1. **What do you understand By Database?**

Answer: The **Database** is an essential part of our life. As we encounter several activities that involve our interaction with databases, for example in the bank, in the railway station, in school, in a grocery store, etc. These are the instances where we need to store a large amount of data in one place and fetch these data easily.

**Data**: statistics it is raw and unprocessed. ex- name, class, marks etc.

**information**: when data is processed.” record is also information “. example – pass or fail table etc.

**Database**: an organized collection of data and information or interrelated data collected at one place.

1. **What is Normalization?**

* Normalization is the process of organizing the data in the database.
* Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
* Normalization divides the larger table into smaller and links them using relationships.
* The normal form is used to reduce redundancy from the database table.

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

**Following are the various types of Normal forms:**

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

**Following are the various types of Normal forms:**



|  |  |
| --- | --- |
| **Normal Form** | **Description** |
| 1NF | A relation is in 1NF if it contains an atomic value. |
| 2NF | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key |
| 3NF | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| BCNF | A stronger definition of 3NF is known as Boyce Codd's normal form. |
| 4NF | A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency. |
| 5NF | A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless. |

1. **What is Difference between DBMS and RDBMS?**

|  |  |
| --- | --- |
| **DBMS** | **RDBMS** |
| DBMS stores data as file | RDBMS stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present | Normalization is present. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |
| It stores data in either a navigational or hierarchical form. | It uses a tabular structure where the headers are the column names, and the rows contain corresponding values. |
| It deals with small quantity of data. | It deals with large amount of data. |
| Data redundancy is common in this model. | Keys and indexes do not allow Data redundancy. |
| It is used for small organization and deal with small data. | It is used to handle large amount of data. |
| Not all Codd rules are satisfied. | All 12 Codd rules are satisfied. |
| Security is less | More security measures provided. |
| It supports single user. | It supports multiple users. |
| Data fetching is slower for the large amount of data. | Data fetching is fast because of relational approach. |
| The data in a DBMS is subject to low security levels with regards to data manipulation. | There exist multiple levels of data security in a RDBMS. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| Examples: XML, Window Registry, Forxpro, dbaseIIIplus etc. | Examples: MySQL, PostgreSQL, SQL Server, Oracle, Microsoft Access etc. |

Conclusion:

Hence, it can be deduced that the administration system for databases is a software that oversees diverse operations like the technique of information input, the rapidity of information acquisition, and the capability to manage diverse categories of information encompassing structured, semi-structured, and unstructured. It is beneficial when dealing with a limited quantity of data. Alternatively, a relational database pertains to a database that manages organized data. It comprises of distinct elements such as tuples (also known as rows) and schema (also known as tables). It stores data in a tabular form and establishes relationships between tables through key constraints. This type of database is beneficial when handling vast quantities of data.

1. **What is EF Codd Rule of RDBMS Systems?**

Rule 0 − Foundation rule

Any relational database management system that is propounded to be RDBMS or advocated to be a RDBMS should be able to manage the stored data in its entirety through its relational capabilities.

Rule 1 − Rule of Information

Relational Databases should store the data in the form of relations. Tables are relations in Relational Database Management Systems. Be it any user defined data or meta-data, it is important to store the value as an entity in the table cells.

Rule 2 − Rule of Guaranteed Access

The use of pointers to access data logically is strictly forbidden. Every data entity which is atomic in nature should be accessed logically by using a right combination of the name of table, primary key represented by a specific row value and column name represented by attribute value.

Rule 3 − Rule of Systematic Null Value Support

Null values are completely supported in relational databases. They should be uniformly considered as ‘missing information’. Null values are independent of any data type. They should not be mistaken for blanks or zeroes or empty strings. Null values can also be interpreted as ‘inapplicable data’ or ‘unknown information.’

Rule 4 − Rule of Active and online relational Catalog

In the Database Management Systems lexicon, ‘metadata’ is the data about the database or the data about the data. The active online catalog that stores the metadata is called ‘Data dictionary’. The so called data dictionary is accessible only by authored users who have the required privileges and the query languages used for accessing the database should be used for accessing the data of data dictionary.

Rule 5 − Rule of Comprehensive Data Sub-language

A single robust language should be able to define integrity constraints, views, data manipulations, transactions and authorizations. If the database allows access to the aforementioned ones, it is violating this rule.

Rule 6 − Rule of Updating Views

Views should reflect the updates of their respective base tables and vice versa. A view is a logical table which shows restricted data. Views generally make the data readable but not modifiable. Views help in data abstraction.

