**RDBMS**

A **relational database** refers to a database that stores data in a structured format, using rows and columns. This makes it easy to locate and access specific values within the database. It is "**relational**" because the values within each table are related to each other. Tables may also be related to other tables.

* **Three Types of Relationships in an ERD Diagram**

There are three types of relationships that can exist between two entities.  
An entity-relationship (ER) diagram is created based on these three types, which are listed below:

one-to-one relationship: In relational database design, a one-to-one (1:1) relationship exists when zero or one instance of entity A can be associated with zero or one instance of entity B, and zero or one instance of entity B can be associated with zero or one instance of entity A. (abbreviated 1:1)

one-to-many relationship: (abbreviated 1:N) In relational database design, a one-to-many (1:N) relationship exists when, for one instance of entity A, there exists zero, one, or many instances of entity B; but for one instance of entity B, there exists zero or one instance of entity A.

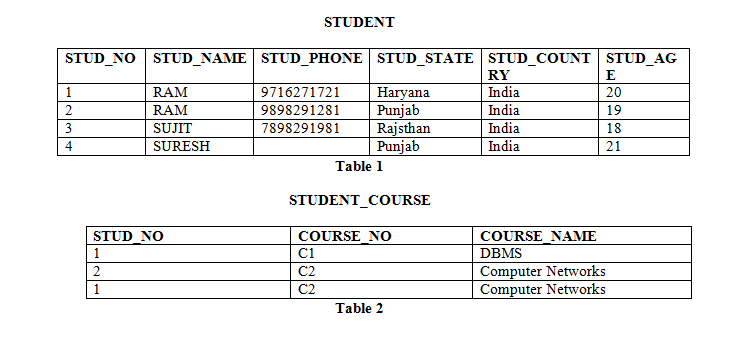
many-to-many relationship: In relational database design, a many-to-many (M:N) relationship exists when, for one instance of entity A, there exists zero, one, or many instances of entity B; and for one instance of entity B, there exists zero, one, or many instances of entity A. (abbreviated M:N)

Following are simple examples of each:

|  |  |
| --- | --- |
| 1:1 relationship | In a traditional American marriage, a man can be married to only one woman; a woman can be married to only one man. |
| 1:N relationship | A child has exactly one biological father; a father can have many biological children. |
| M:N relationship | A student can enroll in many classes; a class can have many enrolled students. |

In the business world, one-to-one relationships are few and far between. One-to-many and many-to-many relationships, on the other hand, are common. However, as will be explained later, many-to-many relationships are not permitted in a relational database and must be converted into one-to-many relationships. Relational databases are comprised almost entirely of tables in one-to-many relationships.

* **Types of Keys in Relational Model**



**Candidate Key:** The minimal set of attributes which can uniquely identify a tuple [row] is known as candidate key. For Example, STUD\_NO in STUDENT relation. 

* The value of Candidate Key is unique and non-null for every tuple.
* There can be more than one candidate key in a relation. For Example, STUD\_NO is candidate key for relation STUDENT.
* The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD\_NO, COURSE\_NO} is a composite candidate key for relation STUDENT\_COURSE.

**Super Key:**The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD\_NO, (STUD\_NO, STUD\_NAME) etc. 

* Adding zero or more attributes to candidate key generates super key.
* A candidate key is a super key but vice versa is not true.

**Primary Key:** There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD\_NO, as well as STUD\_PHONE both, are candidate keys for relation STUDENT but STUD\_NO can be chosen as the primary key (only one out of many candidate keys).

**Alternate Key:**The candidate key other than the primary key is called an alternate key. For Example, STUD\_NO, as well as STUD\_PHONE both, are candidate keys for relation STUDENT but STUD\_PHONE will be alternate key (only one out of many candidate keys).

**Foreign Key:** A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.

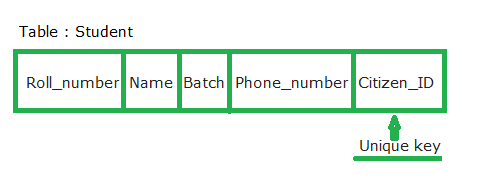
The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

It may be worth noting that unlike, Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint.

For Example, STUD\_NO in STUDENT\_COURSE relation is not unique. It has been repeated for the first and third tuple. However, the STUD\_NO in STUDENT relation is a primary key and it needs to be always unique and it cannot be null.

**Unique key:** constraints also identifies an individual tuple uniquely in a relation or table. A table can have more than one unique key unlike primary key. Unique key constraints can accept only one NULL value for column. Unique constraints are also referenced by the foreign key of another table. It can be used when someone wants to enforce unique constraints on a column and a group of columns which is not a primary key.

For better understanding of unique key we take Student table with Roll\_number, Name, Batch, Phone\_number and Citizen\_ID attributes.



Roll number attribute is already assigned with the primary key and Citizen\_ID can have unique constraints where each entry in a Citizen\_ID column should be unique because each citizen of a country must have his or her Unique identification number like Aadhaar Number. But if student is migrated to another country in that case, he or she would not have any Citizen\_ID and the entry could have a NULL value as only one NULL is allowed in the unique constraint.

