DATABASE MANAGEMENT SYSTEMS
BASIC SQL QUERIES

Retrieving Information from a Table

The SELECT statement is used to pull information from a table.

The general form of the statement is:

SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;

what_to_select indicates what you want to see. This can be a list of columns, or * to indicate "all columns."
which_table indicates the table from which you want to retrieve data.
The WHERE clause is optional. If it is present, **conditions_to_satisfy** specifies one or more conditions that rows must satisfy to qualify for retrieval.

Example:

```
SELECT * FROM Customers WHERE Country = 'Mexico';

SELECT * FROM Customers WHERE CustomerID = 1;
```

Operators in The WHERE Clause

Arithmatic Operators:

()	Parentheses
/	Division
*	Multiplication
-	Subtraction
+	Addition
%	Modulo

Comparison Operators

=	equal
<>, !=	not equal
>	Greater Than
<	Less Than
>=	Greater than or equal to
<=	Less than or equal to

Logical operators

AND	BETWEEN	IS NULL
OR	IN	IS NOT NULL
NOT	LIKE	

Few Examples:

```
SELECT * FROM emp;
SELECT Ename FROM emp where sal>50000;
SELECT * FROM emp where sal + bonus < 50000;
SELECT * FROM Customers WHERE Country='Mexico,,;
SELECT * FROM Customers WHERE Country='Germany' AND City='Berlin';
SELECT * FROM Customers WHERE City='Berlin' OR City='München';
SELECT * FROM Customers WHERE NOT Country='Germany';
```

Alias (as) names:

Used to assign names to the columns when they are retrieved from the database table.

Syntax:

Select expr1 [as alias1], expr2 [as alias2] [, …]
From table1 [, table2, …]
[Where condition]

Example:

Select city, ((1.8 + avg_temp) + 32) AS temperature From Temperature

A multiword heading needs to be enclosed in double quotes

Example:

Select city, ((1.8 + avg_temp) + 32) AS “Average Temperature” From Temperature.

Working with NULL values:

Conceptually, NULL means “a missing unknown value” and it is treated somewhat differently from other values.
Null values are no values in the field - so the regular operators can not be used for comparison.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation

IS NULL is used to check if the field contains a null value or not.
IS NOT NULL is used to see if a field is not null

Example

```
Select eid, ename From emp Where bonus IS NULL;

Select eid, ename From emp Where bonus IS NOT NULL;
```

Distinct

The SELECT DISTINCT statement is used to return only distinct (different) values.
Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.
Eliminates all the duplicate entries in the table resulting from the query.

Syntax:

Select [DISTINCT] select_list
From table[, table, …]
[Where expression]
[Order By expression]

Example: SELECT DISTINCT Country FROM Customers;

In Operator:

IN condition checks if the values in a column are present in a given list (set of values).

Syntax:

Select select_list
From table
Where column [not] in (value_list);

Example (Using IN):

Select sid, sname From student Where branch IN („CSE", „IT", „CSD");

Example (not Using IN)

Select sid, sname From student Where branch=„CSE" OR , branch=„IT" OR branch= „CSD";

NOT IN can similarly be used to select rows where values do not match

Select sid, sname From student Where branch NOT IN („CSE", „IT", „CSD");

Between Operator:

Between condition is used to see if the value of a column lies between specified ranges

Syntax:

Select col1, col2,…
From table
Where column [not] between lower_value and upper_value;

Example:

Select eid, emp From Emp Where sal between 50000 and 99999

Alternate Query:

Select eid, emp From Emp Where sal>=50000 and sal<=99999;

Pattern Matching usin LIKE operator:

Like allows a matching of patterns in the column data

Syntax:

Select select_list
From table
Where column [not] like 'pattern' [Escape char]

Wildcards:

- Any Single Character
% (or *) 0 or more characters

A combination of „-„ and „%" along with other characters can be used to represent different patterns.

For test of fixed number of characters multiple dashes can be used

For example „---" will select all 3 letter words from the column

Example: select * from emp where ename like „a%";
select * from emp where ename not like „a%";

Few Examples:

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

LIMIT Clause:

The LIMIT clause is used to specify the number of records to return.

The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance. Hence use limit to control the no. of rows displayed.

Syntax:

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

Example: select * from emp limit 3;

Limit can also be used to skip certain number of columns and then continue to retrieve by limiting to no. of rows specified.

Example: select * from emp limit 3, 2;

Order By Clause:

The ORDER BY keyword is used to sort the result-set in ascending or descending order.
The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.
Multiple levels of sort can be done by specifying multiple columns

Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example:

Display employee names in alphabetical order

```
SELECT ename FROM emp ORDER BY ename;
```

Display the details of employee earning highest to lowest salary

SELECT * FROM emp ORDER BY sal DESC;

Display details of employees department wise in alphabetical order

SELECT * FROM emp ORDER BY dno, ename;

Display employee list department wise in descending order of salary

SELECT * FROM emp ORDER BY dno, sal desc;

DATABASE MANAGEMENT SYSTEMS
SET OPERATORS

Relational databases employ set operators to modify and compare data sets. Simply put, a set operator creates a new set of data that satisfies particular requirements by combining or comparing two or more data sets.

The frequently used set operators in MySQL are: UNION, UNION ALL, INTERSECT, and EXCEPT. With these operators, the output of two or more SELECT statements can be combined or compared.

UNION: To merge the output of two or more SELECT operations into a single result set, use UNION. Each SELECT statement's unique rows are included in the result set.

UNION ALL operator in MySQL is used to combine the result sets of two or more SELECT statements into a single result set. Unlike the UNION operator, which removes duplicate rows, UNION ALL preserves all rows from all SELECT statements.

INTERSECT, on the other hand, returns only the rows that are shared by two SELECT statements' result sets. This operator is useful when comparing two tables to find only the common records between them.

EXCEPT returns those rows from the first SELECT statement that are not present in the second. This operator is analogous to the difference between two sets in mathematics.

Set operators are powerful tools that allow you to manipulate data sets in various ways. As a result, you can increase the efficiency and performance of your database operations by incorporating them properly into your MySQL queries.

Necessary Condition for the Usage of Set Operators

Set operators in MySQL are a key feature that allows you to combine or compare data sets. However, certain prerequisites must be completed before utilizing set operators in your queries.