Rule 7 − Rule of Set level insertion, update and deletion

A single operation should be sufficient to retrieve, insert, update and delete the data.

Rule 8 − Rule of Physical Data Independence

Batch and end user operations are logically separated from physical storage and respective access methods.

Rule 9 − Rule of Logical Data Independence

Batch and end users can change the database schema without having to recreate it or recreate the applications built upon it. Rule 10 − Rule of Integrity Independence Integrity constraints should be available and stored as metadata in data dictionary and not in the application programs.

Rule 11 − Rule of Distribution Independence

The Data Manipulation Language of the relational system should not be concerned about the physical data storage and no alterations should be required if the physical data is centralized or distributed.

Rule 12 − Rule of Non Subversion

Any row should obey the security and integrity constraints imposed. No special privileges are applicable. Almost all full scale DBMSs are RDMSs. Oracle implements 11+ rules and so does Sybase. SQL Server also implements 11+ rules while FoxPro implements 7+ rules.

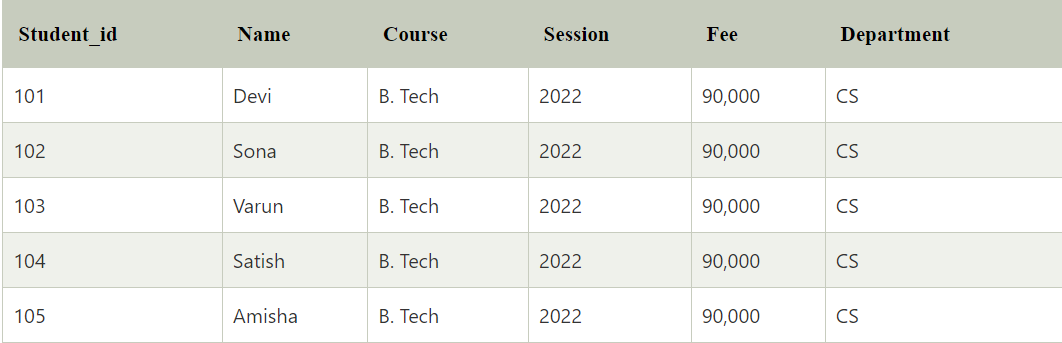
1. **What do you understand By Data Redundancy?**

In DBMS, when the same data is stored in different tables, it causes data redundancy.

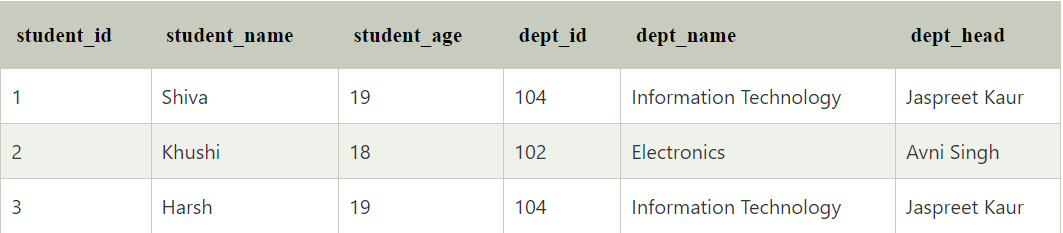
Sometimes, it is done on purpose for recovery or backup of data, faster access of data, or updating data easily. Redundant data costs extra money, demands higher storage capacity, and requires extra effort to keep all the files up to date.

Sometimes, unintentional duplicity of data causes a problem for the database to work properly, or it may become harder for the end user to access data. Redundant data unnecessarily occupy space in the database to save identical copies, which leads to space constraints, which is one of the major problems.

Let us understand redundancy in DBMS properly with the help of an example.



Lets take a example using data redundancy



Problems that are caused due to redundancy in the database

Redundancy in DBMS gives rise to anomalies, and we will study it further. In a database management system, the problems that occur while working on data include inserting, deleting, and updating data in the database.

1. Insertion Anomaly:

Insertion anomaly arises when you are trying to insert some data into the database, but you are not able to insert it.

Example: If you want to add the details of the student in the above table, then you must know the details of the department; otherwise, you will not be able to add the details because student details are dependent on department details.

1. Deletion Anomaly:

Deletion anomaly arises when you delete some data from the database, but some unrelated data is also deleted; that is, there will be a loss of data due to deletion anomaly.

Example: If we want to delete the student detail, which has student\_id 2, we will also lose the unrelated data, i.e., department\_id 102, from the above table.

1. Updating Anomaly:

An update anomaly arises when you update some data in the database, but the data is partially updated, which causes data inconsistency.

