**SQL Question Bank**

**1.** What is Relational Database management system (RDBMS)?

**Ans:** A relational database management system (RDBMS) refers to the software used to store, manage, & query data. Data is stored in tables & can be linked to other datasets based on shared information, hence the name “relational”.

2. How does a RDBMS differ from a DBMS?

Ans:

|  |  |  |
| --- | --- | --- |
| **No.** | **RDBMS** | **DBMS** |
| **1.** | An RDBMS stores data in a relational table with rows & columns | A DBMS stores data as a file |
| **2.** | An RDBMS provides access to multiple users (including client – server side interaction) | A DBMS only supports single user |

3. What are some of most popular RDBMS?

Ans: Some of the most popular RDBMS are:

* Oracle Database
* MySQL
* Microsoft SQL Server
* PostgreSQL
* IBM DB2
* SQLite

4. What is the role of SQL (Structured Query language)?

Ans: SQL is the programming language used in an RDBMS.

5. What is a query?

Ans: A query is a request for data or information from a database. There are 2 main types of SQL queries:

* A **select query** is a query that groups data from a table for analytical purposes.
* An **action query** is a query that changes the contents of the database based on specified criteria

6. What is a subquery?

Ans: A subquery is a query that is embedded within another statement that requires multiple steps.

The subquery provides the enclosing query with additional information needed to execute a task, such as when the completion of one query depends first on the results of another.

7. SQL datatype

* SQL datatype is used to define the values that a column can contain.
* Every column is required to have a name & data type in the database table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SQL Datatype** | | | | |
| **Binary datatype** | **Approximate Numeric datatype** | **Extract Numeric datatype** | **String datatype** | **Date datatype** |
| binary | float | int | char | date |
| varbinary | real | smallint | varchar | time |
| image |  | bit | text | timestamp |
|  |  | decimal |  |  |
|  |  | numeric |  |  |

1. **Binary datatypes**

|  |  |  |
| --- | --- | --- |
| **No.** | **Data type** | **Description** |
| **1.** | binary | It has a max length of 8000 bytes. It contains fixed – length binary data |
| **2.** | varbinary | It has a max length of 8000 bytes. It contains variable – length binary data |
| **3.** | image | It has a max length of 2.147 Gbs. It contains variable – length binary data |

1. **Approximate Numeric Datatype**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Data type** | **Range** | **Description** |
| **1.** | float | -1.79E+308 to 1.79E+308 | It is used to specify a floating – point value e.g., 6.2, 2.9 etc. |
| **2.** | real | -3.40E+38 to 3.40E+38 | It specifies a single precision floating point number |

1. **Exact Numeric Datatype**

|  |  |  |
| --- | --- | --- |
| **No.** | **Data type** | **Description** |
| **1.** | int | It is used to specify an integer value. |
| **2.** | smallint | It is used to specify small integer value. |
| **3.** | bit | It has the number of bits to store. |
| **4.** | decimal | It specifies a numeric value that can have a decimal number. |
| **5.** | numeric | It is used to specify a numeric value. |

1. **Character String Datatype**

|  |  |  |
| --- | --- | --- |
| **No.** | **Data type** | **Description** |
| **1.** | char | It has a max length of 8000 characters. It contains fixed-length non-unicode characters. |
| **2.** | varchar | It has a max length of 8000 characters. It contains variable-length non-unicode characters. |
| **3.** | text | It has a max length of 2,147,483,647 characters. It contains variable-length non-unicode characters. |

1. **Date & Time Datatype**

|  |  |  |
| --- | --- | --- |
| **No.** | **Data type** | **Description** |
| **1.** | date | It is used to store the year, month, & days value. |
| **2.** | time | It is used to store the hour, minute, & second values. |
| **3.** | timestamp | It stores the year, month, day, hour, minute, & the second value. |