**Key Differences Between Primary key and Unique key:**

1. Primary key will not accept NULL values whereas Unique key can accept one NULL value.
2. A table can have only primary key whereas there can be multiple unique key on a table.
3. A Clustered index automatically created when a primary key is defined whereas Unique key generates the non-clustered index.

* **Database Schema**

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them.

A **schema** is a collection of database objects like tables, triggers, stored procedures, etc. A schema is connected with a user which is known as the schema owner. Database may have one or more schema.



A database schema can be divided broadly into two categories −

* **Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
* **Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.
* **Schema diagram [ER Model to Relational Model] and their mapping**

**--pending**

* **Relational operation**

**--pending**

* **SQL**

SQL is a standard language for accessing and manipulating databases.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Every table is broken up into smaller entities called fields. A field is a column in a table that is designed to maintain specific information about every record in the table.

A record, also called a row, is each individual entry that exists in a table. A record is a horizontal entity in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

MySQL is an open-source relational database management system that uses SQL commands to perform specific functions/operations in a database.

SQL is a query language, whereas MySQL is a relational database that uses SQL to query a database.

* **Introduction to SQL Keywords**

In SQL, the keywords are the reserved words that are used to perform various operations in the database. There are many keywords in SQL and as SQL is case insensitive, it does not matter if we use for example SELECT or select.

List of SQL Keywords

The SQL keywords can be used as explained in the below examples for various operations.

#### 1. CREATE

The CREATE Keyword is used to create a database, table, views, and index. We can create the table CUSTOMER as below.

CREATE TABLE CUSTOMER (CUST\_ID INT PRIMARY KEY, NAME VARCHAR(50), STATE VARCHAR(20));

#### 2. PRIMARY KEY

This keyword uniquely identifies each of the records.

A Database in SQL can be created with the usage of CREATE DATABASE statement as below:

CREATE DATABASE DATABASE\_NAME;

A View in SQL can be created by using CREATE VIEW as below:

CREATE VIEW VIEW\_NAME AS  
SELECT COLUMN1, COLUMN2, COLUMN3...  
FROM TABLE\_NAME WHERE [CONDITION];

#### 3. INSERT

The [INSERT Keyword is used](https://www.educba.com/insert-in-oracle/) to insert the rows of data to a table. We can insert the below rows to the already created CUSTOMER table by using the queries as below.

INSERT INTO CUSTOMER VALUES (121,'Rajesh','Maharashtra');  
INSERT INTO CUSTOMER VALUES(256,'Leela','Punjab');  
INSERT INTO CUSTOMER VALUES(908,'Priya','Jharkhand');  
INSERT INTO CUSTOMER VALUES(787,'Rohit','UP');

The above statements will insert the rows to the table “CUSTOMER”. We can see the result by using a simple SELECT statement below

SELECT \* FROM CUSTOMER;

#### 4. SELECT

This keyword is used to select the data from the database or table. The ‘\*’ is used in the select statement to select all the columns in a table.

SELECT NAME FROM CUSTOMER;

The result of the above query will display the column NAME from the CUSTOMER table as below

#### 5. FROM

The keyword is used to indicate the table from which the data is selected or deleted.

#### 6. ALTER

The Keyword ALTER is used to modify the columns in tables. The ALTER COLUMN statement modifies the data type of a column and the ALTER TABLE modifies the columns by adding or deleting them.

We can modify the columns of the CUSTOMER table as below by adding a new column “AGE”.

ALTER TABLE CUSTOMER ADD AGE INT;  
SELECT \* FROM CUSTOMER;

#### 7. ADD

This is used to add a column to the existing table.

#### 8. DISTINCT

The keyword DISTINCT is used to select distinct values. We can use SELECT DISTINCT to select only the distinct values from a table.

SELECT DISTINCT(STATE) FROM CUSTOMER;

#### 9. UPDATE

This keyword is used in an SQL statement to update the existing rows in a table.

UPDATE CUSTOMER SET STATE ='Rajasthan' WHERE CUST\_ID= 121;

#### 10. SET

This Keyword is used to specify the column or values to be updated.

#### 11. DELETE

This is used to delete the existing rows from a table.

DELETE FROM CUSTOMER WHERE NAME='Rajesh';

#### 12. TRUNCATE

This is used to delete the data in a table, but it does not delete the structure of the table.

#### 13. AS

TRUNCATE TABLE CUSTOMER;

The Keyword AS is used as an alias to rename the column or table.

SELECT CUST\_ID AS CUSTOMER\_ID, NAME AS CUSTOMER\_NAME FROM CUSTOMER;

#### 14. ORDER BY

This is used to sort the result in descending or ascending order. This sorts the result by default in ascending order.

#### 15. ASC

This keyword is used for sorting the data returned by the SQL query in ascending order.