Example: If we want to update the details of dept\_head from Jaspreet Kaur to Ankit Goyal for Dept\_id 104, then we have to update it everywhere else; otherwise, the data will get partially updated, which causes data inconsistency.

**Advantages of data redundancy in DBMS**

Provides Data Security: Data redundancy can enhance data security as it is difficult for cyber attackers to attack data that are in different locations.

Provides Data Reliability: Reliable data improves accuracy because organizations can check and confirm whether data is correct.

Create Data Backup: Data redundancy helps in backing up the data.

**Disadvantages of data redundancy in DBMS**

Data corruption: Redundant data leads to high chances of data corruption.

Wastage of storage: Redundant data requires more space, leading to a need for more storage space.

High cost: Large storage is required to store and maintain redundant data, which is costly.

**How to reduce data redundancy in DBMS**

We can reduce data redundancy using the following methods:

Database Normalization: We can normalize the data using the normalization method. In this method, the data is broken down into pieces, which means a large table is divided into two or more small tables to remove redundancy. Normalization removes insert anomaly, update anomaly, and delete anomaly.

Deleting Unused Data: It is important to remove redundant data from the database as it generates data redundancy in the DBMS. It is a good practice to remove unwanted data to reduce redundancy.

Master Data: The data administrator shares master data across multiple systems. Although it does not remove data redundancy, but it updates the redundant data whenever the data is changed.

**Conclusion**: You have read this article about Data Redundancy in Database Management Systems. You have understood that data redundancy refers to the repetition of similar data, which may be done intentionally or it may be accidentally.

You have studied the problems caused by data redundancy, such as delete anomaly, insert anomaly, and update anomaly.

You have studied the advantages and disadvantages of data redundancy in DBMS.

You have studied some of the methods which reduce data redundancy in DBMS.

1. **What is DDL Interpreter?**

A DDL (Data Definition Language) interpreter is a software component or tool that executes and processes commands written in a Data Definition Language. DDL is a subset of SQL (Structured Query Language) that is specifically used for defining and managing the structure of a database schema.

An interpreter is a program that takes input in a specific language and processes it to perform the desired actions. In the case of a DDL interpreter, it takes DDL commands as input and carries out the necessary operations to create, modify, or delete database objects such as tables, indexes, views, constraints, and so on.

The DDL interpreter typically parses the input commands, validates the syntax and semantics, and performs the requested actions on the database schema. It interacts with the database management system (DBMS) to execute the necessary SQL statements to implement the requested changes.

Some common DDL commands that a DDL interpreter can handle include:

- CREATE: Used to create new database objects such as tables, indexes, views, etc.

- ALTER: Used to modify the structure of existing database objects.

- DROP: Used to delete or remove database objects.

- TRUNCATE: Used to remove all data from a table while keeping its structure intact.

- RENAME: Used to rename existing database objects.

- COMMENT: Used to add comments or annotations to the database objects.

By using a DDL interpreter, developers and database administrators can easily manage the structure of a database without directly interacting with the underlying database system. It provides a convenient and controlled way to define and modify the schema, ensuring data integrity and consistency.

1. **What is DML Compiler in SQL?**

In SQL (Structured Query Language), a DML (Data Manipulation Language) compiler is a component of a database management system (DBMS) responsible for processing and executing DML statements. DML statements are used to manipulate and query data within a database.

The primary role of a DML compiler is to interpret and execute DML statements written in SQL. These statements include commands such as SELECT, INSERT, UPDATE, and DELETE, which perform various operations on the data stored in database tables.

When a DML statement is issued, the DML compiler performs the following tasks:

1. Syntax Parsing: The DML compiler verifies the syntax of the statement to ensure it adheres to the rules and grammar of SQL. If there are any syntax errors, it will raise an error or exception.

2. Semantic Analysis: The DML compiler performs semantic analysis to validate the statement's semantics. This involves checking the correctness and consistency of the statement, such as verifying that the referenced tables and columns exist and the user has the necessary privileges to perform the operation.

3. Query Optimization: In complex queries involving multiple tables and conditions, the DML compiler applies various optimization techniques to determine the most efficient way to execute the query. It analyzes factors such as indexing, query statistics, and query execution plans to optimize the query's performance.

4. Query Execution: Once the DML statement is parsed, validated, and optimized, the DML compiler generates an execution plan or query plan. This plan outlines the steps required to execute the statement efficiently. The DML compiler passes the execution plan to the query execution engine, which carries out the necessary actions to manipulate or retrieve the data.

In summary, a DML compiler in SQL is responsible for parsing, validating, optimizing, and executing DML statements. It plays a crucial role in processing queries and ensuring the efficient manipulation of data within a database.

1. **What is SQL Key Constraints? writing an Example of SQL Key Constraints.**

In SQL, key constraints are rules or conditions applied to the columns or groups of columns in a database table. They enforce data integrity and define the relationships between the data within a table. Key constraints ensure that certain properties or characteristics hold true for the specified columns. There are several types of key constraints commonly used in SQL:

1. Primary Key Constraint: A primary key constraint ensures that each row in a table is uniquely identified. It specifies that the values in one or more columns of a table must be unique and not null. A primary key constraint is typically applied to a single column or a combination of columns that uniquely identify each record in the table.

2. Unique Key Constraint: A unique key constraint ensures that the values in one or more columns of a table are unique. Unlike a primary key constraint, a unique key constraint allows null values, meaning that one or more null values can exist in the unique key column(s). However, the non-null values must still be unique.

3. Foreign Key Constraint: A foreign key constraint establishes a relationship between two tables. It ensures that the values in a column (or a group of columns) in one table correspond to the values in a primary key or unique key column(s) of another table. The foreign key constraint maintains referential integrity and helps enforce data consistency across related tables.

4. Check Constraint: A check constraint defines a condition that values in a column must satisfy. It allows you to specify a logical expression or condition that the data in the column must adhere to. Check constraints are commonly used to enforce domain integrity, ensuring that only valid data is stored in the table.

These key constraints play a vital role in maintaining data accuracy, consistency, and integrity within a database. They help enforce rules and relationships, preventing the insertion of invalid or inconsistent data.

EXAMPLES:

Primary key constraint: -

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT

);

Unique key constraint: -

CREATE TABLE Students (

StudentID INT,

StudentName VARCHAR(50),

Email VARCHAR(100) UNIQUE

);

Foreign key constraints: -

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

Check constraints: -

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(50),

Price DECIMAL(10, 2),

Quantity INT,

CONSTRAINT CHK\_Quantity CHECK (Quantity >= 0)

);

1. **What is save Point? How to create a save Point write a Query?**

The SAVEPOINT statement sets a named transaction savepoint with a name of identifier. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The ROLLBACK TO SAVEPOINT statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does not release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

Here the syntax of SAVEPOINT:

SAVEPOINT savepoint\_name;

Example of SAVEPOINT:

START TRANSACTION;

INSERT INTO employees (employee\_id, first\_name, last\_name) VALUES (1, 'John', 'Doe');

SAVEPOINT sp1;

INSERT INTO employees (employee\_id, first\_name, last\_name) VALUES (2, 'Jane', 'Smith');

INSERT INTO employees (employee\_id, first\_name, last\_name) VALUES (3, 'Michael', 'Johnson');

SAVEPOINT sp2;

INSERT INTO employees (employee\_id, first\_name, last\_name) VALUES (4, 'Sarah', 'Williams');

ROLLBACK TO SAVEPOINT sp1;

COMMIT;

In this example, a transaction is initiated using the START TRANSACTION statement. Two savepoints, "sp1" and "sp2," are created at different points within the transaction. If an error occurs or a rollback is desired, the ROLLBACK TO SAVEPOINT statement can be used to roll back to a specific savepoint. Finally, the COMMIT statement is used to commit the changes made in the transaction.

Savepoints provide flexibility within transactions, allowing you to control the granularity of rollback operations and handle errors or exceptional conditions more effectively.

1. **What is trigger? and how to create a Trigger in SQL?**

A trigger in MySQL is a set of SQL statements that reside in a system catalog. It is a special type of stored procedure that is invoked automatically in response to an event. Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Generally, triggers are of two types according to the SQL standard: row-level triggers and statement-level triggers.

Row-Level Trigger: It is a trigger, which is activated for each row by a triggering statement such as insert, update, or delete. For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the insert, update, or delete statement.

Statement-Level Trigger: It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.

Types of Triggers in MySQL?

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

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

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

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

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

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

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

When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

**Syntax of trigger:**

CREATE TRIGGER trigger\_name

{BEFORE | AFTER} {INSERT | UPDATE | DELETE}

ON table\_name

FOR EACH ROW

BEGIN

-- SQL statements to be executed

END;

**Example of trigger:**

CREATE TRIGGER before\_insert\_employee

BEFORE INSERT

ON employees

FOR EACH ROW

BEGIN

SET NEW.column\_names,column2\_name…;

END;