Q.1] For the database system to be usable, it must retrieve data efficiently.The need of efficiency has led designers to use complex data structures to represent data in the database. Developers hide this complexity from the database system users through several levels of abstraction. Explain those levels of abstraction in detail with example.

ANS: The concept of abstraction is crucial in designing database systems, as it allows users and developers to work with complex data structures and operations without needing to understand all the underlying details. Abstraction provides different levels of interaction and hides complexities, making the system more usable and manageable. In the context of database systems, there are typically three levels of abstraction: the physical level, the logical level, and the view level. Let's explore each of these levels with examples:

1. Physical Level (Internal Level): This is the lowest level of abstraction and deals with how data is stored on the physical storage devices like hard drives. It involves details such as data storage formats, access methods, and indexing techniques.Users and application developers rarely interact with this level directly.

Example: At the physical level, the database management system (DBMS) might use techniques like clustering and indexing to store data efficiently. If you have a table "Employees", the physical level would involve how the rows of data are stored on the disk, and how indexes are used to speed up data retrieval.

2. Logical Level (Conceptual Level): The logical level abstracts the physical implementation details and focuses on describing the overall structure of the database. It defines the schema, which includes tables, relationships, keys, and constraints. This level is more user-centric than the physical level and provides a way to create, modify, and query the database using high level constructs.

Example: Consider a database for a library. At the logical level, you would define tables like "Books", "Authors", and "Customers", along with their attributes and relationships. You would also specify constraints, such as ensuring that a book can't be checked out by multiple customers simultaneously.

3. View Level (External Level): The view level is the highest level of abstraction visible to end-users and application programs. It allows different users to have different customized views of the database. Users can define and interact with their own perspective of the data without knowing about the underlying physical and logical structures.

Example: In the library database, different views could be created for librarians, customers, and administrators. A librarian's view might show information about overdue books and available copies. A customer's view might show their borrowing history and current checkouts. These views are tailored to the specific needs of each group without exposing unnecessary complexity.

Q.2] Explain the concept of candidate key and primary key, foreign key. Identify above listed key for the following schema:

Person (driver_id, name, address, contactno)

Car(licence, model, year)

Owns (driver_id, licence)

ANS:

1.Candidate Key:

A candidate key is a unique identifier for a tuple (row) within a relation (table). It's a set of one or more attributes that, when combined, uniquely identify each row in the table. There can be multiple candidate keys in a table, but one of them is chosen as the primary key.

2.Primary Key:

The primary key is a candidate key that is chosen as the main unique identifier for a table. It ensures that each row in the table is uniquely identifiable and that no two rows have the same values for the primary key attributes. A primary key also ensures that null values are not allowed in the key attributes.

3.Foreign Key:

A foreign key is an attribute or set of attributes in a table that refers to the primary key of another table. It establishes a relationship between two tables, allowing data to be linked between them. The foreign key enforces referential integrity, ensuring that the values in the foreign key column(s)

SPPU ENGINEERING UPDATES

Study material provided by : Vishwajeet Londhe

Join community by clicking below links

@SPPU_ENGINEERING_UPDATE

Insta Page

https://www.instagram.com/sppu_engineering_update

@SPPU_TE_BE_COMP

Telegram Channel

https://t.me/SPPU_TE_BE_COMP

WhatsApp Channel

https://whatsapp.com/channel/0029VaAhRMdAzNbmPG0jyq2x

match existing values in the primary key column(s) of the referenced table or are null. In the schema provided:

- Candidate Keys:
  - Candidate Key for the "Person" table could be "driver_id" since it uniquely identifies each person.
  - Candidate Key for the "Car" table could be "licence" since it uniquely identifies each car.
- Primary Keys:
  - Primary Key for the "Person" table could be "driver_id".
  - Primary Key for the "Car" table could be "licence".
- Foreign Key:
  - In the "Owns" table, "driver_id" and "licence" are foreign keys that reference the "Person" and "Car" tables respectively. These foreign keys establish a relationship between the "Owns" table and the other two tables.