SELECT \* FROM CUSTOMER ORDER BY NAME ASC;

The above query will select all the columns from the CUSTOMER table and sorts the data by the NAME column in ascending order.

#### 16. DESC

This keyword is to sort the result set in descending order.

SELECT \* FROM CUSTOMER ORDER BY CUST\_ID DESC;

The above query will sort all the selected fields of the table with the descending order of CUST\_ID.

#### 17. BETWEEN

This keyword is used to select values within a given range. The below query uses the BETWEEN keyword to select the CUST\_ID and NAME within a given range of values for the CUST\_ID.

SELECT CUST\_ID, NAME FROM CUSTOMER WHERE CUST\_ID BETWEEN 100 AND 500;

#### 18. WHERE

This keyword is used to filter the result set so that only the values satisfying the condition are included.

SELECT \* FROM CUSTOMER WHERE STATE ='Punjab';

#### 19. AND

This keyword is used along with the WHERE clause to select the rows for which both the conditions are true.

SELECT \* FROM CUSTOMER WHERE STATE ='Punjab' AND CUST\_ID= 256;

#### 20. OR

This is used with the WHERE clause to include the rows in the result set in case of either condition is true.

The below SQL statement will select the fields from the CUSTOMER table if the state is Punjab or UP.

SELECT \* FROM CUSTOMER WHERE STATE='Punjab' OR STATE='UP';

#### 21. NOT

The keyword NOT is used with a WHERE clause to include the rows in the result set where a condition is not true.

We can use the NOT keyword in the below query to not include the rows from the state Punjab as below.

SELECT \* FROM CUSTOMER WHERE NOT STATE = 'Punjab';

#### 22. LIMIT

This keyword retrieves the records from the table in order to limit them based on the limit value.

SELECT \* FROM CUSTOMER LIMIT 3;

#### 23. IS NULL

The keyword IS NULL is used to check for NULL values.

The below query will show all the records for which the AGE column has NULL values.

SELECT \* FROM CUSTOMER WHERE AGE IS NULL;

IS NOT NULL

This is used to search the NOT NULL values.

SELECT \* FROM CUSTOMER WHERE STATE IS NOT NULL;

#### 24. DROP

The DROP keyword can be used to delete a database, table, view, column, index, etc.

#### 25. DROP COLUMN

We can delete an existing column in a table by using a DROP COLUMN along with an ALTER statement. Let us delete the column AGE by using the below query.

ALTER TABLE CUSTOMER DROP COLUMN AGE;

#### 26. DROP DATABASE

A database in SQL can be deleted by using the DROP DATABASE statement.

DROP DATABASE DATABASE\_NAME;

#### 27. DROP TABLE

A table in SQL can be deleted by using a DROP TABLE statement.

DROP TABLE TABLE\_NAME;

We can delete the table CUSTOMER by using the DROP TABLE keyword as below.

But we need to be careful while using DROP TABLE as it will remove the table definition along with all the data and indexes etc.

#### 28. GROUP BY

This is used along with the aggregate functions like COUNT, MAX, MIN, AVG, SUM, etc. and groups the result set. The below query will group the CUST\_ID according to the various states.

SELECT COUNT(CUST\_ID),STATE FROM CUSTOMER GROUP BY STATE;

The result shows the count of different CUST\_ID grouped by states.

#### 29. HAVING

This keyword is used with aggregate functions and GROUP BY instead of the WHERE clause to filter the values of a result set.

SELECT COUNT(CUST\_ID),STATE FROM CUSTOMER GROUP BY STATE HAVING COUNT(CUST\_ID)>=2;

The above query will filter the result set by displaying only those values which satisfy the condition given in the [HAVING clause](https://www.educba.com/oracle-having-clause/).

The above result set shows the values for which the count of the customer ids is more than 2.

#### 30. IN

The IN keyword is used within a WHERE clause to specify more than 1 value or we can say that it can be used instead of the usage of multiple OR keyword in a query.

The below query will select the records for the states Maharashtra, Punjab and UP by the use of the IN keyword.

SELECT \* FROM CUSTOMER WHERE STATE IN ('Maharashtra','Punjab','UP');

#### 31. JOIN

The keyword JOIN is used to combine the rows between two or more tables with related columns among the tables. The JOIN can be INNER, LEFT, RIGHT, OUTER JOIN, etc.

Lets us take another table ‘CUST\_ORDER’ as an example.

#### 31. JOIN

The keyword JOIN is used to combine the rows between two or more tables with related columns among the tables. The JOIN can be INNER, LEFT, RIGHT, OUTER JOIN, etc.

Lets us take another table ‘CUST\_ORDER’ as an example.

We can perform an inner join of the CUSTOMER and CUST\_ORDER tables as below

SELECT CUSTOMER.NAME, CUSTOMER.STATE, CUST\_ORDER.ITEM\_DES  
FROM CUSTOMER INNER JOIN CUST\_ORDER  
ON CUSTOMER.CUST\_ID =CUST\_ORDER.ID;

The above query will join the two tables CUSTOMER and CUST\_ORDER on the columns CUST\_ID and ID and display only the values which are present in both the tables.

