Relational Database Management System

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model .RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.The data in an RDBMS is stored in database objects which are called as **Tables**.Tables is basically a collection of related data stored in the form of **Rows** and **Columns** ,Which is the simplest form of storing data in relational database.The CUSTOMER table below is the example for RDBMS.

ID	NAME	AGE	SALARY
1	ARNAV	24	20000
2	ZAIN	34	30000
3	REENU	32	4000
4	SASHA	25	12222

Every Table has is broken into small entities called **Fields.**The fields in the CUSTOMER table consist of ID, NAME, AGE, and SALARY.A Field is a column in a table that is designed to maintain specific information about every record in the table.A **Record** is also called as a row of data is each individual entry that exists in a table. For example, there are 4 records in the above CUSTOMERS table and single record is represented as below:

1	ARNAV	24	20000

Thus 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.For example, a column in the CUSTOMER table is AGE, which represents AGE and would be as shown below –

AGE
24
34
32
25

DATABASE ENGINE

The Database Engine is the core part of a DBMS that provides access to the data in a database managed by the DBMS.A storage engine is the underlying software component that a database management system (DBMS) uses to create, read, update and delete **(CRUD)** data from a database. Most database management systems include their own application programming interface (API) that allows the user to interact with their underlying engine without going through the user interface of the DBMS.A database engine is software that handles the data structure and physical storage and management of data. Different storage engines have different features and performance characteristics, so a single DBMS could use multiple engines. Ideally, they should not affect the logical view of data presented to users of the DBMS.
For example InnoDB is the default storage engine in MySQL 8.0. InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data.

DDL

DDL stands for Data Definition Language and is used to define the structures like schema, database, tables, constraints etc.It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. The commands are CREATE,DROP,RENAME,ALTER.

List of DDL commands:

- **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).

  Syntax For creating database and table are given below:

1. **CREATE DATABASE** Database_Name;

   Here **Database_Name** specifies the name of the database which we want to create in the system.

   Eg:CREATE DATABASE School;

2.**CREATE TABLE** tablename

   ("column1" "data type",

   "column2" "data type",

   "column3" "data type",

   ...

   "columnN" "data type");

Eg: CREATE TABLE student(studentid **int**, Name **varchar**(255), email **varchar**(255));

- **DROP**: This command is used to delete objects from the database.

  Syntax For dropping database and table are given below:

  1.**DROP DATABASE** Database_Name;

  Eg: DROP DATABASE school;

  2.**DROP TABLE** tablename;

  Eg:DROP TABLE student;

- **ALTER:** The ALTER TABLE statement in Structured Query Language allows you to add, modify, and delete columns of an existing table.

  ADD command can be used to Add column to table.

  Syntax for adding columns to table:

  1.ALTER TABLE table_name ADD column_name column-definition; (for adding single column)

  Eg:ALTER TABLE Cars ADD Car_Model Varchar(20);

2.ALTER TABLE table_name

ADD (column_Name1 column-definition,

column_Name2 column-definition,

.....

column_NameN column-definition);

The above syntax is for adding multiple column to table.

Eg: ALTER TABLE Employee ADD (Emp_ContactNo. Number(13), Emp_EmailID varchar(50))
;

Modify command is used to modify the column definitions.

Syntax for modifying column definition using Modify command is given below:

1.ALTER TABLE table_name MODIFY column_name column-definition;

Eg:ALTER TABLE Cars MODIFY Car_Color Varchar(50);

Above syntax is used to modify single column of table.

2.ALTER TABLE table_name

MODIFY (column_Name1 column-definition,

column_Name2 column-definition,

.....

column_NameN column-definition);

Above syntax is used for modifying multiple columns of table.

For Eg we want to modify both Emp_ContactNo**.** and Emp_EmailID of the above Employee table,Then we should write as:

ALTER TABLE Employee ADD (Emp_ContactNo. Int, Emp_EmailID varchar(80));

DROP command is used when we have to drop the column or columns in the table.

Syntax for using drop with alter is :

ALTER TABLE table_name DROP Column column_name ;

Eg:ALTER TABLE Cars DROP COLUMN Car_Color ;

.In order to drop two or more columns we have to use the below method.

.ALTER TABLE Cars DROP COLUMN Emp_Salary ;

ALTER TABLE Cars DROP COLUMN Emp_City ;

RENAME is used to change the column name or fields of existing table.

Syntax :

ALTER TABLE table_name RENAME COLUMN old_name to new_name;

Eg: Suppose, we want to change the name of the **Car_Color** column of the above Cars table. For this, you have to type the following query in the SQL:

ALTER TABLE Cars RENAME COLUMN Car_Color to Colors;

- **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT**: This is used to add comments to the data dictionary.
- **RENAME:** This is used to rename an object existing in the database.

DML

DML stands for Data Manipulation Language and is used to manipulate data. It is the component of the SQL statement that controls access to data and to the database.Examples of DML are insert, update and delete statements.