Q.3] Draw architecture of DBMS system and explain function of following components:

i) Storage manager

ii) Query Processor

ANS: DBMS Architecture:

A typical DBMS architecture can be divided into several components, with each component having specific roles and responsibilities. Here's a simplified description:

1. User Interfaces: This is the front-end of the DBMS where users interact with the system. It includes interfaces for database administrators, application developers, and end-users. Different types of interfaces could be command line interfaces, graphical user interfaces (GUIs), or web-based interfaces.
2. Query Processor: This component receives and processes user queries and requests. It translates high-level queries written in SQL (Structured Query Language) into low-level operations that can be executed by the system. The query processor optimizes queries for efficient execution.
3. Storage Manager: This is a crucial component responsible for managing the physical storage of data. It interacts directly with the file system and manages how data is stored, retrieved, and updated on the storage devices.
4. Data Dictionary Manager: The data dictionary stores metadata about the database, including information about the structure of tables, data types, constraints, indexes, and more. It's used by various components of the DBMS to understand and manage the database's schema.
5. Transaction Manager: This component ensures the ACID properties (Atomicity, Consistency, Isolation, Durability) of database transactions. It manages concurrent transactions, schedules their execution, and ensures that data remains consistent despite multiple simultaneous users.
6. Concurrency Control Manager: It's responsible for managing concurrent access to the database by multiple users or transactions. This component ensures that transactions do not interfere with each other, maintaining data integrity.
7. Buffer Manager: The buffer manager handles data in memory, keeping frequently accessed data blocks in a buffer cache to improve data retrieval performance. It decides what data to bring into memory and what data to write back to disk.

Functions of the Components:

i) Storage Manager: The Storage Manager plays a critical role in managing how data is stored, retrieved, and organized on physical storage devices. Its functions include:

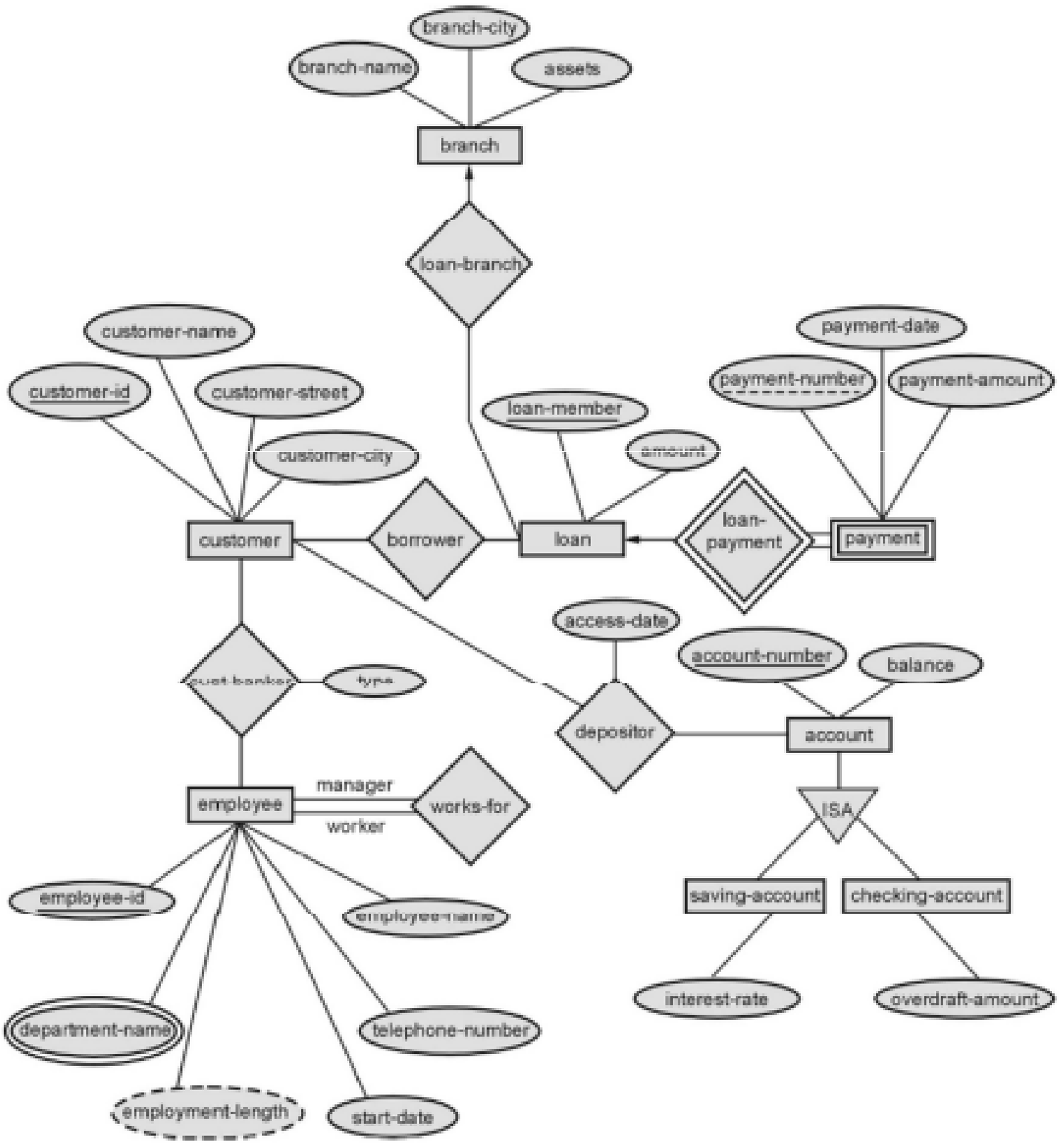
- File Organization: Deciding how data files are organized on disk for efficient storage and retrieval.
- Data Storage: Managing the allocation and deallocation of space for data on disk.
- Data Retrieval: Fetching data from disk into memory when requested by the Query Processor or other components.
- Data Update: Writing changes made to data back to disk to maintain consistency.
  - Indexing: Managing data indexes to speed up data retrieval operations. ●Buffer Management: Managing the buffer cache in memory to optimize data access times.