#### 32. UNION

The UNION keyword is used to combine the distinct values of two or more select statements.

SELECT CUST\_ID FROM CUSTOMER UNION SELECT ID FROM CUST\_ORDER;

If these 2 tables have duplicate values then UNION will show only 1 in result.

#### 33. UNION ALL

This keyword combines two or more select statements but allows duplicate values.

SELECT CUST\_ID FROM CUSTOMER UNION ALL SELECT ID FROM CUST\_ORDER;

#### 34. EXISTS

The keyword EXISTS checks if a certain record exists in a sub-query.

SELECT NAME FROM CUSTOMER WHERE EXISTS (SELECT ITEM\_DES FROM CUST\_ORDER WHERE CUST\_ID = ID);

It returns true or false.

#### 35. LIKE

This keyword is used to search along with a WHERE clause for a particular pattern. Wildcard % is used to search for a pattern.

In the below query let us search for a pattern ‘ya’ which occurs in the column ‘NAME’.

SELECT NAME FROM CUSTOMER WHERE NAME LIKE '%ya';

#### 36. CASE

This keyword is used to display different output according to different conditions.

SELECT CUST\_ID, NAME,  
CASE WHEN STATE = 'Punjab' THEN "State is Punjab"  
ELSE "State is NOT Punjab"  
END AS Output  
FROM CUSTOMER;

* **SQL Join (Inner, Left, Right and Full Joins)**

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN

Consider the two tables below:

**Student**

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/table1-3.png)

**StudentCourse**

[](https://media.geeksforgeeks.org/wp-content/uploads/table5.png)

The simplest Join is INNER JOIN.

**INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword 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 same.  
**Syntax**:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching\_column = table2.matching\_column;

**table1**: First table.

**table2**: Second table

**matching\_column**: Column common to both the tables.

**Note**: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.



**Example Queries(INNER JOIN)**

* + This query will show the names and age of students enrolled in different courses.
  + SELECT StudentCourse.COURSE\_ID, Student.NAME, Student.AGE FROM Student INNER JOIN StudentCourse ON Student.ROLL\_NO = StudentCourse.ROLL\_NO;

**Output**:  
[](https://media.geeksforgeeks.org/wp-content/uploads/table22.png)

**LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

**Note**: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.  
[](https://i.stack.imgur.com/VkAT5.png)

**Example Queries(LEFT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID FROM Student LEFT JOIN StudentCourse ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output**:  
[](https://media.geeksforgeeks.org/wp-content/uploads/table31.png)

**RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This 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 join. The rows for which there is no matching row on 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

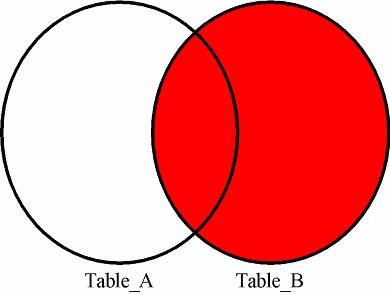
RIGHT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

**Note**: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.  


**Example Queries(RIGHT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**  
[](https://media.geeksforgeeks.org/wp-content/uploads/table6.png)

**FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values.**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.



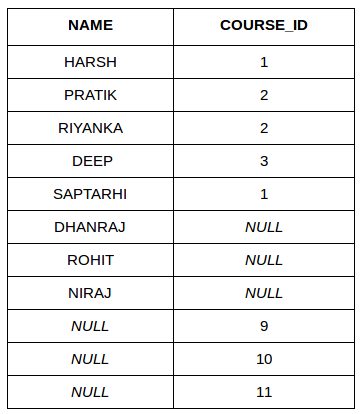
**Example Queries(FULL JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**  
[](https://media.geeksforgeeks.org/wp-content/uploads/table7.png)

**Views**

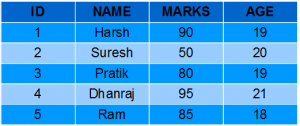
Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

**Sample Tables**:

StudentDetails

[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-57.png)

StudentMarks

[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-58.png)

**CREATING VIEWS**

We can create View using **CREATE VIEW** statement. A View can be created from a single table or multiple tables.

**Syntax**:

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE condition;

**view\_name**: Name for the View

**table\_name**: Name of the table

**condition**: Condition to select rows

**Examples**:

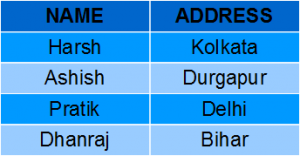
* **Creating View from a single table:**

In this example we will create a View named DetailsView from the table StudentDetails.  
Query:

CREATE VIEW DetailsView AS SELECT NAME, ADDRESS FROM StudentDetails WHERE S\_ID < 5;

To see the data in the View, we can query the view in the same manner as we query a table.

