Shahrukh  | Khan       | 1965-11-02 | Male   | Married       |                600 |
| 11 | priyanka  | Chopra     | 1982-07-18 | Female | Married       |                 28 |
+----+-----------+------------+------------+--------+---------------+--------------------+
11 rows in set (0.00 sec)

mysql>
```

Stored Procedures

```
DELIMITER **
CREATE PROCEDURE ShowActors()
BEGIN
    SELECT * FROM Actors;
END **
DELIMITER ;
```

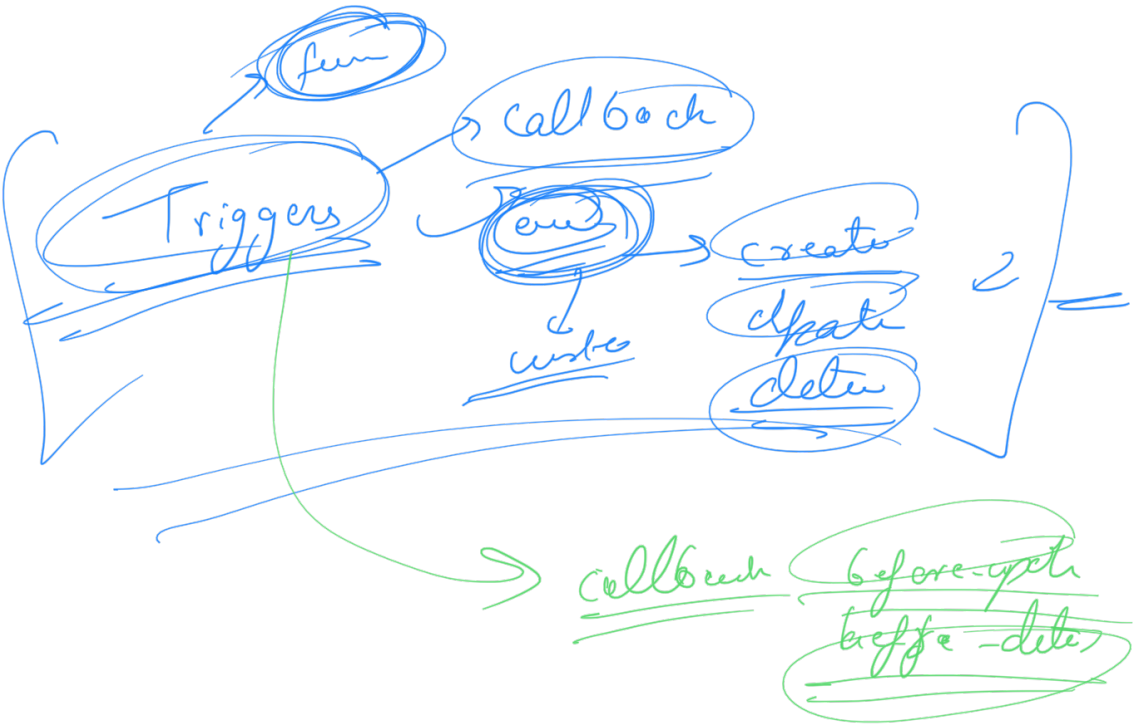
Variables

```
CALL ShowActors();
```

```
DELIMITER **

CREATE PROCEDURE Summary()
BEGIN
    DECLARE TotalM, TotalF INT DEFAULT 0;
    DECLARE AvgNetWorth DEC(6,2) DEFAULT 0.0;

    SELECT COUNT(*) INTO TotalM
    FROM Actors
    WHERE Gender = 'Male';

    SELECT COUNT(*) INTO TotalF
    FROM Actors
    WHERE Gender = 'Female';

    SELECT AVG(NetWorthInMillions)
    INTO AvgNetWorth
    FROM Actors;

    SELECT TotalM, TotalF, AvgNetWorth;
END**

DELIMITER ;
```

```
DELIMITER **

CREATE PROCEDURE GetActorCountByNetWorth (
    IN NetWorth INT,
    OUT ActorCount INT
)
BEGIN
    SELECT COUNT(*)
    INTO ActorCount
    FROM Actors
    WHERE NetWorthInMillions >= NetWorth;
END**

DELIMITER ;
```

We have specified two parameters for our stored procedure, one is input and the other output. To receive the return value, we will pass a session variable.



```
DELIMITER **

CREATE PROCEDURE GenderCountByNetWroth(
    IN NetWorth INT,
    OUT MaleCount INT,
    OUT FemaleCount INT)
BEGIN
        SELECT COUNT(*) INTO MaleCount
        FROM Actors
        WHERE NetWorthInMillions >= NetWorth
              AND Gender = 'Male';

    SELECT COUNT(*) INTO FemaleCount
        FROM Actors
        WHERE NetWorthInMillions >= NetWorth
              AND Gender = 'Female';

END**
DELIMITER ;
```

```
CALL GenderCountByNetWroth(500, @Male, @Female);

SELECT @Male, @Female;
```