ii) Query Processor: The Query Processor is responsible for handling user queries and translating them into executable operations. Its functions include: ●Query Parsing: Breaking down user queries into their syntactic and semantic components.

- Query Optimization: Deciding the most efficient way to execute a query, considering indexes, join algorithms, and other factors.
- Execution Plan Generation: Creating a plan that outlines the sequence of operations needed to execute the query.
- Accessing Metadata: Consulting the Data Dictionary to understand the structure and constraints of the database.
- Data Retrieval and Manipulation: Initiating operations to retrieve, insert, update, or delete data based on the query.
- Error Handling: Detecting and handling errors in the query, such as syntax errors or incorrect references.

Q.4] Draw an ER diagram for the banking system. Assume the banking requirements are as given below. • The bank is organized into branches. Each branch is located in a particular city.
• The bank offers two types of accounts: saving and current. Accounts can be held by more than one customer and a customer can have more than one account.
• A loan originates at a particular branch and can be held by one or more customers
Identify the relationship among the entities along with the mapping cardinalities, keys in the E.R. diagram. Construct appropriate tables for E-R diagram designed with above requirements

ANS:-



Q.5] Define DBMS. Explain advantages of DBMS over file system.

ANS: DBMS (Database Management System): A Database Management System (DBMS) is a software system that allows users to efficiently store, manage, retrieve, and manipulate data in a structured manner. It provides a set of tools and functionalities to create, maintain, and control databases, ensuring data consistency, security, and ease of access.

Advantages of DBMS over File System:

1. Data Integrity and Security: DBMS enforces data integrity by applying rules and constraints to ensure accurate and reliable data. It also offers security features to control who can access and modify the data, protecting sensitive information.
2. Reduced Data Redundancy: In a file system, the same data might be duplicated across multiple files, leading to redundancy and inconsistency.
   DBMS centralizes data storage, minimizing redundancy and ensuring data consistency.
3. Data Sharing and Collaboration: DBMS allows multiple users and applications to access and manipulate the data simultaneously. This promotes efficient collaboration and eliminates issues of data being locked in inaccessible files.
4. Data Querying and Reporting: DBMS provides query languages (e.g., SQL) that allow users to retrieve specific data quickly. It supports complex queries, aggregations, and reporting, making data analysis more powerful and efficient.
5. Data Centralization and Management: DBMS centralizes data management tasks, such as backups, data recovery, and data migration. This simplifies administrative tasks and reduces the risk of data loss.

