1. Identifying Tools and Statements for Modifying Database Content

a. PART 1:

1. INSERT Statement

The INSERT statement is used to add new records (rows) into a table.

Syntax:

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

- You specify the table name and the columns in which you want to insert values.
- If values are provided for all columns, the column list is optional.

Example:

```
INSERT INTO Employees (FirstName, LastName, Age)
VALUES ('John', 'Doe', 28);
```

2. UPDATE Statement

The UPDATE statement modifies existing records in a table. You can update one or more records depending on the condition.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

- The SET clause specifies which columns to update and their new values.
- The WHERE clause determines which rows should be updated (without it, all rows will be affected).

Example:

```
UPDATE Employees
SET Age = 29
WHERE FirstName = 'John' AND LastName = 'Doe';
```

3. DELETE Statement

The DELETE statement removes existing records from a table. The WHERE clause is important to specify which rows should be deleted.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

- Without a WHERE clause, all rows in the table will be deleted.

Example:

```
DELETE FROM Employees
WHERE FirstName = 'John' AND LastName = 'Doe';
```

4. TRUNCATE Statement

The TRUNCATE statement removes all records from a table, but unlike DELETE, it cannot be rolled back and does not fire triggers.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE Employees;
```

5. ALTER Statement

The ALTER statement modifies the structure of a table. It can be used to add, delete, or modify columns, or change the table constraints.

Syntax to add a column:

```
ALTER TABLE table_name
ADD column_name datatype;
```

Syntax to modify a column:

```
ALTER TABLE table_name
MODIFY column_name datatype;
```

Syntax to delete a column:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example (Add a new column):

```
ALTER TABLE Employees
ADD Address VARCHAR(255);
```

Example (Modify an existing column):

```
ALTER TABLE Employees
MODIFY Age INT;
```

6. DROP Statement

The DROP statement is used to delete a table or database completely. This action cannot be undone.

Syntax:

```
DROP TABLE table_name;
```

- You can also use it to drop databases:

```
DROP DATABASE database_name;
```

Example (Drop a table):

```
DROP TABLE Employees;
```

Example (Drop a database):

```
DROP DATABASE Company;
```

7. CREATE Statement

The CREATE statement is used to create new tables, databases, indexes, or views in SQL.

Syntax to create a table:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
);
```

Example (Create a table):

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT
);
```

8. RENAME Statement

The RENAME statement allows you to rename an existing table.

Syntax:

```
RENAME TABLE old_table_name TO new_table_name;
```

Example:

```
RENAME TABLE Employees TO Staff;
```

9. REPLACE Statement

The REPLACE statement works similarly to INSERT, but if a row with the same unique key already exists, it replaces the old row with the new one.

Syntax:

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

Example:

```
REPLACE INTO Employees (EmployeeID, FirstName, LastName, Age)
VALUES (1, 'John', 'Doe', 29);
```

10. MERGE Statement

The MERGE statement allows you to perform INSERT, UPDATE, or DELETE operations in a single statement based on conditions. It's used to merge records from two tables.

Syntax:

```
MERGE INTO target_table USING source_table
ON merge_condition
WHEN MATCHED THEN
    UPDATE SET column1 = value1, column2 = value2, ...
WHEN NOT MATCHED THEN
    INSERT (column1, column2, ...) VALUES (value1, value2, ...);
```

Example:

```
MERGE INTO Employees AS Target
USING NewEmployees AS Source
ON Target.EmployeeID = Source.EmployeeID
WHEN MATCHED THEN
    UPDATE SET Target.Age = Source.Age
WHEN NOT MATCHED THEN
    INSERT (EmployeeID, FirstName, LastName, Age)
    VALUES (Source.EmployeeID, Source.FirstName, Source.LastName, Source.Age);
```

b. Part 2:

MySQL Workbench is a powerful tool for database management and development that includes various features and tools for working with MySQL databases. Here's a description of the key functionalities of the tools you mentioned:

1. SQL Editor

- Purpose: The SQL Editor in MySQL Workbench is the main interface for interacting with your MySQL database using SQL queries.
- Functionality:
    - Write and Execute Queries: You can write SQL queries directly and execute them to interact with the database. This includes SELECT, INSERT, UPDATE, DELETE, and other SQL operations.
    - Syntax Highlighting: The editor provides syntax highlighting for SQL keywords, making it easier to read and understand your queries.
    - Autocomplete: The editor includes auto-completion for SQL commands, table names, column names, and other objects, which speeds up query writing and reduces errors.
    - Query Execution: Queries can be executed individually or in batches. Results are displayed in tabs, with options to view data in a grid or in raw text format.
    - Error Highlighting: If there's an issue with your SQL code, errors are highlighted, and helpful messages are displayed to guide you in fixing them.
    - SQL History: MySQL Workbench keeps a history of queries you've executed, allowing you to quickly access previous queries.

2. Schema Inspector

- Purpose: The Schema Inspector provides detailed information about the structure of your database schemas (databases).
- Functionality:
    - Database Structure Overview: It allows you to view and inspect the structure of your database, including tables, views, stored procedures, and triggers.
    - Table Details: You can view detailed information about tables, including column types, indexes, foreign keys, constraints, and more.
    - Metadata Management: The Schema Inspector provides a visual interface for managing the metadata of your database, such as renaming tables, altering column types, and modifying indexes or constraints.
    - Object Dependencies: It helps identify dependencies between database objects, making it easier to manage complex schemas.

- Data Modeling: You can perform basic data modeling and design tasks, such as analyzing and modifying existing schemas.
- Database Validation: The tool allows you to check for issues or inconsistencies in the schema, such as invalid foreign keys or missing indexes.

3. Query Builder

- Purpose: The Query Builder is a visual tool that helps users design SQL queries without needing to manually write SQL code.
- Functionality:
  - Visual Query Design: The Query Builder lets you design complex SQL queries by dragging and dropping tables and columns, instead of writing the SQL code manually.
  - Join and Relationship Visualization: You can visually represent table relationships and apply joins (INNER, LEFT, RIGHT, etc.) without worrying about the syntax.
  - Filter and Sorting Options: You can specify conditions (WHERE clauses), order by sorting, and limit the number of results directly through the graphical interface.
  - Preview SQL Code: As you build your query visually, MySQL Workbench generates the corresponding SQL code automatically, so you can see the query behind the visual design.
  - Ease of Use for Beginners: This tool is particularly useful for users who are not familiar with SQL syntax or those who prefer working with a more graphical interface to create queries.

5. Understanding Transactions

A transaction in database management is a sequence of one or more operations (such as INSERT, UPDATE, DELETE) that are executed as a single unit. Transactions ensure that the database remains in a consistent state even if a failure occurs during the operation. They are crucial for ensuring the ACID properties:

1. Atomicity: All operations within a transaction are treated as a single unit. Either all operations are successfully completed, or none are (rollback in case of failure).

2. Consistency: A transaction ensures that the database transitions from one consistent state to another. Any operation that violates the database's integrity constraints will be rolled back.

3. Isolation: Each transaction is executed independently of other transactions, ensuring that one transaction's operations do not affect another's.

4. Durability: Once a transaction is committed, its changes are permanent and survive system crashes.

Transactions are crucial in database systems to maintain data integrity, concurrency control, and error recovery.

7. Understanding Record Locking Policies

Record locking is a critical mechanism for ensuring data consistency in concurrent systems. Two primary types of locking mechanisms are used to manage simultaneous access to records:

1. Pessimistic Locking: Locks a record as soon as a transaction accesses it, ensuring that no other user can modify it until the transaction completes. This method prevents conflicts but can lead to performance issues such as deadlocks or long wait times.

2. Optimistic Locking: Allows multiple users to read and modify data simultaneously but checks for conflicts only at the time of committing changes. If a conflict is detected, the transaction fails, and the user must retry. This method is better suited for systems with high concurrency and low conflict rates.

In a test scenario where two users attempt to update the same record simultaneously, pessimistic locking prevents conflicts by blocking one of the users, while optimistic locking allows both updates but handles conflicts at the commit stage.

8. Ensuring Data Integrity and Consistency

In this section, you should discuss:

- Potential Data Integrity Issues: Explain the types of data integrity issues that were identified in the Sakila database, such as orphan records, invalid data, and missing references.

- Foreign Key Constraints: Discuss how foreign key constraints were implemented to maintain referential integrity and prevent orphan records. For each foreign key, explain the relationship between tables and the ON DELETE and ON UPDATE actions chosen.

- Triggers: Discuss the triggers that were created to enforce business rules, such as preventing rental returns before the rental date and automatically updating the last_update field when a customer is updated.

This approach ensures that the database remains consistent, adheres to business rules, and handles data integrity issues automatically through constraints and triggers.

Cites:

- chatgpt.com
- https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html
- https://dev.mysql.com/doc/refman/8.0/en/create-table-check-constraints.html