List of DML commands:

INSERT: It is used to insert data into a table.

Syntax for inserting data to table without specifying the column name is:

```
INSERT INTO table_name
VALUES (value1, value2, value3....);
```

Eg:**INSERT INTO** STUDENTS
VALUES (5, ABHIRAM, ABHI@GMAIL.COM);

Syntax for inserting data to table by specifying the column name :

```
INSERT INTO table_name (column1, column2, column3....)
VALUES (value1, value2, value3.....);
```

Eg:**INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
VALUES (1, ABHIRAM, 22, ALLAHABAD);

UPDATE: It is used to update existing data within a table.The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database. The SQL DELETE command uses a WHERE clause.**SQL UPDATE** statement is used to change the data of the records held by tables. Which rows is to be updated, it is decided by a condition. To specify condition, we use WHERE clause.

Syntax :

```
UPDATE table_name
SET column_name = expression
WHERE conditions
```

Eg:

```
UPDATE students
SET name= 'beinghuman'
WHERE Student_Id = '3'
```

Inorder to update multiple field we can set it as:

```
UPDATE students
SET name= 'beserious', email= 'Johnny@gmail.com'
WHERE Student_Id = '3'
```

DELETE: It is used to delete records from a database table.If we dont specify the where clause it will delete all the columns from the table.

Syntax :

```
DELETE FROM table_name [WHERE condition];
```

Eg: DELETE FROM students where name=”beserious”;

Here we are deleting the record of student with name beserious

SELECT: It also shows the particular record of a particular column by using the WHERE clause.

Syntax:

1. **SELECT** Column_Name_1, Column_Name_2, ..., Column_Name_N FROM Table_Name;
2. **SELECT * from** tablename**;**

Eg:SELECT * from students;

This will display all rows from students table.

ACID PRINCIPLES

ACID is an acronym that refers to the set of 4 key properties that define a transaction: **Atomicity, Consistency, Isolation, and Durability**. If a database operation has these ACID properties, it can be called an ACID transaction, and data storage systems that apply these operations are called transactional systems.In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

1.Atomicity:The entire transaction takes place at once or doesnt take place at all.

2.Consistency:The database must be consistent before and after transaction.

3.Isolation:Multiple transaction should take place independently without interferance.

4.Durability:Change of successful transaction must occur even if the system fails.

ATOMICITY

Atomicity means either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. It involves the following two operations.

Abort: If a transaction aborts, changes made to the database are not visible.

Commit: If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

Suppose if the transaction fails after T1 then the data will be inconsistent as the money deducted from x but not transferred to y ,So to avoid this the transaction must entirely take place.

Consistency

This means that the integrity constraint of database must be maintained before and after transaction.It refers to the correctness of database.

Total **before T** occurs = **500 + 200 = 700**. Total **after T** occurs = **400 + 300 = 700**. Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

ISOLATIONS

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference.This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.
Let **X**= 500, **Y** = 500.
Consider two transactions **T** and **T".**

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write (Y)	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result, interleaving of operations takes place due to which **T''** reads the correct value of **X** but the incorrect value of **Y** and sum computed by
T'': (X+Y = 50, 000+500=50, 500)
is thus not consistent with the sum at end of the transaction:
T: (X+Y = 50, 000 + 450 = 50, 450).
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

DURABILITY

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.The effects of the transaction, thus, are never lost.

NORMALIZATIONS

Normalization is **the process of minimizing redundancy from a relation or set of relations**. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.Following are the list of normal forms :

1NF (First Normal Form) Rules

FULL NAME	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATIONS
RINA	FIRST PLOT	PIRATES,SALT TREE	MS
PAYAL	SEVENTH VILLA	LITTLE GIRL,CAR	MR
PAYAL	ROSE VILLA	SALT TREES	MR

Here you see **Movies Rented column has multiple values.** Rules of 1NF are:

- Each table cell should contain a single value.
- Each record needs to be unique.

we know that An SQL KEY is a single column or combination of multiple columns used to uniquely identify rows or tuples in the table.A primary is a single column value used to identify a database record uniquely.A composite key is a primary key composed of multiple columns used to identify a record uniquely

In our database, we have two people with the same name payal, but they live in different places.Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

So the above table in 1NF can be written as:

FULL NAME	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATIONS
RINA	FIRST PLOT	PIRATES	MS
RINA	FIRST PLOT	SALT TREE	MS
PAYAL	SEVENTH VILLA	LITTLE GIRL	MR
PAYAL	SEVENTH	CAR	MR
PAYAL	ROSE VILLA	SALT TREES	MR

2NF (Second Normal Form) Rules

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key that does not functionally dependant on any subset of candidate key relation.

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

What we have done is to remove the partial functional dependency that we initially had. Now, in the table [TABLE_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

A database is in third normal form if it satisfies the following conditions:

- It is in second normal form
- There is no transitive functional dependency

By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B.

3rd Normal Form Example

Consider the following example:

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.
To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].