Q.6] Different components of database management systems like query processor, storage manager, transaction manager etc. are functional for processing the query submitted by user. Explain the functions of each component in view of getting query output

ANS: The components of a Database Management System (DBMS) work together to process user queries and provide query output. Let's delve into the functions of each of these components in view of getting query output:

1. Query Processor: The Query Processor is responsible for handling user queries and transforming them into executable tasks that retrieve or manipulate data. Its functions include:
   - Query Parsing: Breaking down the user's query into syntactic and semantic components to understand its structure and purpose.
   - Query Optimization: Deciding the most efficient way to execute the query. This involves considering factors like available indexes, join algorithms, and cost estimates.
   - Execution Plan Generation: Creating a plan that outlines the sequence of operations needed to execute the query. This plan is often represented as a query execution tree.
   - Data Retrieval and Manipulation: Initiating the actual operations to fetch, insert, update, or delete data based on the query.
   - Error Handling: Detecting and handling errors in the query, such as syntax errors or incorrect references.
2. Storage Manager: The Storage Manager handles the physical storage of data on disk and its efficient retrieval into memory. Its functions include:
   - File Organization: Deciding how data files are organized on disk for optimal storage and retrieval.
   - Data Storage: Managing the allocation and deallocation of space for data on disk.
   - Buffer Management: Keeping frequently accessed data blocks in a buffer cache in memory to improve data access times.
   - Data Retrieval and Update: Fetching data from disk into memory when requested and writing changes back to disk to maintain consistency.
   - Indexing: Managing data indexes to speed up data retrieval operations.
3. Transaction Manager: The Transaction Manager ensures the ACID properties (Atomicity, Consistency, Isolation, Durability) of database transactions. Its functions include:
   - Transaction Control: Managing the start, commit, and rollback of transactions.
   - Concurrency Control: Handling multiple transactions executing simultaneously to ensure data consistency and isolation.
   - Isolation Levels: Defining levels of isolation to control how much one transaction can see the changes made by other transactions.
   - Deadlock Detection and Resolution:

Identifying and resolving deadlocks that occur when transactions are waiting for each other to release resources.

4. Data Dictionary Manager: The Data Dictionary Manager stores metadata about the database's structure and constraints. Its functions include: ●Metadata Storage: Storing information about tables, columns, indexes, constraints, and more.
   - ●Schema Enforcement: Ensuring that data entered into the database conforms to the specified structure and constraints.
   - ●Access Control: Managing permissions and access rights for different users and roles.

Q.7] Explain with example what is physical data independence. Also explain its importance.

ANS: Physical Data Independence refers to the concept in database management systems (DBMS) where changes to the physical storage and organization of data do not affect the way users and applications interact with the data. In other words, modifications made to how data is stored on the disk (physical level) should not require changes to the way users query or manipulate the data (logical level).

Example:

Let's say you have a database with a table named "Employees." At the physical level, the data is stored on the disk in specific files and data blocks. Now, imagine that you need to reorganize the data storage to improve performance. You might decide to move some data to a different disk location, change the storage format, or implement new indexing techniques.

With physical data independence, you can make these changes to the storage structure without needing to modify the SQL queries or application code that interacts with the "Employees" table. Users and applications would still write and retrieve data from the "Employees" table using the same queries as before, even though the data's physical arrangement has changed.

Importance of Physical Data Independence: Physical data independence is important for several reasons:

1. Flexibility: It allows database administrators to make improvements to the system's performance, storage, and maintenance without affecting the users or applications. This flexibility makes it easier to adapt the database as technology evolves.
2. Reduced Maintenance: Separating the physical and logical levels means that changes to the underlying storage structure won't necessitate modifications to applications or queries. This reduces the effort and complexity of maintenance.
3. Efficiency: Database administrators can fine-tune the physical storage to optimize performance without worrying about breaking existing functionality. This is particularly crucial for large databases that need efficient storage management.
4. Database Evolution: As the database grows over time, physical data independence enables administrators to reorganize and scale the database system as needed, keeping it efficient and responsive.
5. Data Security and Privacy: Changes made to enhance security (like encryption) can be implemented at the physical level without affecting the logical level where users interact with the data.

Q.8] What is view and how to create it? Can you update view? If yes, how? If not, why not?

ANS: What is a View: A view in a database is like a virtual table that doesn't store any data itself but instead displays data from one or more existing tables. It's a way to present a specific subset of data or a customized perspective of the data to users without changing the underlying tables. Views allow users to query and manipulate data as if they were working with a regular table.

Creating a View: To create a view, you define a query that specifies the data you want the view to display. Here's a simple example using a hypothetical "Students" and "Courses" table:

Let's say you have two tables:

- ●Students (Student_ID, Student_Name, Age)
- ●Courses (Course_ID, Course_Name, Instructor)

Updating a View: Yes, you can update a view, but there are some restrictions. You can perform the following operations on a view:

- Insert new rows using the view.
- Update existing rows using the view.
- Delete rows using the view.