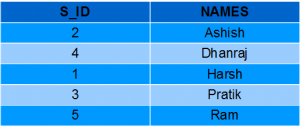
SELECT \* FROM DetailsView;

Output:  
[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-571.png)

In this example, we will create a view named StudentNames from the table StudentDetails.  
Query:  
  
CREATE VIEW StudentNames AS SELECT S\_ID, NAME FROM StudentDetails ORDER BY NAME;

If we now query the view as,

SELECT \* FROM StudentNames;

Output:  
[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-64.png)

* **Creating View from multiple tables**: In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement

. Query:

CREATE VIEW MarksView AS SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME = StudentMarks.NAME;

To display data of View MarksView:

SELECT \* FROM MarksView;

Output:  
[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-591.png)

**DELETING VIEWS**

We have learned about creating a View, but what if a created View is not needed any more? Obviously we will want to delete it. SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

**Syntax**:

DROP VIEW view\_name;

**view\_name**: Name of the View which we want to delete.

For example, if we want to delete the View **MarksView**, we can do this as:

DROP VIEW MarksView;

**UPDATING VIEWS**

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.  
**Syntax**:

CREATE OR REPLACE VIEW view\_name AS

SELECT column1,coulmn2,..

FROM table\_name

WHERE condition;

For example, if we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks**Table, we can do this as:

CREATE OR REPLACE VIEW MarksView AS

SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS, StudentMarks.AGE

FROM StudentDetails, StudentMarks

WHERE StudentDetails.NAME = StudentMarks.NAME;

If we fetch all the data from MarksView now as:

SELECT \* FROM MarksView;

Output:  
[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-60.png)

**Inserting a row in a view**:  
We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.**Syntax**:

INSERT INTO view\_name(column1, column2 , column3,..)

VALUES(value1, value2, value3..);

**view\_name**: Name of the View

**Example**:  
In the below example we will insert a new row in the View DetailsView which we have created above in the example of “creating views from a single table”.

INSERT INTO DetailsView(NAME, ADDRESS)

VALUES("Suresh","Gurgaon");

If we fetch all the data from DetailsView now as,

SELECT \* FROM DetailsView;

Output:  
[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-62.png)

* **Deleting a row from a View**:  
  Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.  
  **Syntax**:

DELETE FROM view\_name WHERE condition;

**view\_name**:Name of view from where we want to delete rows

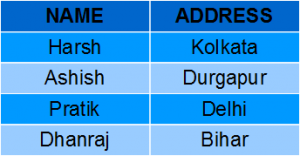
**condition**: Condition to select rows

**Example**:  
In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

DELETE FROM DetailsView WHERE NAME="Suresh";

If we fetch all the data from DetailsView now as,

SELECT \* FROM DetailsView;

Output:  
[](https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-571.png)

**SQL Trigger**

**Trigger:** A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

**Syntax:**

create trigger [trigger\_name]

[before | after]

{insert | update | delete}

on [table\_name]

[for each row]

[trigger\_body]

**Explanation of syntax:**

1. create trigger [trigger\_name]: Creates or replaces an existing trigger with the trigger\_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table\_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger\_body]: This provides the operation to be performed as trigger is fired

**BEFORE and AFTER of Trigger:**  
BEFORE triggers run the trigger action before the triggering statement is run.  
AFTER triggers run the trigger action after the triggering statement is run.

**Example:**  
Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

**Suppose the database Schema –**

mysql> desc Student;

+-------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------+-------------+------+-----+---------+----------------+

| tid | int(4) | NO | PRI | NULL | auto\_increment |

| name | varchar(30) | YES | | NULL | |

| subj1 | int(2) | YES | | NULL | |

| subj2 | int(2) | YES | | NULL | |

| subj3 | int(2) | YES | | NULL | |

| total | int(3) | YES | | NULL | |

| per | int(3) | YES | | NULL | |

+-------+-------------+------+-----+---------+----------------+

7 rows in set (0.00 sec)

SQL Trigger to problem statement.

*create trigger stud\_marks before INSERT on Student for each row set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per = Student.total \* 60 / 100;*

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);

Query OK, 1 row affected (0.09 sec)

mysql> select \* from Student;

+-----+-------+-------+-------+-------+-------+------+

| tid | name | subj1 | subj2 | subj3 | total | per |

+-----+-------+-------+-------+-------+-------+------+

| 100 | ABCDE | 20 | 20 | 20 | 60 | 36 |

+-----+-------+-------+-------+-------+-------+------+

1 row in set (0.00 sec)

**SQL**Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

## SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

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

--pending

**Delete Vs Truncate:**

| S.NO | Delete | Truncate |
| --- | --- | --- |
| 1. | The DELETE command is used to delete specified rows(one or more). | While this command is used to delete all the rows from a table. |
| 2. | It is a DML(Data Manipulation Language) command. | While it is a DDL(Data Definition Language) command. |
| 3. | There may be WHERE clause in DELETE command in order to filter the records. | While there may not be WHERE clause in TRUNCATE command. |
| 4. | In the DELETE command, a tuple is locked before removing it. | While in this command, data page is locked before removing the table data. |
| 5. | The DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row. | TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log. |
| 6. | DELETE command is slower than TRUNCATE command. | While TRUNCATE command is faster than DELETE command. |
| 7. | To use Delete you need DELETE permission on the table. | To use Truncate on a table we need at least ALTER permission on the table. |
| 8. | Identity of column retains the identity after using DELETE Statement on table. | Identity of the column is reset to its seed value if the table contains an identity column. |
| 9. | The delete can be used with indexed views. | Truncate cannot be used with indexed views. |