8. SQL commands

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DDL** | **DML** | **DCL** | **TCL** | **DQL** |
| Data Definition Language | Data Manipulation Language | Data Control Language | Transaction Control Language | Data Query Language |
|  | | | | |
| Create | Insert | Grant | Commit | Select |
| Alter | Update | Revoke | Rollback |  |
| Drop | Delete |  | Savepoint |  |
| Truncate |  |  |  |  |

**DDL** (Data definition language)

* DDL changes the structure of the table like creating a table, deleting a table, altering a table etc.
* All the commands of DDL are auto – committed i.e., it permanently save all the changes in the database.
* Commands:

1. **CREATE**: used to create a new table in the database

**CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]);**

e.g., CREATE TABLE EMPLOYEE (Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

1. **ALTER**: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

* To add a new column in the table

**ALTER TABLE table\_name ADD column\_name COLUMN-definition;**

e.g., ALTER TABLE STU\_DETAILS ADD (ADDRESS VARCHAR2(20));

* To modify existing column in the table

**ALTER TABLE table\_name MODIFY (column\_definitions....);**

e.g., ALTER TABLE STU\_DETAILS MODIFY (NAME VARCHAR2(20));

1. **DROP**: It is used to delete both the structure and record stored in the table.

**DROP TABLE table\_name;**

e.g., DROP TABLE EMPLOYEE;

1. **TRUNCATE**: It is used to delete all the rows from the table & free the space containing the table.

**TRUNCATE TABLE table\_name;**

e.g., TRUNCATE TABLE EMPLOYEE;

**DML** (Data Manipulation language)

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto – committed i.e., it can’t permanently save all the changes in the database. They can be rollback.
* Commands:

1. **INSERT**: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

|  |
| --- |
| **INSERT INTO TABLE\_NAME**  **(col1, col2, col3,.... col N)**  **VALUES (value1, value2, value3, .... valueN);**  OR  **INSERT INTO TABLE\_NAME**  **VALUES (value1, value2, value3, .... valueN);** |

e.g., INSERT INTO BOOK (Author, Subject) VALUES ("Shivam", "DBMS");

1. **UPDATE**: This command is used to update or modify the value of a column in the table.

**UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]**

e.g., UPDATE STUDENT SET User\_Name = 'Shivam’ WHERE Student\_Id = '3'

1. **DELETE**: It is used to remove one or more row from a table

**DELETE FROM table\_name [WHERE condition];**

e.g., DELETE FROM STUDENT WHERE Author="Shivam";

**DCL** (Data Control language)

* DCL commands are used to grant & take back authority from any database user.
* Commands

1. Grant: It is used to give user access privileges to a database.

e.g., GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

1. Revoke: It is used to take back permissions from the user.

e.g., REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

**TCL** (Transaction Control language)

* TCL commands can only use with DML commands like INSERT, DELETE or UPDATE only.
* These operations are automatically committed in the database that’s why they can’t be used while creating tables or dropping them.
* Commands:

1. COMMIT: This command is used to save all the transactions to the database.

Syntax: **COMMIT;**

e.g.,

|  |
| --- |
| DELETE FROM CUSTOMERS WHERE AGE = 25;  COMMIT; |

1. ROLLBACK: This command is used to undo transactions that have not already been saved to the database.

Syntax: **ROLLBACK;**

e.g.,

|  |
| --- |
| DELETE FROM CUSTOMERS WHERE AGE = 25;  ROLLBACK; |

1. SAVEPOINT: This command is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax: **SAVEPOINT savepoint\_name;**

**DQL** (Data Query language)

* It is used to fetch the data from the database.
* It uses only one command:

1. SELECT: It is used to select the attribute based on the condition describe by WHERE clause.

**SELECT expressions FROM TABLES WHERE conditions;**

e.g., SELECT emp\_name FROM employee WHERE age > 20;

**9. SQL Operators**

* 3 types of SQL operators:

1. Arithmetic operator
2. Comparison operator
3. Logical operator

SQL Arithmetic operator: Suppose a = 20, b = 10

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Operator | Description | Example |
| 1. | + | It adds the value of both operands. | a+b will give 30 |
| 2. | - | It is used to subtract the right – hand operator from the left – hand operand. |  |
| 3. | \* | It is used to multiply the value of both operands. | a\*b will give 200 |
| 4. | / | Quotient when the left-hand operand divided by the right-hand operator | a/b will give 2 |
| 5. | % | Remainder when left-hand operand divided by right-hand operand | a%b will give 0 |

SQL Comparison Operators: Suppose a = 20, b = 10

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Operator | Description | Example |
| 1. | =, !=, <> | It checks if 2 operand values are equal or not | (a=b) is not true |
| 2. | >, <, >=, <=, !<, !> | Greater than, Less than, Greater than equal, Less than equal | (a>b), (a>=b), (a!<b) is true |

SQL Operators

|  |  |  |
| --- | --- | --- |
| No. | Operator | Description |
| 1. | ALL | It compares a value to all values in another value set. |
| 2. | AND | It allows the existence of multiple conditions in an SQL statement |
| 3. | ANY | It compares the values in the list according to the condition |
| 4. | BETWEEN | It is used to search the values that are within a set of values |
| 5. | IN | It compares a value to that specified list value. |
| 6. | NOT | It reverses the meaning of any logical operator |
| 7. | OR | It combines multiple conditions in SQL statements. |
| 8. | EXISTS | It is used to search for the presence of a row in a specified table. |
| 9. | LIKE | It compares a value to similar values using wildcard operator (%value%) |

**10. SQL Views**

* Views in SQL are considered as a virtual table. A view also contains rows & columns.
* To create a view, we can select the fields from one or more tables present in the database.
* Syntax:

**CREATE VIEW** view\_name **AS** (**SELECT** column1, column2..... **FROM** table\_name **WHERE** condition);

e.g., **CREATE VIEW** DetailsView **AS** (**SELECT** NAME, ADDRESS **FROM** Student\_Details **WHERE** STU\_ID < 4);

**Creating view from multiple table**:

**CREATE VIEW** MarksView **AS** (**SELECT** Student\_Detail.NAME, Student\_Detail.ADDRESS, Student\_Marks.MARKS **FROM** Student\_Detail, Student\_Mark **WHERE** Student\_Detail.NAME = Student\_Marks.NAME);

* **Syntax for deleting view:** DROP VIEW view\_name;

**11. SQL Index**

* Indexes are special lookup tables. It is used to retrieve data from the database very fast.
* An index is used to speed up select queries and where clauses. Indexes can be created or dropped without affecting the data.
* An index in a database is just like an index in the back of a book.
* For example: When you reference all pages in a book, you first have to refer the index, which alphabetically lists all the topics & then referred to one or more specific page numbers.

1. Create Index statement (This allows duplicate value)

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

e.g., **CREATE INDEX** idx\_name **ON** Persons (LastName, FirstName);

Unique index statement (This doesn’t allow duplicate value)

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

1. Drop index statement: **DROP INDEX** index\_name;

**12. SQL Sub Query**

* A Subquery is a query within another SQL query & embedded within the WHERE clause.
* The outer query is known as the main query & the inner query is known as the subquery.

Important Rule:

* A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
* We can use subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, <=, >=, IN, BETWEEN etc.
* Subqueries are on the right side of the comparison operator.
* A subquery is enclosed in parenthesis.
* **In the subquery, ORDER BY command can’t be used. But GROUP BY command can be used to perform the same function as ORDER BY command.**

1. Subqueries with the SELECT statement

**SELECT** column\_name **FROM** table\_name

**WHERE** column\_name **expression operator** (**SELECT** column\_name **from** table\_name **WHERE** ...);

e.g., **SELECT** \* **FROM** EMPLOYEE **WHERE** ID **IN** (**SELECT** ID **FROM** EMPLOYEE **WHERE** SALARY > 4500);