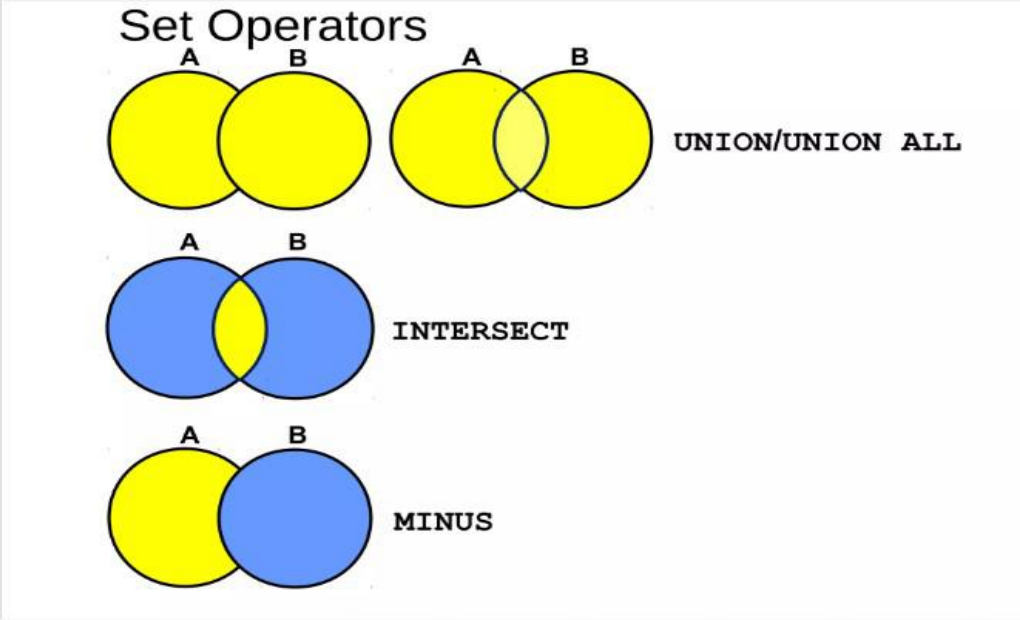
The first requirement is that the SELECT queries should have the same number of columns and be of the same data type. This ensures that the data sets are interoperable and easily integrated or compared.

According to the second criterion, the columns in the SELECT statements must be in the same order. This is significant because the set operators combine or compare rows based on the order of the columns.

Furthermore, the data types of the columns in the SELECT statements must be compatible. A set operator, for example, cannot be used to combine a column of integers with a column of strings.

GENERAL SYNTAX :

```
SELECT column1, column2
FROM table1
UNION/UNION ALL/INTERSECT/EXCEPT
SELECT column1, column2
FROM table2;
```



Let's say we have two tables called employees and customers for illustration as follows:

EMPLOYEES:

id	name	salary
1	John	50000
2	Jane	60000
3	Bob	55000

CUSTOMERS:

id	name	city
1	Alice	Boston
2	Bob	Miami
3	Charlie	Austin

UNION OPERATOR

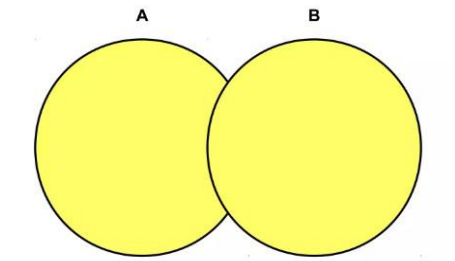
The Union is a binary set operator in DBMS. It is used to combine the result set of two select queries. Thus, It combines two result sets into one. In other words, the result set obtained after union operation is the collection of the result set of both the tables.

But two necessary conditions need to be fulfilled when we use the union command.

These are:

1. Both SELECT statements should have an equal number of fields in the same order.
2. The data types of these fields should either be the same or compatible with each other.

The Union operation can be demonstrated as follows:



The UNION operator returns results from both queries after eliminating duplications.

The syntax for the UNION operation is as follows:

```
SELECT (coloumn_names) from table1 [WHERE condition]
UNION
SELECT (coloumn_names) from table2 [WHERE condition];
```

Example:

```
SELECT name FROM employees
UNION
SELECT name FROM customers;
```

Output:

name
Alice
Bob
Charlie
Jane
John

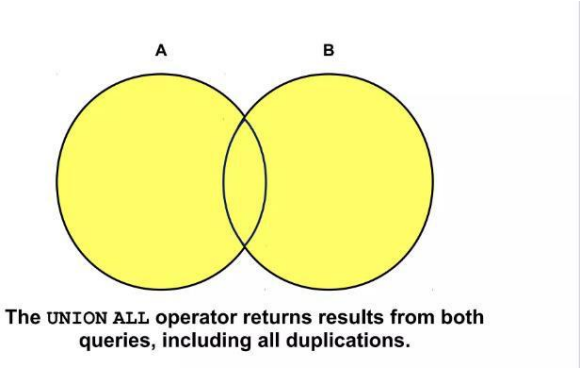
UNION ALL OPERATOR:

The Union operation gives us distinct values. If we want to allow the duplicates in our result set, we'll have to use the **'Union-All'** operation.

Union All operation is also similar to the union operation. The only difference is that it allows duplicate values in the result set.

The syntax for the UNION ALL operation is as follows:

```
SELECT (coloumn_names) from table1 [WHERE condition]
UNION ALL
SELECT (coloumn_names) from table2 [WHERE condition];
```



The UNION ALL operator returns results from both queries, including all duplications.

Example:

```
SELECT name FROM employees
UNION ALL
SELECT name FROM customers;
```

Output:

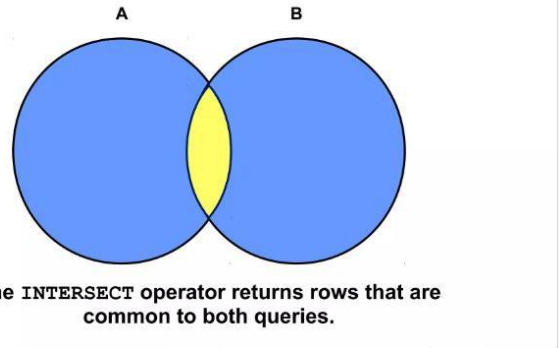
name
John
Jane
Bob
Alice
Bob
Charlie

INTERSECT OPERATOR

Intersect is a binary set operator in DBMS. The intersection operation between two selections returns only the common data sets or rows between them. It should be noted that the intersection operation always returns the distinct rows. The duplicate rows will not be returned by the intersect operator.

Here also, the above conditions of the union and minus are followed, i.e., the number of fields in both the SELECT statements should be the same, with the same data type, and in the same order for the intersection.

The intersection operation can be demonstrated as follows:



The INTERSECT operator returns rows that are common to both queries.

The syntax for the INTERSECT operation is as follows:

```
SELECT (coloumn_names) from table1 [WHERE condition]
INTERSECT
SELECT (coloumn_names) from table2 [WHERE condition];
```

Example:

```
SELECT name FROM employees
INTERSECT
SELECT name FROM customers;
```

Output:

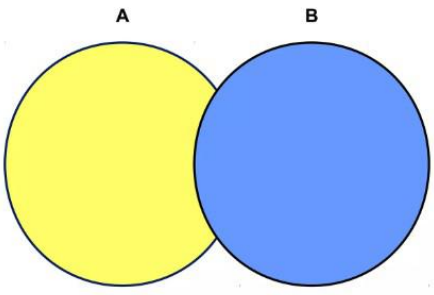
name
Bob

EXCEPT OPERATOR

EXCEPT is a binary set operator in DBMS. The EXCEPT operation between two selections returns the rows that are present in the first selection but not in the second selection. The Minus EXCEPT operator returns only the distinct rows from the first table.

It is a must to follow the above conditions that we've seen in the union, i.e., the number of fields in both the SELECT statements should be the same, with the same data type, and in the same order for the minus operation.

The EXCEPT operation can be demonstrated as follows:



The MINUS operator returns rows in the first query that are not present in the second query.

The syntax for the EXCEPT operation is as follows:

```
SELECT (coloumn_names) from table1 [WHERE condition]
INTERSECT
SELECT (coloumn_names) from table2 [WHERE condition];
```

DATABASE MANAGEMENT SYSTEMS
NESTED QUERIES AND CORRELATED QUERIES

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Syntax

```
SELECT column_name FROM table_name
WHERE column_name expression operator
(SELECT column_name from table_name WHERE condition );
```

The subquery (inner query) execute before the main query (outer query).
The result of the subquery is used by the main query.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

Let us use the employee table for the illustrations of the query

Types of Subqueries:

1 .Single row Subquery – returns zero or one row

2. Multiple row Subquery – returns one or more rows

3. Multi Column Subquery – returns one or more columns

1. Single Row Subquery:

SQL subqueries are most frequently used with the Select statement. Single row subqueries returns one row. Single row subquery uses comparison operators like (=, >, <, >=, <=, <>)

Example

1. Find the name, job and sal of employees whose salary is less than that of an employee with empno=7876.

```
SELECT ENAME, JOB , SAL FROM EMP
WHERE SAL < (SELECT SAL FROM EMP WHERE EMPNO=7876);
```

OUTPUT:

```
mysql> SELECT ENAME, JOB, SAL FROM EMP
    -> WHERE SAL< (SELECT SAL FROM EMP WHERE EMPNO=7876);
+-------+-------+---------+
| ENAME | JOB   | SAL     |
+-------+-------+---------+
| SMITH | CLERK | 800.00 |
| JAMES | CLERK | 950.00 |
+-------+-------+---------+
```

2. Display the employee name who is having the same job as '7369' and earning more than '7876'

```
SQL> SELECT    ename, job
  2  FROM      emp
  3  WHERE     job =
  4                    (SELECT      job
  5                    FROM         emp
  6                    WHERE        empno = 7369)
  7  AND       sal >
  8                    (SELECT      sal
  9                    FROM         emp
  10                   WHERE        empno = 7876);
```

3. Display the details of employees who is earning the least salary

```
SQL> SELECT   ename, job, sal
  2  FROM     emp
  3  WHERE    sal =
  4                  (SELECT      MIN(sal)
  5                  FROM         emp);
```

4. List out the departments whose Minimum salary is more than the least salary of department 20

```
SQL> SELECT       deptno, MIN(sal)
  2  FROM         emp
  3  GROUP BY     deptno
  4  HAVING       MIN(sal) >
  5                        (SELECT    MIN(sal)
  6                       FROM        emp
  7                       WHERE       deptno = 20);
```

MULTI - ROW SUBQUERIES:

Multi row subqueries return set of rows. This type of query uses set comparison operators like (IN, ANY, ALL). We cannot use comparison operators with multi row subqueries.

IN - Equals to any member in the list
ANY - Return rows that match any value on a list
ALL - Return rows that match all the values in a list.

IN OPERATOR:

Find the employees whose salary is same as the minimum salary of employees in the department

SELECT ENAME, SAL FROM EMP
WHERE SAL IN (SELET MIN(SAL) FROM EMP GROUP BY DEPTNO);

```
mysql> SELECT ENAME, SAL FROM EMP
    -> WHERE SAL IN (SELECT MIN(SAL) FROM EMP GROUP BY DEPTNO);
+-------+---------+
| ENAME | SAL     |
+-------+---------+
| SMITH |  800.00 |
| JAMES |  950.00 |
| MILER | 1300.00 |
+-------+---------+
```

ANY OPERATOR:

Display the employees whose salary is more than the maximum salary of the employees in any department.

SELECT ENAME, SAL FROM EMP
WHERE SAL > ANY (SELECT MAX(SAL) FRM EMP GROUP BY DEPTNO);

```
mysql> SELECT ENAME, SAL FROM EMP
    -> WHERE SAL>ANY (SELECT MAX(SAL) FROM EMP
    -> GROUP BY DEPTNO);
+-------+---------+
| ENAME | SAL     |
+-------+---------+
| JONES | 2975.00 |
| SCOTT | 3000.00 |
| KING  | 5000.00 |
| FORD  | 3000.00 |
+-------+---------+
```

ALL OPERATOR:

Display the details of employees who earn salary less than those whose job is ‘MANAGER’

SELECT EMPNO, ENAME, JOB, SAL FROM EMP

WHERE SAL < ALL (SELECT SAL FROM EMP WHERE JOB=”MANAGER”)

AND JOB <> “MANAGER”;

```
mysql> SELECT EMPNO, ENAME, JOB, SAL FROM EMP
    -> WHERE SAL<ALL (SELECT SAL FROM EMP WHERE JOB='MANAGER')
    -> AND JOB <> 'MANAGER';
+-------+--------+----------+---------+
| EMPNO | ENAME  | JOB      | SAL     |
+-------+--------+----------+---------+
|  7369 | SMITH  | CLERK    |  800.00 |
|  7499 | ALLEN  | SALESMAN | 1600.00 |
|  7521 | WARD   | SALESMAN | 1250.00 |
|  7654 | MARTIN | SALESMAN | 1250.00 |
|  7844 | TURNER | SALESMAN | 1500.00 |
|  7876 | ADAMS  | CLERK    | 1100.00 |
|  7900 | JAMES  | CLERK    |  950.00 |
|  7934 | MILER  | CLERK    | 1300.00 |
+-------+--------+----------+---------+
```

MULTIPLE COLUMN SUBQUERY:

A subquery that compares more than one column between the parent query and subquery is called the multiple column subqueries.

Example: List the details of the employees who earn the same salary as of employee no. 7521 with the same job also.

SELECT ENAME, JOB, SAL, EMPNO FROM EMP

WHERE (JOB, SAL) IN (SELECT JOB, SAL FROM EMP WHERE EMPNO=7521);

Here there needs to be a pairwise comparison of the outer query rows with the inner query output.

```
mysql> SELECT ENAME, JOB, SAL, EMPNO FROM EMP
    -> WHERE (JOB,SAL) IN (SELECT JOB, SAL FROM EMP WHERE EMPNO=7521);
+--------+----------+---------+-------+
| ENAME  | JOB      | SAL     | EMPNO |
+--------+----------+---------+-------+
| WARD   | SALESMAN | 1250.00 |  7521 |
| MARTIN | SALESMAN | 1250.00 |  7654 |
+--------+----------+---------+-------+
```

Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

INSERT INTO table_name (column1, column2, column3....)
SELECT * FROM table_name WHERE VALUE OPERATOR
(SELECT COLUMN_NAME FROM TABLE_NAME WHERE condition);

Example

Consider a table EMPLOYEE_DUP similar to EMPLOYEE.

INSERT INTO EMPLOYEE_DUP
SELECT * FROM EMP WHERE SAL IN
(SELECT SAL FROM EMP WHERE DEPTNO=20);

Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

UPDATE table SET column_name = new_value WHERE VALUE OPERATOR (SELECT COLUMN_NAME FROM TABLE_NAME WHERE condition);

Example

UPDATE EMP_DUP SET COMM= SAL * 0.25 WHERE EMPNO IN (SELECT EMPNO FROM EMP WHERE COMM IS NULL AND DEPTNO=20);

Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

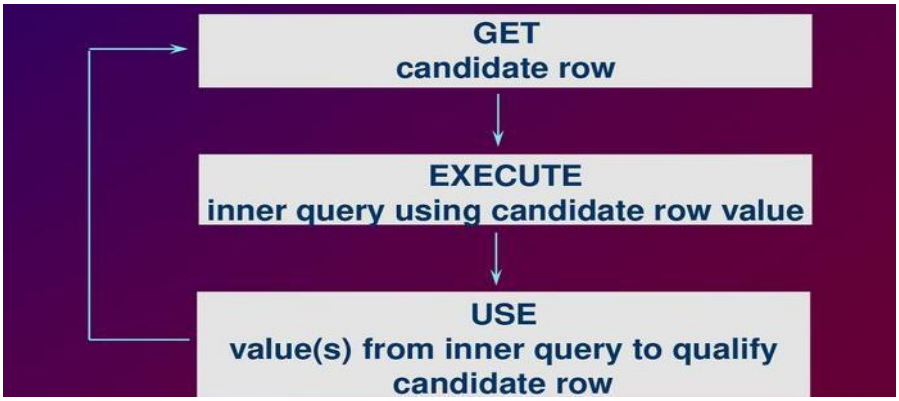
DELETE FROM TABLE_NAME WHERE VALUE OPERATOR (SELECT COLUMN_NAME FROM TABLE_NAME WHERE condition);

Example:

DELETE FROM EMP_DUP WHERE SAL IN (SELECT SAL FROM EMP WHREE SAL>50000);

CORRELATED SUBQUERIES

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

DATABASE MANAGEMENT SYSTEMS
Triggers

A trigger is a procedure which is automatically invoked by the DBMS in response to changes to the database, and is specified by the database administrator (DBA).

A database with a set of associated triggers is generally called an active database.

In MySQL, a trigger is a stored database object that is automatically executed or fired on occurrence of a DML command (also satisfying a specified condition.)

Triggers are used to enforce business rules, validate input data, and maintain an audit trail.

Parts of a Trigger:

A Trigger description contains three parts:

Event-Condition-Action (ECA)

Event is a change to the database which activates the trigger -

Trigger Event - INSERT | UPDATE | DELETE

Trigger Activation Time – BEFORE | AFTER

Condition is a query that is run when the trigger is activated - SQL condition

Action is a procedure that is executed when a trigger is activated due to meeting the specified condition

All data actions performed by the trigger, execute within the same transaction in which the trigger fires.

Cannot contain transaction control statements (COMMIT, SAVEPOINT, ROLLBACK)

Uses of Trigger:

1. **Enforcing Business Rules :**Triggers can ensure that certain business rules are automatically enforced within the database. For example, a trigger can ensure that an employee's salary cannot be decreased. Passwords must be changed every month, insertion not allowed beyond working hours.
2. **Validating Data:** Triggers can be used to validate data before it is inserted or updated in the database. For example, ensuring that an email address has a valid format.
3. **Maintaining Audit Trails:** Triggers can be used to keep an audit trail of changes made to important data. This is useful for tracking who changed what data and when.
4. **Synchronizing Tables:** Triggers can be used to synchronize data between tables. For example, if you have a master table and several dependent tables, you can use triggers to update dependent tables whenever the master table is modified.
5. **Preventing Invalid Transactions**: Triggers can be used to prevent certain transactions that could lead to invalid or inconsistent data. For example, preventing deletion of records that are referenced by other records.

6. **Automatic Calculations :** Triggers can be used to perform automatic calculations and updates. For example, updating the total amount in an order whenever an order line item is added or updated.
7. **Enforcing Referential Integrity:** Triggers can enforce referential integrity rules, such as cascading updates or deletes, beyond what foreign keys alone can do.
8. **Data Transformation :** Triggers can be used to transform data before it is stored. For example, automatically converting all text to uppercase.

Types of Trigger

In MySQL, triggers are always defined at the row level, meaning they execute once for each row affected by the triggering event.

We can define the maximum six types of actions or events in the form of triggers:

Before Insert: It is activated before the insertion of data into the table.

After Insert: It is activated after the insertion of data into the table.

Before Update: It is activated before the update of data in the table.

After Update: It is activated after the update of the data in the table.

Before Delete: It is activated before the data is removed from the table.

After Delete: It is activated after the deletion of data from the table.

Trigger Syntax:

```
CREATE TRIGGER trigger_name
  (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
     ON table_name FOR EACH ROW
     BEGIN
    --variable declarations
    --trigger code
    END;
```

NOTE: When a trigger code contains multiple statements, we can change the delimiter from the default ; to some other character.

Example-1:

Let us create a trigger that would make a log entry whenever a new record is inserted into the EMP table along with the timestamp.

1. CREATE A TABLE EMP_LOG as below

```
CREATE TABLE EMP_LOG (
LOG_ID INT AUTO_INCREMENT PRIMARY KEY,
EMP_ID INT, MSG VARCHAR(50),
CHANGE_TIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

2. Create the trigger as below

```
CREATE TRIGGER emp_insert_log
after INSERT ON emp
FOR EACH ROW
  INSERT INTO EMP_LOG (EMP_ID, MSG)
  VALUES (NEW.EMPNO, "ROW INSERTED AT");
```

3. TESTING: - run the following commands and check the output
    a. Insert record into emp table
    b. Select * from emp_log (check the new record inserted automatically)

Example-2:

Create a trigger for making the log entry whenever the salary is updated recording the empno, old and new salary

1. CREATE A TABLE SAL_UPDATE_LOG as below

```
CREATE TABLE SAL_UPDATE_LOG (
LOG_ID INT AUTO_INCREMENT PRIMARY KEY,
EMP_ID INT,
OLD_SALARY FLOAT(7,2), NEW_SALARY FLOAT(7,2),
CHANGE_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

2. CREATE THE TRIGGER

```
CREATE TRIGGE emp_log
after UPDATE ON emp
FOR EACH ROW
  INSERT INTO SAL_UPDATE_LOG (EMP_ID, OLD_SALARY,
NEW_SALARY)
  VALUES (NEW.EMPNO, OLD.SAL, NEW.SAL);
```

3. TESTING:
    a. Update salary for an employee in emp table
    b. Select * from sal_update_log

Example – 3:

Create a trigger to be executed before insert into emp which ensures the minimum salary of the new employees is 30000

```
DELIMITER //
Create Trigger before_insert_empsalary
BEFORE INSERT ON emp FOR EACH ROW
BEGIN
IF NEW.sal < 30000 THEN SET NEW.sal = 30000;
END IF;
```

```
END //
DELIMITER ;
```

TESTING:

Insert a new record into emp table with salary less than 30000
Select * from emp
Ensure that the salary is set to 30000 in the newly entered row.

Triggers that raise the error:

In MySQL, you can create a trigger that will raise an error when a certain condition is met by using the SIGNAL SQL statement. The SIGNAL statement allows you to set an error condition, which effectively raises an error.

SIGNAL SQLSTATE '45000': Raises an error with a custom SQLSTATE value of '45000', which is a generic state indicating an error condition.

SET MESSAGE_TEXT = ‘xxxxxxxxx': Sets the error message that will be returned.

Example:

Let us assume we should not allow insertion into emp table beyond office hours [9am-6pm]

```
DELIMITER //

CREATE TRIGGER control_access
BEFORE INSERT ON emp
FOR EACH ROW
BEGIN
   DECLARE HR INT;
   SET HR = HOUR(CURTIME());
   IF HR < 9 AND HR >18 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'INSERTION IS PROHIBITED AT THIS TIME';
   END IF;
END //

DELIMITER ;
```

Listing the available Triggers:

List all the available triggers on a database

SHOW TRIGGERS;

```
----------------+
| Trigger              | Event  | Table | Statement                                              | Timing |
Created                                                                          |       sql_mode
| Definer       | character_set_client | collation_connection | Database Collation |
--+--------------------+
| before_insert_empsalary | INSERT | emp | BEGIN
IF NEW.sal < 30000 THEN SET NEW.sal = 30000;
END IF;
END          |          BEFORE          |          2024-07-11          14:45:16.49          |
ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZER
O_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION            |
root@localhost | cp850              | cp850_general_ci     | utf8mb4_0900_ai_ci |
+------------------------+--------+-------+--
```

Deleting a Trigger:

We can drop/delete/remove a trigger in MySQL using the DROP TRIGGER statement.

DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;

Example:

DROP TRIGGER before_insert_empsalary;

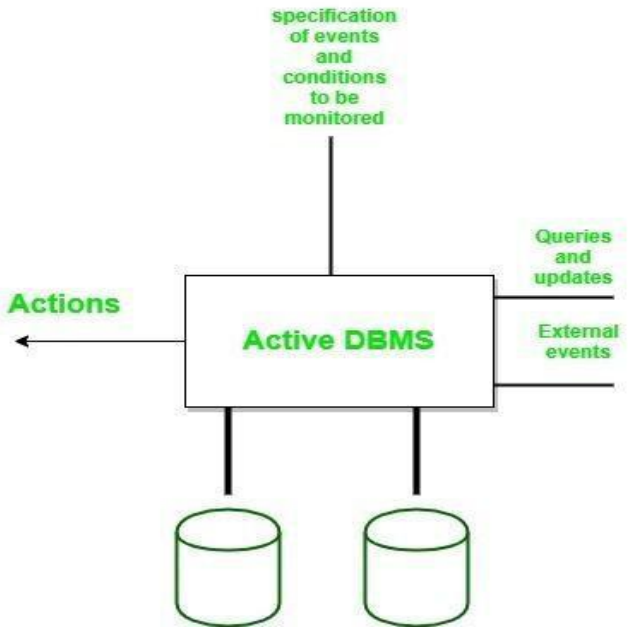
DATABASE MANAGEMENT SYSTEMS
Miscellaneous Topics

Active Databases:

Active Database is a database consisting of set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database is activated or not, prior to executing the statement.

If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement.

In such situation, DBMS processes each of the trigger randomly. The execution of an action part of a trigger may either activate other triggers or the same trigger that Initialized this action. Such types of trigger that activates itself is called as 'recursive trigger'. The DBMS executes such chains of trigger in some pre-defined manner but it affects the concept of understanding.



Features of Active Database:

1. It possess all the concepts of a conventional database i.e. data modelling facilities, query language etc.

2. It supports all the functions of a traditional database like data definition, data manipulation, storage management etc.
3. It supports definition and management of ECA (Event–Condition–Action) (ECA) rules.
4. It detects event occurrence.
5. It must be able to evaluate conditions and to execute actions.
6. It means that it has to implement rule execution.

Advantages:

- Enhances traditional database functionalities with powerful rule processing capabilities.
- Enable a uniform and centralized description of the business rules relevant to the information system.
- Avoids redundancy of checking and repair operations.
- Suitable platform for building large and efficient knowledge base and expert systems.

MySQL Create User :

The MySQL user is a record in the USER table of the MySQL server that contains the login information, account privileges, and the host information for MySQL account. It is essential to create a user in MySQL for accessing and managing the databases.

The MySQL Create User statement allows us to create a new user account in the database server.

When the MySQL server installation completes, it has a ROOT user account only to access and manage the databases.

We can create non-root users using the following syntax:

CREATE USER [IF NOT EXISTS] username@hostname IDENTIFIED BY 'password';

Example:

```
mysql> create user savy@localhost identified by 'savram123';
```

When you create a user that already exists, it gives an error. But if you use, IF NOT EXISTS clause, the statement gives a warning for each named user that already exists instead of an error message.

```
mysql> CREATE USER IF NOT EXISTS ram@localhost IDENTIFIED BY 'esha123';
```

Grant Privileges to the MySQL New User

MySQL server provides multiple types of privileges to a new user account. Some of the most commonly used privileges are given below:

ALL PRIVILEGES: It permits all privileges to a new user account.

CREATE: It enables the user account to create databases and tables.

DROP: It enables the user account to drop databases and tables.

DELETE: It enables the user account to delete rows from a specific table.

INSERT: It enables the user account to insert rows into a specific table.

SELECT: It enables the user account to read a database.

UPDATE: It enables the user account to update table rows.

Examples:

mysql> GRANT ALL PRIVILEGES ON * . * TO savy@localhost;

mysql> GRANT CREATE, SELECT, INSERT ON * . * TO ram@localhost;

Note: *.* indicates all databases and all tables ---- db.* - means all tables on the database db

To see the existing privileges for the user, execute the following command.

mysql> SHOW GRANTS for username;

Drop User:

The MySQL Drop User statement allows to remove one or more user accounts and their privileges from the database server. If the account does not exist in the database server, it gives an error.

DROP USER 'account_name';

Example: **DROP** USER ram@localhost;

DATABASE MANAGEMENT SYSTEMS
Stored Procedures

A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database.

It is a subroutine or a subprogram in the regular computing language. A procedure always contains a name, parameter lists, and SQL statements.

We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc.

Stored Procedure Features

- Stored Procedure increases the performance of the applications. Once stored procedures are created, they are compiled and stored in the database.
- Stored procedure reduces the traffic between application and database server. Because the application has to send only the stored procedure's name and parameters instead of sending multiple SQL statements.
- Stored procedures are reusable and transparent to any applications.
- A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

Syntax :

The following syntax is used for creating a stored procedure in MySQL. It can return one or more value through parameters or sometimes may not return at all. By default, a procedure is associated with our current database. But we can also create it into another database from the current database by specifying the name as database_name.procedure_name.

```
DELIMITER //
CREATE PROCEDURE procedure_name [ [IN | OUT | INOUT]
parameter_name datatype [, parameter_name datatype]) ]
BEGIN
   Declaration_section
   Executable_section
END //
DELIMITER ;
```

Parameter Name	Descriptions
procedure_name	It represents the name of the stored procedure.
Parameter	presents the number of parameters. It can be one or more than one.
Declaration_section	It represents the declarations of all variables.
Executable_section	It represents the code for the function execution.

Different types of parameters used:

1. IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

2. OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

3. INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

Calling the Procedure:

We can use the CALL statement to call a stored procedure. This statement returns the values to its caller through its parameters (IN, OUT, or INOUT). The following syntax is used to call the stored procedure in MySQL:

CALL procedure_name (parameter(s))

Examples:

1. Procedures without Parameters

Suppose we want to display the list and count of employees who are earning commission.

```
DELIMITER //
CREATE PROCEDURE comm_details()
BEGIN
   SELECT * FROM emp WHERE comm IS NOT NULL;
   SELECT COUNT(comm) AS 'Commission Earners' FROM emp;
END //
DELIMITER ;
```

Calling the procedure :

```
> CALL comm_details();
```

2. Procedures with IN Parameters

Suppose we want to display the list of employees who are from a specified department.

```
DELIMITER //
CREATE PROCEDURE dept_employees (IN dno int)
BEGIN
        SELECT * from EMP where DEPTNO=dno;
END //
DELIMITER ;
```

Calling the procedure :

```
> CALL dept_employees(20);
```

3. Procedures with OUT Parameters

Suppose we want to display the highest salary paid.

```
DELIMITER //
CREATE PROCEDURE display_max_sal(OUT highestsal INT)
BEGIN
   SELECT MAX(sal) INTO highestsal FROM emp;
END //
DELIMITER ;
```

NOTE: When we call the procedure, the OUT parameter tells the database systems that its value goes out from the procedures. Now, we will pass its value to a session variable @M in the CALL statement as follows:

Calling the procedure :

```
> CALL display_max_sal (@X);
> SELECT @X;
```

4. Procedures with IN & OUT Parameters

Suppose we want to display the highest salary paid in the specified department

```
DELIMITER //
CREATE PROCEDURE display_deptmax_sal(OUT highestsal INT, IN dno INT)
BEGIN
   SELECT MAX(sal) INTO highestsal FROM emp WHERE deptno=dno;
END //
DELIMITER ;
```

Calling the procedure :

```
> CALL display_deptmax_sal (@M, 10);
```

6. Procedures with IN & OUT Parameters

Suppose we want to display the salary earned by a selected employee

```
DELIMITER &&
CREATE PROCEDURE get_emp_sal (INOUT data INT)
BEGIN
   SELECT sal INTO data FROM emp WHERE empno = data;
END &&
DELIMITER ;
```

Calling the procedure :

```
> SET @M = 7900
> CALL get_emp_sal(@M);
> SELECT @M;
```

Displaying the list of All procedures

When we have several procedures in the MySQL server, we can list all procedure stored on the current MySQL server as follows:

SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search_condition]

Example:

SHOW PROCEDURE STATUS WHERE DB='IT2B';

Deleting Stored Procedures:

MySQL also allows a command to drop the procedure. When the procedure is dropped, it is removed from the database server also. The following statement is used to drop a stored procedure in MySQL:

DROP PROCEDURE [IF EXISTS] procedure_name;

Example:

DROP PROCEDURE comm_details;

DATABASE MANAGEMENT SYSTEMS
CONCEPT OF JOINS

Databases usually store large amounts of data. To analyze that data efficiently, analysts and DBAs have a constant need to extract records from two or more tables based on certain conditions.

A relational database consists of multiple related tables linking together using common columns, which are known as foreign key columns. Because of this, the data in each table is incomplete from the business perspective.

JOINS are used to retrieve data from multiple tables in a single query by establishing logical relationship between them. This operation is made feasible with a common key value between tables.

JOIN clauses are used in the SELECT, UPDATE, and DELETE statements.

Different types of JOINs in MySQL
MySQL JOIN type defines the way two tables are related in a query.

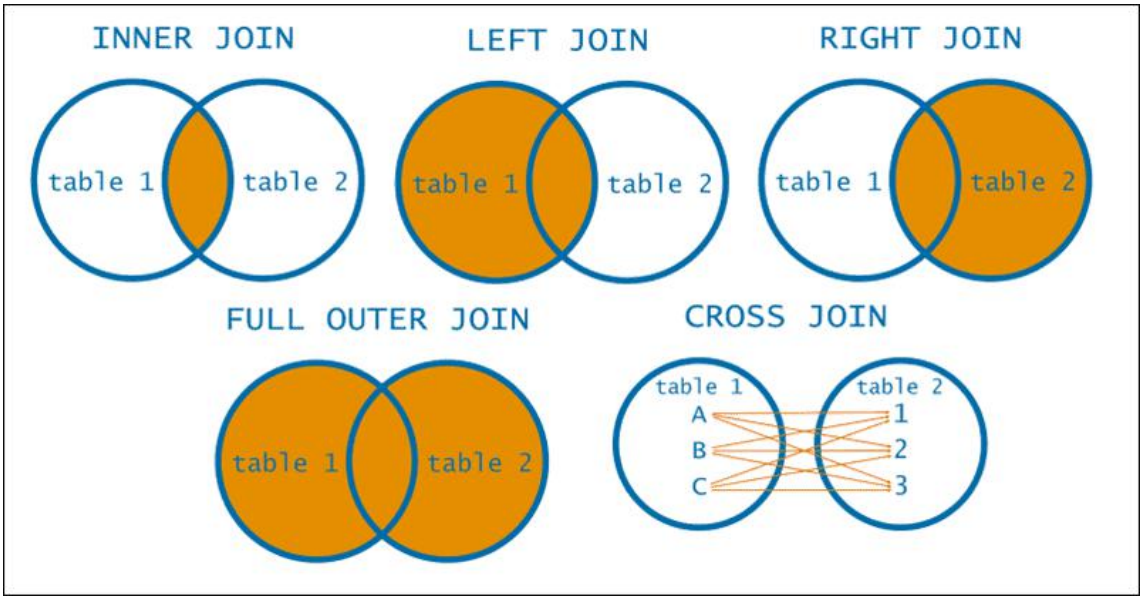
INNER JOIN: Returns records that have matching values in both tables

LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table

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

CROSS JOIN: Returns Cartesian product of left and right table

SELF JOIN: join that is used to join a table with itself



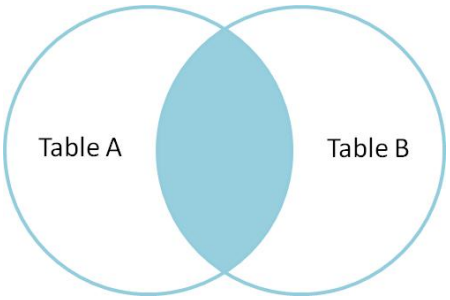
Let us use these SUPPLIER and PARTS tables for illustration

```
mysql> SELECT * FROM SUPPLIER;
+-----+-------+--------+-----------+
| SNO | SNAME | STATUS | CITY      |
+-----+-------+--------+-----------+
| S1  | Smith |     20 | London    |
| S2  | Jones |     10 | Paris     |
| S3  | Blake |     30 | Paris     |
| S4  | Clark |     20 | London    |
| S5  | Adams |     30 | Athens    |
| S6  | Pavan |     24 | Hyderabad |
+-----+-------+--------+-----------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM PARTS;
+-----+------+-------+-------+-------+--------+------+
| PNO | SNO  | PNAME | COLOR | WEIGH | CITY   | cost |
+-----+------+-------+-------+-------+--------+------+
| P1  | S1   | Nut   | Red   |    12 | London |   50 |
| P2  | S1   | Bolt  | Green |    17 | Paris  |   70 |
| P3  | S2   | Screw | Blue  |    17 | Rome   |   80 |
| P4  | S3   | Screw | Red   |    14 | London |   80 |
| P5  | S2   | Cam   | Blue  |    12 | Paris  |   90 |
| P6  | S3   | Cog   | Red   |    19 | London |   68 |
+-----+------+-------+-------+-------+--------+------+
6 rows in set (0.00 sec)
```

INNER JOIN:

The **INNER JOIN** will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.



SYNTAX:

SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 INNER JOIN table2
ON table1.matching_column = table2.matching_column;

Here,

table1: First table.
table2: Second table
matching_column: Column common to both the tables.

Example:

Fetch data of the supplier with the parts name sold by the Supplier.

```
select A.SNAME, B.PNAME
from supplier A INNER JOIN parts B
ON A.SNO=B.SNO;
```

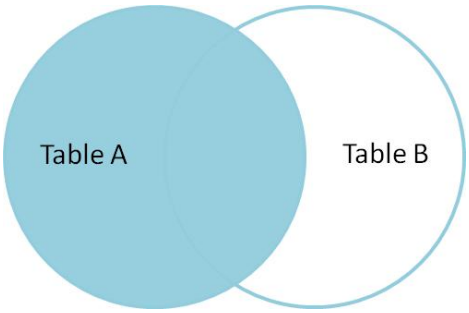
OUTPUT:

```
mysql> select A.SNAME, B.PNAME from supplier A INNER JOIN parts B ON A.SNO=B.SNO;
+-------+-------+
| SNAME | PNAME |
+-------+-------+
| Smith | Nut   |
| Smith | Bolt  |
| Jones | Screw |
| Blake | Screw |
| Jones | Cam   |
| Blake | Cog   |
+-------+-------+
6 rows in set (0.00 sec)
```

SQL LEFT (OUTER) JOIN

In contrast to INNER JOINs, OUTER JOINs return not only matching rows but non-matching ones as well. In case there are non-matching rows in a joined table, the NULL values will be shown for them.

LEFT JOIN clause allows retrieving all rows from the left table(Table A), along with those rows from the right table(Table B) for which the join condition is satisfied. Wherever any record of the left table does not match with the right table NULL is displayed for right-side columns.



SYNTAX:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example:

Fetch data of all the suppliers whether they sold any part or not..

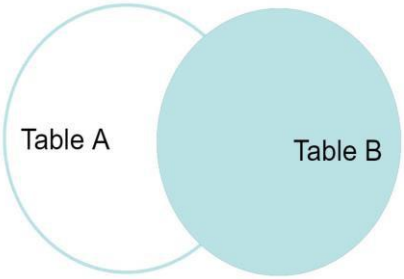
```
select A.SNAME, A.CITY, B.PNAME, B.COLOR
from supplier A LEFT JOIN parts B
ON A.SNO=B.SNO;
```

OUTPUT:

```
mysql>  select a.SNAME, a.CITY, b.PNAME, b.COLOR from supplier a LEFT JOIN parts b ON a.SNO=b.SNO;
+-------+-----------+-------+-------+
| SNAME | CITY      | PNAME | COLOR |
+-------+-----------+-------+-------+
| Smith | London    | Bolt  | Green |
| Smith | London    | Nut   | Red   |
| Jones | Paris     | Cam   | Blue  |
| Jones | Paris     | Screw | Blue  |
| Blake | Paris     | Cog   | Red   |
| Blake | Paris     | Screw | Red   |
| Clark | London    | NULL  | NULL  |
| Adams | Athens    | NULL  | NULL  |
| Pavan | Hyderabad | NULL  | NULL  |
+-------+-----------+-------+-------+
9 rows in set (0.00 sec)
```

SQL RIGHT (OUTER) JOIN

The RIGHT OUTER JOIN returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. It is very similar to LEFT JOIN For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.



SYNTAX:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1 RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

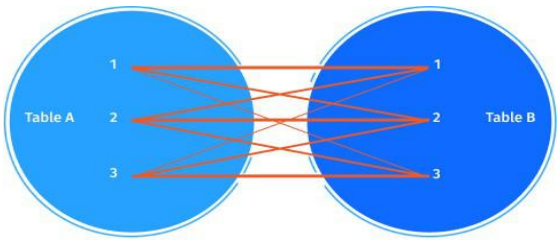
Example:

Fetch data of parts irrespective of whether they were sold by any supplier or not.

```
select A.SNAME, A.CITY, B.PNAME, B.COLOR
from supplier A RIGHT JOIN parts B
ON A.SNO=B.SNO;
```

SQL CROSS JOIN

MySQL CROSS JOIN, also known as a Cartesian join, retrieves all combinations of rows from each table. In this type of JOIN, the result set is returned by multiplying each row of table 1 with all rows in table 2 if no additional condition is introduced.



SYNTAX:

```
SELECT columns
FROM table1  CROSS JOIN  table2
```

NOTE: if there are N records in Table 1 and M records in Table 2, then the result set will contain N x M records.

```
mysql> select * from supplier cross join parts;
+-----+-------+--------+-----------+-----+-----+-------+-------+-------+--------+------+
| SNO | SNAME | STATUS | CITY      | PNO | SNO | PNAME | COLOR | WEIGH | CITY   | cost |
+-----+-------+--------+-----------+-----+-----+-------+-------+-------+--------+------+
| S6  | Pavan |     24 | Hyderabad | P1  | S1  | Nut   | Red   |    12 | London |   50 |
| S5  | Adams |     30 | Athens    | P1  | S1  | Nut   | Red   |    12 | London |   50 |
| S4  | Clark |     20 | London    | P1  | S1  | Nut   | Red   |    12 | London |   50 |
| S3  | Blake |     30 | Paris     | P1  | S1  | Nut   | Red   |    12 | London |   50 |
| S2  | Jones |     10 | Paris     | P1  | S1  | Nut   | Red   |    12 | London |   50 |
| S1  | Smith |     20 | London    | P1  | S1  | Nut   | Red   |    12 | London |   50 |
| S6  | Pavan |     24 | Hyderabad | P2  | S1  | Bolt  | Green |    17 | Paris  |   70 |
| S5  | Adams |     30 | Athens    | P2  | S1  | Bolt  | Green |    17 | Paris  |   70 |
| S4  | Clark |     20 | London    | P2  | S1  | Bolt  | Green |    17 | Paris  |   70 |
| S3  | Blake |     30 | Paris     | P2  | S1  | Bolt  | Green |    17 | Paris  |   70 |
| S2  | Jones |     10 | Paris     | P2  | S1  | Bolt  | Green |    17 | Paris  |   70 |
| S1  | Smith |     20 | London    | P2  | S1  | Bolt  | Green |    17 | Paris  |   70 |
| S6  | Pavan |     24 | Hyderabad | P3  | S2  | Screw | Blue  |    17 | Rome   |   80 |
| S5  | Adams |     30 | Athens    | P3  | S2  | Screw | Blue  |    17 | Rome   |   80 |
| S4  | Clark |     20 | London    | P3  | S2  | Screw | Blue  |    17 | Rome   |   80 |
| S3  | Blake |     30 | Paris     | P3  | S2  | Screw | Blue  |    17 | Rome   |   80 |
| S2  | Jones |     10 | Paris     | P3  | S2  | Screw | Blue  |    17 | Rome   |   80 |
| S1  | Smith |     20 | London    | P3  | S2  | Screw | Blue  |    17 | Rome   |   80 |
| S6  | Pavan |     24 | Hyderabad | P4  | S3  | Screw | Red   |    14 | London |   80 |
| S5  | Adams |     30 | Athens    | P4  | S3  | Screw | Red   |    14 | London |   80 |
| S4  | Clark |     20 | London    | P4  | S3  | Screw | Red   |    14 | London |   80 |
| S3  | Blake |     30 | Paris     | P4  | S3  | Screw | Red   |    14 | London |   80 |
| S2  | Jones |     10 | Paris     | P4  | S3  | Screw | Red   |    14 | London |   80 |
| S1  | Smith |     20 | London    | P4  | S3  | Screw | Red   |    14 | London |   80 |
| S6  | Pavan |     24 | Hyderabad | P5  | S2  | Cam   | Blue  |    12 | Paris  |   90 |
| S5  | Adams |     30 | Athens    | P5  | S2  | Cam   | Blue  |    12 | Paris  |   90 |
| S4  | Clark |     20 | London    | P5  | S2  | Cam   | Blue  |    12 | Paris  |   90 |
| S3  | Blake |     30 | Paris     | P5  | S2  | Cam   | Blue  |    12 | Paris  |   90 |
| S2  | Jones |     10 | Paris     | P5  | S2  | Cam   | Blue  |    12 | Paris  |   90 |
| S1  | Smith |     20 | London    | P5  | S2  | Cam   | Blue  |    12 | Paris  |   90 |
| S6  | Pavan |     24 | Hyderabad | P6  | S3  | Cog   | Red   |    19 | London |   68 |
| S5  | Adams |     30 | Athens    | P6  | S3  | Cog   | Red   |    19 | London |   68 |
| S4  | Clark |     20 | London    | P6  | S3  | Cog   | Red   |    19 | London |   68 |
| S3  | Blake |     30 | Paris     | P6  | S3  | Cog   | Red   |    19 | London |   68 |
| S2  | Jones |     10 | Paris     | P6  | S3  | Cog   | Red   |    19 | London |   68 |
| S1  | Smith |     20 | London    | P6  | S3  | Cog   | Red   |    19 | London |   68 |
+-----+-------+--------+-----------+-----+-----+-------+-------+-------+--------+------+
36 rows in set (0.00 sec)
```

SELF JOIN

SELF JOIN implies the joining of a table to itself. It states that each row of a table is joined with itself and with every other row of the same table. When you want to extract the hierarchical data or compare rows within the same table, then self-join is the best choice.

SYNTAX:

SELECT columns

FROM table AS alias1 JOIN table AS alias2

ON alias1.column = alias2.column;

Example:

Determine the names of employees, who earn more than their managers.

SELECT M.ENAME "EMPLOYEE", M.SAL, E.ENAME "MANAGER", E.SAL
FROM EMP M, EMP E
WHERE E.EMPNO=M.MGR AND M.SAL>E.SAL;

```
mysql> SELECT M.ENAME "EMPLOYEE", M.SAL, E.ENAME "MANAGER", E.SAL FROM EMP M, EMP E WHERE E.EMPNO=M.MGR AND M.SAL>E.SAL;
+----------+----------+---------+---------+
| EMPLOYEE | SAL      | MANAGER | SAL     |
+----------+----------+---------+---------+
| SCOTT    | 30000.00 | JONES   | 2975.00 |
| TURNER   | 45500.00 | BLAKE   | 2850.00 |
| JAMES    | 25950.00 | BLAKE   | 2850.00 |
| FORD     | 63000.00 | JONES   | 2975.00 |
| MILLER   | 75300.00 | CLARK   | 2450.00 |
+----------+----------+---------+---------+
5 rows in set (0.00 sec)
```