**Cursor :**

**Cursor** is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:**  
   Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :**  
   Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

**How to create Explicit Cursor:**

1. **Declare Cursor Object.**  
   **Syntax :** DECLARE cursor\_name CURSOR FOR SELECT \* FROM table\_name

DECLARE s1 CURSOR FOR SELECT \* FROM studDetails.

1. **Open Cursor Connection.**  
   **Syntax :** OPEN cursor\_connection

OPEN s1

1. **Fetch Data from cursor.**  
   There are total 6 methods to access data from cursor. They are as follows :  
   **FIRST** is used to fetch only the first row from cursor table.  
   **LAST** is used to fetch only last row from cursor table.  
   **NEXT** is used to fetch data in forward direction from cursor table.  
   **PRIOR** is used to fetch data in backward direction from cursor table.  
   **ABSOLUTE n** is used to fetch the exact nth row from cursor table.  
   **RELATIVE n** is used to fetch the data in incremental way as well as decremental way.  
   **Syntax :** FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor\_name

FETCH FIRST FROM s1

FETCH LAST FROM s1

FETCH NEXT FROM s1

FETCH PRIOR FROM s1

FETCH ABSOLUTE 7 FROM s1

FETCH RELATIVE -2 FROM s1

1. **Close cursor connection.**  
   **Syntax :** CLOSE cursor\_name

CLOSE s1

1. **Deallocate cursor memory.**  
   **Syntax :** DEALLOCATE cursor\_name

DEALLOCATE s1

**Indexing in DBMS:**

## What is Indexing?

**Indexing** is a data structure technique which allows you to quickly retrieve records from a database file. An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. It’s second column contains a set of [pointers](https://www.guru99.com/c-pointers.html) for holding the address of the disk block where that specific key value stored.

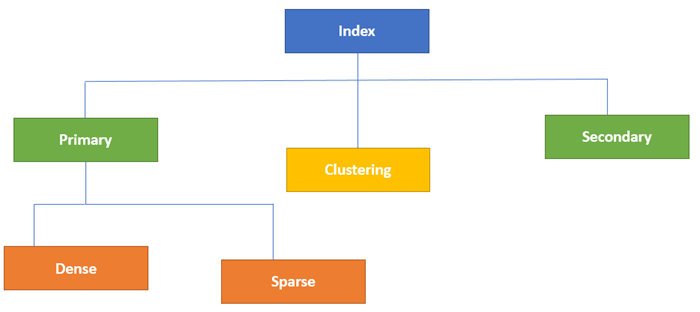
An index -

* Takes a search key as input
* Efficiently returns a collection of matching records.

In this DBMS Indexing tutorial, you will learn:

* [Types of Indexing](https://www.guru99.com/indexing-in-database.html#2)
* [Primary Index](https://www.guru99.com/indexing-in-database.html#3)
* [Secondary Index](https://www.guru99.com/indexing-in-database.html#4)
* [Clustering Index](https://www.guru99.com/indexing-in-database.html#5)
* [What is Multilevel Index?](https://www.guru99.com/indexing-in-database.html#6)
* [B-Tree Index](https://www.guru99.com/indexing-in-database.html#7)
* [Advantages of Indexing](https://www.guru99.com/indexing-in-database.html#8)
* [Disadvantages of Indexing](https://www.guru99.com/indexing-in-database.html#9)

## Types of Indexing

[](https://www.guru99.com/images/1/070119_0833_IndexinginD1.png)Type of Indexes in Database

Indexing in Database is defined based on its indexing attributes. Two main types of indexing methods are:

* Primary Indexing
* Secondary Indexing

## Primary Index

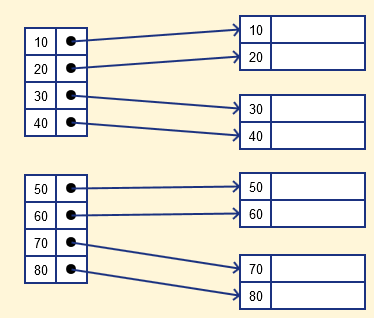
Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.

The primary Indexing in DBMS is also further divided into two types.

* Dense Index
* Sparse Index

### **Dense Index**

In a dense index, a record is created for every search key valued in the database. This helps you to search faster but needs more space to store index records. In this Indexing, method records contain search key value and points to the real record on the disk.

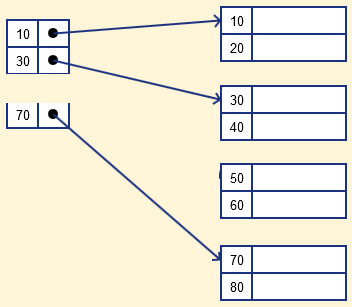
[](https://www.guru99.com/images/1/070119_0833_IndexinginD2.png)

### Sparse Index

It is an index record that appears for only some of the values in the file. Sparse Index helps you to resolve the issues of dense Indexing in DBMS. In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.

However, sparse Index stores index records for only some search-key values. It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records.

Below is an database index Example of Sparse Index

[](https://www.guru99.com/images/1/070119_0833_IndexinginD3.png)

## Secondary Index

The secondary Index in DBMS can be generated by a field which has a unique value for each record, and it should be a candidate key. It is also known as a non-clustering index.

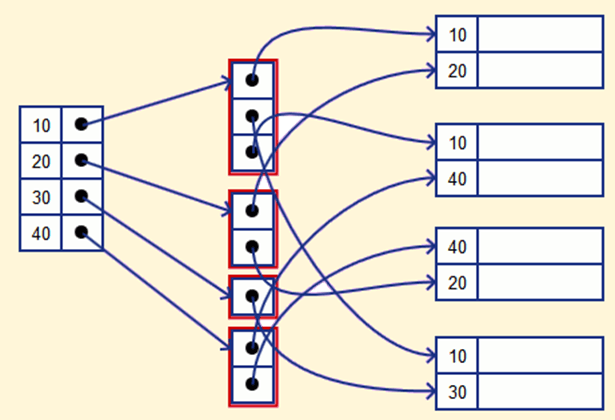
This two-level database indexing technique is used to reduce the mapping size of the first level. For the first level, a large range of numbers is selected because of this; the mapping size always remains small.

### Example of secondary Indexing

Let's understand secondary indexing with a database index example:

In a bank account database, data is stored sequentially by acc\_no; you may want to find all accounts in of a specific branch of ABC bank.

Here, you can have a secondary index in DBMS for every search-key. Index record is a record point to a bucket that contains pointers to all the records with their specific search-key value.

[](https://www.guru99.com/images/1/070119_0833_IndexinginD4.png)

## Clustering Index

In a clustered index, records themselves are stored in the Index and not pointers. Sometimes the Index is created on non-primary key columns which might not be unique for each record. In such a situation, you can group two or more columns to get the unique values and create an index which is called clustered Index. This also helps you to identify the record faster.

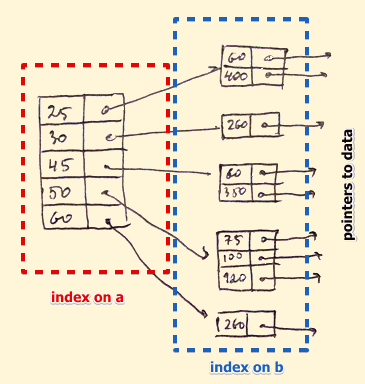
**Example:**

Let's assume that a company recruited many employees in various departments. In this case, clustering indexing in DBMS should be created for all employees who belong to the same dept.

It is considered in a single cluster, and index points point to the cluster as a whole. Here, Department \_no is a non-unique key.

## What is Multilevel Index?

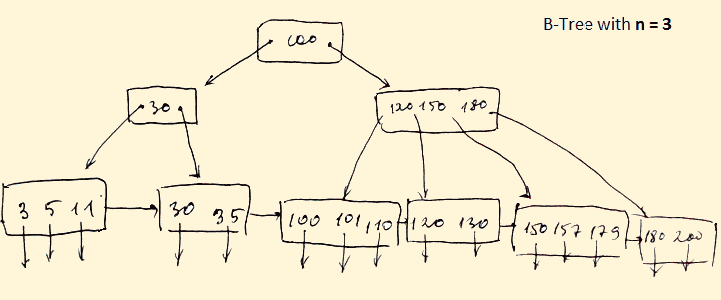
Multilevel Indexing in Database is created when a primary index does not fit in memory. In this type of indexing method, you can reduce the number of disk accesses to short any record and kept on a disk as a sequential file and create a sparse base on that file.

[](https://www.guru99.com/images/1/070119_0833_IndexinginD5.png)

## B-Tree Index

B-tree index is the widely used data structures for tree based indexing in DBMS. It is a multilevel format of tree based indexing in DBMS technique which has balanced binary search trees. All leaf nodes of the B tree signify actual data pointers.

Moreover, all leaf nodes are interlinked with a link list, which allows a B tree to support both random and sequential access.

[](https://www.guru99.com/images/1/070119_0833_IndexinginD6.png)

* Lead nodes must have between 2 and 4 values.
* Every path from the root to leaf are mostly on an equal length.
* Non-leaf nodes apart from the root node have between 3 and 5 children nodes.
* Every node which is not a root or a leaf has between n/2] and n children.

## Advantages of Indexing

Important pros/ advantage of Indexing are:

* It helps you to reduce the total number of I/O operations needed to retrieve that data, so you don't need to access a row in the database from an index structure.
* Offers Faster search and retrieval of data to users.
* Indexing also helps you to reduce tablespace as you don't need to link to a row in a table, as there is no need to store the ROWID in the Index. Thus you will able to reduce the tablespace.
* You can't sort data in the lead nodes as the value of the primary key classifies it.

## Disadvantages of Indexing

Important drawbacks/cons of Indexing are:

* To perform the indexing database management system, you need a primary key on the table with a unique value.
* You can't perform any other indexes in Database on the Indexed data.
* You are not allowed to partition an index-organized table.
* SQL Indexing Decrease performance in INSERT, DELETE, and UPDATE query.

### Summary:

* Indexing is a small table which is consist of two columns.
* Two main types of indexing methods are 1)Primary Indexing 2) Secondary Indexing.
* Primary Index is an ordered file which is fixed length size with two fields.
* The primary Indexing is also further divided into two types 1)Dense Index 2)Sparse Index.
* In a dense index, a record is created for every search key valued in the database.
* A sparse indexing method helps you to resolve the issues of dense Indexing.
* The secondary Index in DBMS is an indexing method whose search key specifies an order different from the sequential order of the file.
* Clustering index is defined as an order data file.
* Multilevel Indexing is created when a primary index does not fit in memory.
* The biggest benefit of Indexing is that it helps you to reduce the total number of I/O operations needed to retrieve that data.
* The biggest drawback to performing the indexing database management system, you need a primary key on the table with a unique value.

Qn1 : 3rd highest salary.

SELECT salary

FROM

(SELECT salary

FROM Table1

ORDER BY salary DESC

LIMIT 3) AS Comp

ORDER BY salary

LIMIT 1;

Qn2: name of employee start with a.

Start with a: SELECT NAME FROM employee WHERE NAME LIKE ' a%';

end with a:SELECT NAME FROM employee WHERE NAME LIKE ' %a';

Qn3 : create an empty table from existing table:

Create table table1 as (select \* from table2 where 1=2);

Qn4: common records from 2 tables:

SELECT T1.\*

FROM T1

INNER JOIN T2

ON T2.FirstName = T1.FirstName

AND T2.LastName = T1.LastName

AND T2.DateOfBirth = T1.DateOfBirth

UNION ALL

SELECT T2.\*

FROM T2

INNER JOIN T1

ON T1.FirstName = T2.FirstName

AND T1.LastName = T2.LastName

AND T1.DateOfBirth = T2.DateOfBirth

Qn 5: alternate records from a table:

To fetch even Numbered row:

SELECT \* FROM table\_name WHERE column\_name % 2 = 0

To fetch odd Numbered row:

SELECT \* FROM table\_name WHERE column\_name % 2 = 1

Qn 6: fetch first 5 character from a string:

SELECT SUBSTRING('SQL Tutorial', 1, 5) AS ExtractString; => SQL T

Qn 7:

In a document database, data is stored as keys and values. A Couchbase bucket contains documents; each document has a unique key and a JSON value. There are no foreign keys (or, more accurately, there are no foreign key constraints).

**Relational vs NoSQL**

Two commonly used types in commercial software are Relational Databases and NoSQL databases.

Relational Database Mamagement Systems (RDBMS) - Represent data as tables where each record of data is another row in the table. Oracle, mySQL and H2 are examples of RDBMS.

NoSQL Databases - Databases that are not relational. These databases are often designed to scale and perform well in a cloud environment. MongoDB, Redis, Cassandra, and Couchbase are examples of NoSQL DBs.

Types of NoSQL Databases

NoSQL databases can be grouped into several types.

\u2022 MapReduce Frameworks, such as Hadoop have 2 calculation / search phases. Map, which breaks problem into sub problems that are distributed and Reduce which combine the results

\u2022 Key-Value Stores like Google BigTable, Amazon Dynamo, Cassandra, and HBase. These are like giant persistent hash tables. You insert into the table by giving a (key, value) pair, and fetch by the key.

\u2022 Document Stores such as Couchbase, MongoDB, SimpleDB. These are like Key-Value Stores except the value is a document, for example an HTML page or a JSON object.

\u2022 Graph Database Systems such as Neo4j, FlockDB, Pregel. These systems are laid out as nodes and edges.

Couchbase DB

Couchbase is an open source distributed NoSQL database which supports fast querying and a document-oriented query-language, N1QL (pronounced "nickel").

Couchbase stores data as JSON documents, each of which has a document id, or document key. A single document can be thought of as the equivalent of a row in an relational database.

Couchbase keeps documents in named Buckets, the equivalent of named databases in a relational database. Multiple instances of Couchbase Server can be combined into a single cluster. Individual nodes can be added, removed, and replaced as appropriate, with no down-time required for the cluster as a whole.

The @Field annotation marks a field for persistence into a JSON field in a Couchbase document. The @ID annotation marks which field should be used as the primary key.