However, there are limitations based on the complexity of the view and the way it's defined. For example:

- If the view involves multiple tables, you might only be able to update rows from one table.
- Views with complex calculations or subqueries might not be updatable. ●Some DBMSs require that the view's definition adheres to specific criteria for updates to be allowed.

Why Not Always Updateable:

Views might not always be updateable due to the complexity of the underlying data relationships. The DBMS needs to ensure that updates made through the view can be accurately translated back to the underlying tables without ambiguity. When the view's definition is too complex or involves multiple tables in intricate ways, the DBMS might prevent updates to maintain data integrity.

Q.9] Defined stored procedure. Explain the creating and calling stored procedure with example.

ANS: Stored Procedure:

A stored procedure is a set of pre-written SQL statements that are stored in the database and can be executed as a single unit. They can perform specific tasks, process data, or retrieve information. Stored procedures provide reusability, security, and a way to encapsulate complex operations.

Creating a Stored Procedure:

To create a stored procedure, you define a series of SQL statements and give it a name. Here's a simple example of creating a stored procedure that retrieves information about students:

```
CREATE PROCEDURE GetStudentInfo
AS
BEGIN
SELECT Student_Name, Age FROM Students;
END;
```

In this example, the stored procedure "GetStudentInfo" doesn't take any input parameters. When executed, it will retrieve the names and ages of all students from the "Students" table.

Calling a Stored Procedure:

You can call a stored procedure to execute its predefined set of SQL statements. Here's how you might call the "GetStudentInfo" stored procedure: When you run this command, the stored procedure will execute its SELECT statement, and the result (student names and ages) will be displayed.

Example Scenario:

Imagine you have a "Sales" table, and you want to create a stored procedure that calculates the total sales for a specific product:

```
CREATE PROCEDURE CalculateTotalSales
@ProductID INT
AS
BEGIN
SELECT SUM(SaleAmount) AS TotalSales
FROM Sales
WHERE Product_ID = @ProductID;
END;
```

To call this stored procedure and get the total sales for a specific product (e.g., ProductID = 101):

OUTPUT:

```
EXEC CalculateTotalSales @ProductID = 101;
```

Q.10] Consider following schema. Student_fee_details (rollno, name, fee_deposited, date) Write a trigger to preserve old values of student fee details before updating in the table.

ANS: In SQL, you can create a trigger to preserve old values of student fee details before updating the table. Here's an example trigger that does that: CREATE TRIGGER PreserveFeeHistory

```
ON Student_fee_details
AFTER UPDATE
AS
BEGIN
                    INSERT INTO FeeUpdateHistory (rollno, name, old_fee_deposited,
                    new_fee_deposited, update_date)
                SELECT i.rollno, i.name, d.fee_deposited, i.fee_deposited, GETDATE()  FROM inserted
                i
INNER JOIN deleted d ON i.rollno = d.rollno;
END;
```

In this example:

- PreserveFeeHistory is the name of the trigger.
- Student_fee_details is the name of the table you want to track updates for. ●FeeUpdateHistory is a hypothetical table where you store the history of fee updates (you need to create this table separately).

This trigger fires after an update operation (AFTER UPDATE) on the Student_fee_details table. It inserts a new record into the FeeUpdateHistory table with information about the old and new fee values, along with the date of the update. The inserted and deleted pseudo-tables are used to access the new and old values affected by the update operation.

Q.11] Consider the following schemes

Supplier(SNO, Sname, Status, City)

Parts (PNO, Pname, Color, Weight, City)

Shipments(SNO,PNO,QTY)

Write SQL queries for the following:

i) Find shipment information (SNO, Sname, PNO, Pname, QTY) for those having quantity less than 157.

ii) List SNO, Sname, PNO, Pname for those suppliers who made shipments of parts whose quantity is larger than the average quantity iii) Find aggregate quantity of PNO 1692 of color green for which shipments made by supplier number who residing Mumbai