1. Subqueries with the INSERT statement

**INSERT** **INTO** table\_name (column1, column2, column3....) **SELECT** \* **FROM** table\_name **WHERE** VALUE OPERATOR

e.g., **INSERT** **INTO** EMPLOYEE\_BKP **SELECT** \* **FROM** EMPLOYEE **WHERE** ID IN (**SELECT** ID **FROM** EMPLOYEE);

1. Subqueries with the UPDATE statement

**UPDATE** table **SET** column\_name = new\_value **WHERE** VALUE OPERATOR (**SELECT** COLUMN\_NAME **FROM** TABLE\_NAME **WHERE** condition);

e.g., **UPDATE** EMPLOYEE **SET** SALARY = SALARY \* 0.25 **WHERE** AGE IN (**SELECT** AGE **FROM** CUSTOMERS\_BKP

**WHERE** AGE >= 29);

1. Subqueries with the DELETE statement

**DELETE** **FROM** TABLE\_NAME **WHERE** VALUE OPERATOR (**SELECT** COLUMN\_NAME **FROM** TABLE\_NAME **WHERE** condition);

e.g., **DELETE** **FROM** EMPLOYEE **WHERE** AGE IN (**SELECT** AGE **FROM** EMPLOYEE\_BKP **WHERE** AGE >= 29);

**13. SQL Clauses**

* 3 Clauses are there in SQL

1. GROUP BY clause
2. HAVING clause
3. ORDER BY clause

**GROUP BY clause**

* SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
* The GROUP BY statement follows the WHERE clause in a SELECT statement & precedes the ORDER BY clause.
* The GROUP BY statement is used with aggregation function.

**SELECT** column **FROM** table\_name **WHERE** conditions **GROUP BY** column **ORDER BY** column

e.g.,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PRODUCT** | **COMPANY** | **QTY** | **RATE** | **COST** |
| Item1 | Com1 | 2 | 10 | 20 |
| Item2 | Com2 | 3 | 25 | 75 |
| Item3 | Com1 | 2 | 30 | 60 |
| Item4 | Com3 | 5 | 10 | 50 |
| Item5 | Com2 | 2 | 20 | 40 |
| Item6 | Cpm1 | 3 | 25 | 75 |
| Item7 | Com1 | 5 | 30 | 150 |
| Item8 | Com1 | 3 | 10 | 30 |
| Item9 | Com2 | 2 | 25 | 50 |
| Item10 | Com3 | 4 | 30 | 120 |

**SELECT** COMPANY, COUNT (\*) **FROM** PRODUCT\_MAST **GROUP BY** COMPANY;

Output:

**Com1 5**

**Com2 3**

**Com3 2**

**HAVING clause**

* HAVING clause is used to specify a search condition for a group or an aggregate.
* HAVING is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

**SELECT** column1, column2 **FROM** table\_name **WHERE** conditions

**GROUP BY** column1, column2

**HAVING** conditions

**ORDER BY** column1, column2;

e.g., **SELECT** COMPANY, COUNT (\*) **FROM** PRODUCT\_MAST

**GROUP BY** COMPANY

**HAVING** COUNT (\*)>2;

Output:

**Com1 5**

**Com2 3**

**ORDER BY**

* The ORDER BY clause sorts the result – set in ascending or descending order.
* It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

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

**ORDER BY** column1, column2... ASC|DESC;

e.g.,

CUSTOMER

|  |  |  |
| --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **ADDRESS** |
| 12 | Kathrin | US |
| 23 | David | Bangkok |
| 34 | Alina | Dubai |
| 45 | John | UK |
| 56 | Harry | US |

**SELECT** \* **FROM** CUSTOMER **ORDER BY** NAME;

Output:

|  |  |  |
| --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **ADDRESS** |
| 34 | Alina | Dubai |
| 23 | David | Bangkok |
| 56 | Harry | US |
| 45 | John | UK |
| 12 | Kathrin | US |