ANS:

i) Find shipment information (SNO, Sname, PNO, Pname, QTY) for those having quantity less than 157:

sql

```
SELECT S.SNO, S.Sname, SH.PNO, P.Pname, SH.QTY
FROM Supplier S
JOIN Shipments SH ON S.SNO = SH.SNO
JOIN Parts P ON SH.PNO = P.PNO
WHERE SH.QTY < 157;
```

ii) List SNO, Sname, PNO, Pname for those suppliers who made shipments of parts whose quantity is larger than the average quantity: sql

```
SELECT S.SNO, S.Sname, SH.PNO, P.Pname
FROM Supplier S
JOIN Shipments SH ON S.SNO = SH.SNO
JOIN Parts P ON SH.PNO = P.PNO
WHERE SH.QTY > (SELECT AVG(QTY) FROM Shipments);
```

iii) Find aggregate quantity of PNO 1692 of color green for which shipments made by supplier number who resides in Mumbai:

sql

```
SELECT SUM(SH.QTY) AS AggregateQuantity
```

FROM Shipments SH

JOIN Supplier S ON SH.SNO = S.SNO

JOIN Parts P ON SH.PNO = P.PNO

WHERE P.PNO = 1692 AND P.Color = 'green' AND S.City = 'Mumbai'; Please note that in these queries, I assumed that the relationships between the tables are as follows:

- Supplier: SNO is the primary key.
- Parts: PNO is the primary key.
- Shipments: SNO and PNO are foreign keys referring to the Supplier and Parts tables respectively.

Q.12] What is an index? What are the advantages and disadvantages of using index on a table?

ANS: Index: An index in a database is like an organized list of pointers that helps the database management system (DBMS) quickly locate specific rows in a table. It's similar to an index in a book that helps you find information faster by referring to page numbers.

Advantages of Using Indexes:

1. Faster Data Retrieval: Indexes allow the DBMS to quickly find the rows that match certain conditions in a query. Without indexes, the DBMS would need to scan the entire table, which can be slow for large tables.
2. Improved Query Performance: Queries that involve filtering, sorting, or joining data are much faster when indexes are used. They help reduce the amount of data the DBMS needs to search through.
3. Efficient Data Modifications: While indexes speed up data retrieval, they also make data modifications (inserts, updates, deletes) slightly slower. However, the overall benefit of faster queries usually outweighs the small slowdown in modifications.
4. Data Integrity: Indexes can enforce data integrity by ensuring that certain columns have unique values or by enforcing constraints.
5. Optimized Joins: Indexes on columns used for joining tables can significantly speed up queries that involve multiple tables.

Disadvantages of Using Indexes:

1. Increased Storage Space: Indexes take up additional storage space on disk. For very large tables, this can become a concern.
2. Slower Data Modifications: As mentioned earlier, while inserts, updates, and deletes are faster without indexes, they can be slightly slower with indexes due to the extra work required to maintain index structures.
3. Maintenance Overhead: Indexes need to be updated whenever the data changes, which adds overhead to data modification operations. 4. Complexity: Managing indexes and deciding which columns to index requires expertise and careful consideration. Poorly chosen indexes can actually slow down queries.
5. Selectivity Impact: Indexes are most effective on columns with high selectivity (columns with many distinct values). Indexes on columns with low selectivity might not provide as much performance improvement.

Q.13] What is a trigger? How to create it? Discuss various types of triggers.

ANS: Trigger:

A trigger in a database is like an automatic response to an event, similar to a "trigger" in the real world that sets off an action when a specific condition is met.

In a database, a trigger is a predefined set of instructions that are automatically executed when certain events (like an insert, update, or delete) occur on a table.

Creating a Trigger:

To create a trigger, you specify the event that should trigger the action, define the action itself, and associate it with a table. Here's a simple example of creating a trigger that updates a "LastModified" timestamp whenever a row is updated in the "Employees" table:

```
sql
CREATE TRIGGER UpdateLastModified
ON Employees
AFTER UPDATE
AS
BEGIN
 UPDATE Employees
 SET LastModified = GETDATE()
                                WHERE Employee_ID IN (SELECT Employee_ID FROM inserted);  END;
```

In this example, the trigger is named "UpdateLastModified." It's set to fire after an update operation (AFTER UPDATE) on the "Employees" table. The trigger updates the "LastModified" column with the current date and time whenever a row is updated.

Types of Triggers:

1. Before Triggers: These triggers fire before the event occurs (e.g., before an insert, update, or delete operation). They can be used to validate data or modify values before they are actually changed.
2. After Triggers: These triggers fire after the event has occurred. They are often used for auditing, logging, or performing additional actions after data has been modified.
3. Instead Of Triggers: These triggers replace the standard action that would occur due to an event. They are often used to customize the behavior of certain actions.