**SELECT** \* **FROM** CUSTOMER **ORDER BY** NAME **DESC**;

Output:

|  |  |  |
| --- | --- | --- |
| **CUSTOMER\_ID** | **NAME** | **ADDRESS** |
| 12 | Kathrin | US |
| 45 | John | UK |
| 56 | Harry | US |
| 23 | David | Bangkok |
| 34 | Alina | Dubai |

**14. SQL Aggregate Functions**

* SQL Aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
* It is also used to summarize the data.
* Types of SQL Aggregation function

1. COUNT
2. SUM
3. AVG
4. MAX
5. MIN

**COUNT**

* COUNT function is used to count the no. of rows in a database table. It can work on both numeric & non numeric datatypes.
* COUNT function uses the COUNT(\*) that returns the count of all the rows in a specified table. COUNT(\*) considers duplicate & Null.

**COUNT (\*)** or **COUNT ([ALL|DISTINCT] expression)**

e.g.,

**PRODUCT\_MAST**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PRODUCT** | **COMPANY** | **QTY** | **RATE** | **COST** |
| Item1 | Com1 | 2 | 10 | 20 |
| Item2 | Com2 | 3 | 25 | 75 |
| Item3 | Com1 | 2 | 30 | 60 |
| Item4 | Com3 | 5 | 10 | 50 |
| Item5 | Com2 | 2 | 20 | 40 |
| Item6 | Com1 | 3 | 25 | 75 |
| Item7 | Com1 | 5 | 30 | 150 |
| Item8 | Com1 | 3 | 10 | 30 |
| Item9 | Com2 | 2 | 25 | 50 |
| Item10 | Com3 | 4 | 30 | 120 |

e.g.,1: **SELECT** COUNT (\*) **FROM** PRODUCT\_MAST; // 10

e.g.,2: **SELECT** COUNT (\*) **FROM** PRODUCT\_MAST **WHERE** RATE>=20; // 7

e.g.,3: **SELECT** COMPANY, COUNT (\*) **FROM** PRODUCT\_MAST **GROUP BY** COMPANY;

Output:

**Com1 5**

**Com2 3**

**Com3 2**

e.g.,4: **SELECT** COMPANY, COUNT (\*) **FROM** PRODUCT\_MAST **GROUP BY** COMPANY **HAVING** COUNT (\*) > 2;

Output:

**Com1 5**

**Com2 3**

**SUM**

* Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

**SUM ()** or **SUM ([ALL|DISTINCT] expression)**

e.g.,1: **SELECT** **SUM**(COST) **FROM** PRODUCT\_MAST; // 670

e.g.,2: **SELECT** **SUM**(COST) **FROM** PRODUCT\_MAST **WHERE** QTY>3; // 320

e.g.,3: **SELECT** **SUM**(COST) **FROM** PRODUCT\_MAST **WHERE** QTY>3 **GROUP BY** COMPANY;

e.g.,4: **SELECT** COMPANY, **SUM**(COST) **FROM** PRODUCT\_MAST **GROUP BY** COMPANY **HAVING** **SUM**(COST)>=170;

Output:

**Com1 335**

**Com3 170**

**AVG. Function**

* The AVG function is used to calculate the average value of the numeric type. AVG function return the average of all non – Null values.

**AVG ()** or **AVG ([ALL|DISTINCT] expression)**

e.g.,1: **SELECT** AVG(COST) **FROM** PRODUCT\_MAST; // 67.00

**MAX Function**

* MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

**MAX ()** or **MAX ([ALL|DISTINCT] expression)**

e.g., **SELECT** MAX(RATE) **FROM** PRODUCT\_MAST; // 30

**MIN Function**

* MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

**MIN ()** or **MIN ([ALL|DISTINCT] expression)**

e.g., **SELECT** MIN(RATE) **FROM** PRODUCT\_MAST; // 10

**15. SQL Join**

* In case of SQL, JOIN means “to combine two or more tables”.
* In SQL, JOIN clause is used to combine the records from 2 or more tables in a database
* Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN

**EMPLOYEE**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **CITY** | **SALARY** | **AGE** |
| 1 | Angelina | Chicago | 200000 | 30 |
| 2 | Robert | Austin | 300000 | 26 |
| 3 | Christian | Denver | 100000 | 42 |
| 4 | Kristen | Washington | 500000 | 29 |
| 5 | Russell | Los angels | 200000 | 36 |
| 6 | Marry | Canada | 600000 | 48 |

**PROJECT**:

|  |  |  |
| --- | --- | --- |
| **PROJECT\_NO** | **EMP\_ID** | **DEPARTMENT** |
| 101 | 1 | Testing |
| 102 | 2 | Development |
| 103 | 3 | Designing |
| 104 | 4 | Development |

**INNER JOIN**

* In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied. It returns the combination of all rows from both the tables where the condition satisfies.

**SELECT** table1.column1, table1.column2, table2.column1,.... **FROM** table1

**INNER JOIN** table2 **ON** table1.matching\_column = table2.matching\_column;

e.g., **SELECT** EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT **FROM** EMPLOYEE

**INNER JOIN** PROJECT **ON** PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;

|  |  |
| --- | --- |
| **EMP\_NAME** | **DEPARTMENT** |
| Angelina | Testing |
| Robert | Development |
| Christian | Designing |
| Kristen | Development |

**LEFT JOIN**

* The SQL Left join returns all the values from left table & the matching values from the right table. If there is no matching join value, it will return NULL.

**SELECT** table1.column1, table1.column2, table2.column1,.... **FROM** table1

**LEFT JOIN** table2 **ON** table1.matching\_column = table2.matching\_column;

e.g., **SELECT** EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT **FROM** EMPLOYEE

**LEFT JOIN** PROJECT **ON** PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;

|  |  |
| --- | --- |
| **EMP\_NAME** | **DEPARTMENT** |
| Angelina | Testing |
| Robert | Development |
| Christian | Designing |
| Kristen | Development |
| Russell | NULL |
| Marry | NULL |

**RIGHT JOIN**

* In SQL, RIGHT JOIN returns all the values from the rows of right table & the matched values from the left table. If there is no matching in both tables, it will return NULL.

**SELECT** table1.column1, table1.column2, table2.column1,.... **FROM** table1

**RIGHT JOIN** table2 **ON** table1.matching\_column = table2.matching\_column;

e.g., **SELECT** EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT **FROM** EMPLOYEE

**RIGHT JOIN** PROJECT **ON** PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;

|  |  |
| --- | --- |
| **EMP\_NAME** | **DEPARTMENT** |
| Angelina | Testing |
| Robert | Development |
| Christian | Designing |
| Kristen | Development |

**FULL JOIN**

* In SQL, FULL JOIN is the result of a combination of both left & right outer join. Join tables have all the records from both tables.
* It puts NULL on the place of matches not found.

**SELECT** table1.column1, table1.column2, table2.column1,.... **FROM** table1

**FULL JOIN** table2 **ON** table1.matching\_column = table2.matching\_column;

e.g., **SELECT** EMPLOYEE.EMP\_NAME, PROJECT.DEPARTMENT **FROM** EMPLOYEE

**FULL JOIN** PROJECT **ON** PROJECT.EMP\_ID = EMPLOYEE.EMP\_ID;

|  |  |
| --- | --- |
| **EMP\_NAME** | **DEPARTMENT** |
| Angelina | Testing |
| Robert | Development |
| Christian | Designing |
| Kristen | Development |
| Russell | NULL |
| Marry | NULL |

16. SQL Set Operation

* The SQL set operation is used to combine the two or more SQL SELECT statements.
* Types of Set operation

1. Union
2. UnionAll
3. Intersect
4. Minus

**First:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**Second:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**Union**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**UnionAll**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**Intersect**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |

**Minus**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |