Basic SQL statements.

In MySQL, SQL statements like `INSERT`, `UPDATE`, `DELETE`, and `ALTER` enable users to modify the content and structure of databases. MySQL Workbench provides powerful tools such as the SQL Editor for executing queries, Schema Inspector for managing schema objects, and Query Builder for visually constructing SQL queries. These tools simplify database management tasks, making them more accessible for both beginner and advanced users.

Transactions in Database Management.

A **transaction** is a sequence of **one or more SQL operations** executed as a single unit of work. Transactions follow the **ACID properties** to ensure data integrity and consistency.

ACID Properties

1. **Atomicity** – All operations **succeed completely or fail entirely** (no partial changes).
2. **Consistency** – The database remains in a **valid state** before and after a transaction.
3. **Isolation** – Transactions **don't interfere** with each other, preventing data corruption.
4. **Durability** – Once committed, changes **persist even after failures** (e.g., power loss).

Why Are Transactions Important?

- **Prevent data corruption** in case of errors or crashes.
- **Ensure consistency** in multi-step processes (e.g., bank transfers).
- **Allow rollback** to undo unintended changes.

Pessimistic Locking:

- **Definition:** In pessimistic locking, when a transaction locks a record, it prevents other transactions from modifying or reading that record until the transaction is completed. This method assumes that conflicts will occur, and therefore, locks are placed on data to avoid these conflicts.

- **Process:**

    - Locks are placed on data when it is accessed, and these locks prevent others from accessing the same data until the first transaction is complete.
    - This can lead to reduced system throughput because other transactions are forced to wait for the lock to be released.
- **When to Use:**

    - Suitable in high-conflict environments where the likelihood of data inconsistencies is high.

    - Common in banking or financial systems where data integrity is critical.
- **Disadvantages:**
    - Reduced concurrency, which can cause delays or performance bottlenecks.
    - Risk of deadlocks if two transactions lock resources in conflicting ways.

Optimistic Locking:

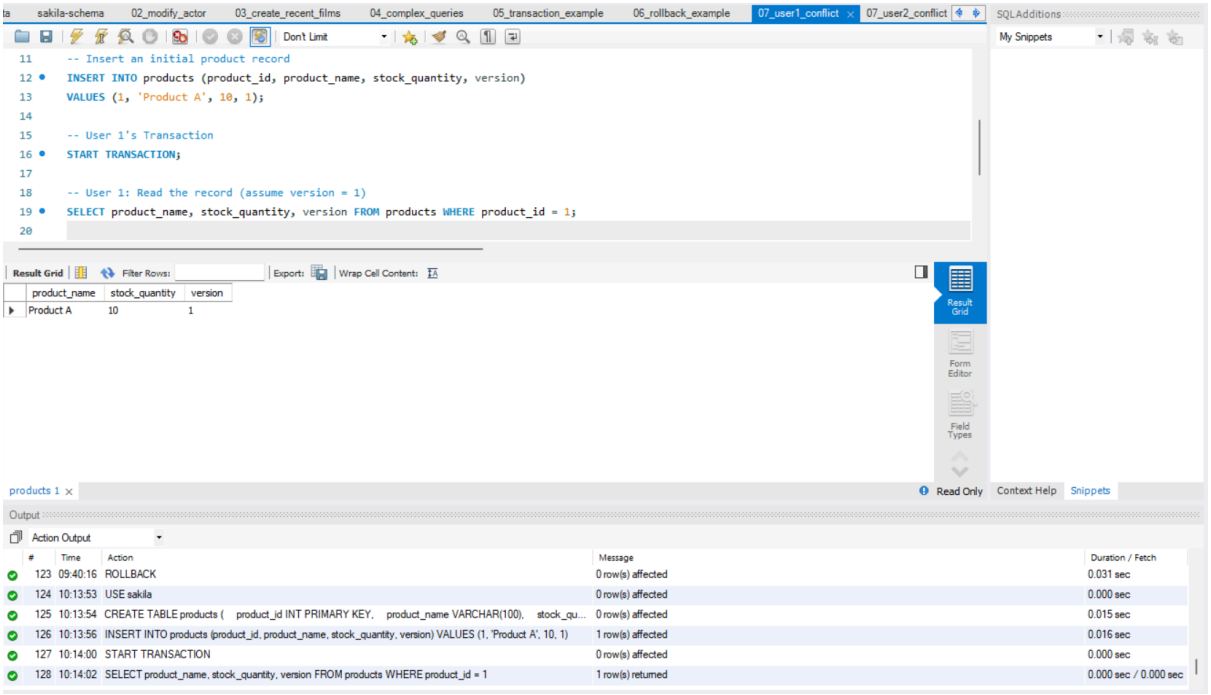
- **Definition:** Optimistic locking, on the other hand, assumes that conflicts are rare and does not lock data when it is being read or updated. Instead, the transaction checks whether another transaction has modified the data before committing the changes.

- **Process:**
    - A transaction proceeds without acquiring a lock on the data.
    - Upon committing the transaction, it verifies if the data has been altered by other transactions. If changes have occurred, the transaction is rolled back and must be retried or resolved.
- **When to Use:**
    - Best suited for low-conflict environments with many concurrent reads and fewer updates.
    - Often used in applications like content management systems, online stores, or other systems where data conflicts are unlikely.
- **Disadvantages:**
    - Potential for conflicts if two users try to modify the same data concurrently.
    - Requires a mechanism to detect and resolve conflicts when they occur, such as versioning or timestamp checks.

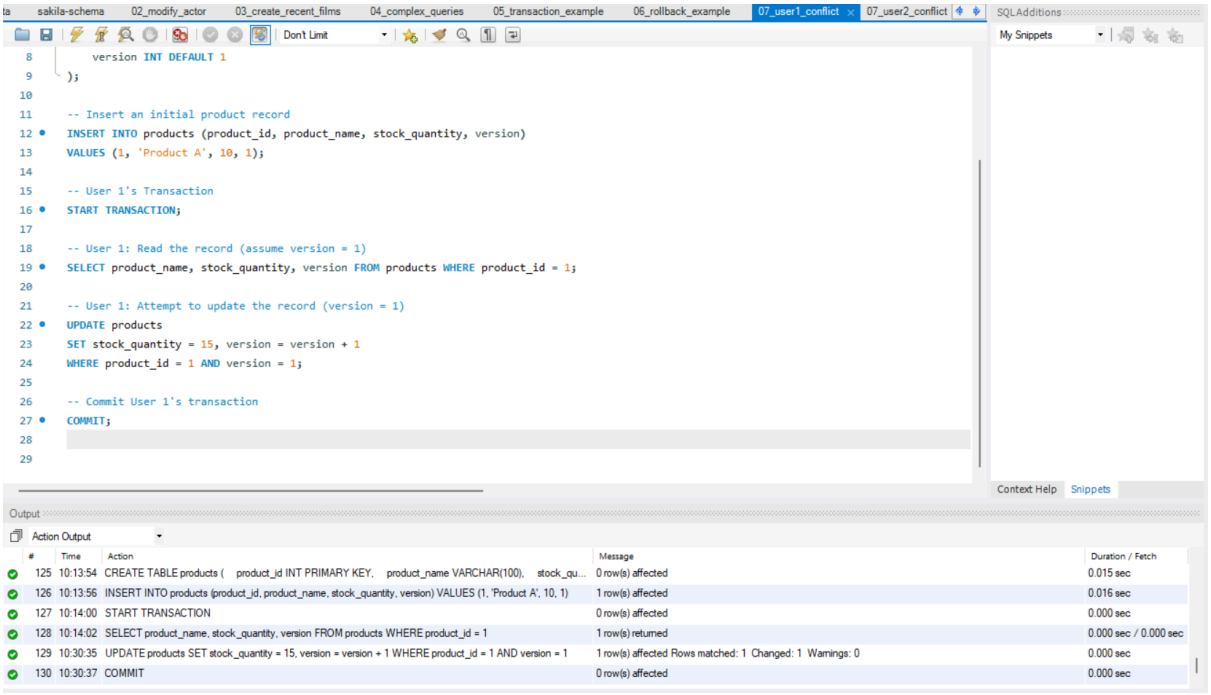
Example:

In order to simulate a scenario where two users try to update the same record simultaneously, I will be using two scripts to simulate the two users and I'll add a new table "products" and when the first one adds a product and commits the changes, the second one will try and update when an error occurs due to the change made by the first user. The example, will be demonstrated with screenshots and the queries used, will be uploaded to the repo.

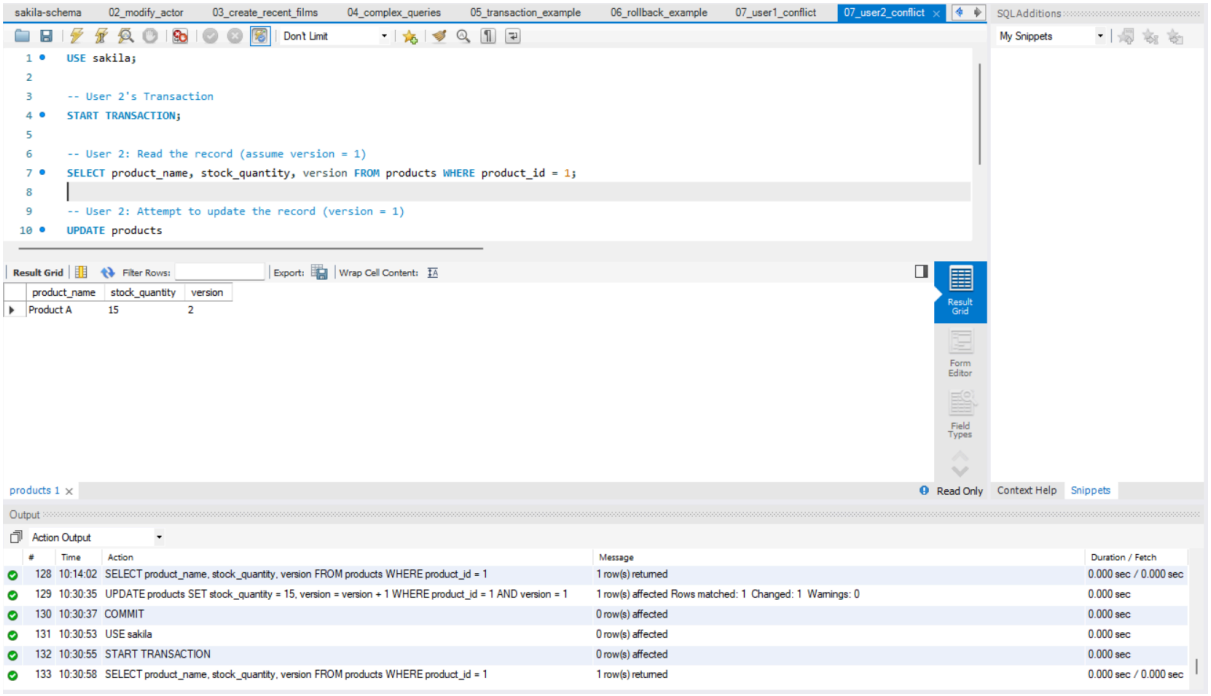
The new table is created and the user 1, adds a Product A



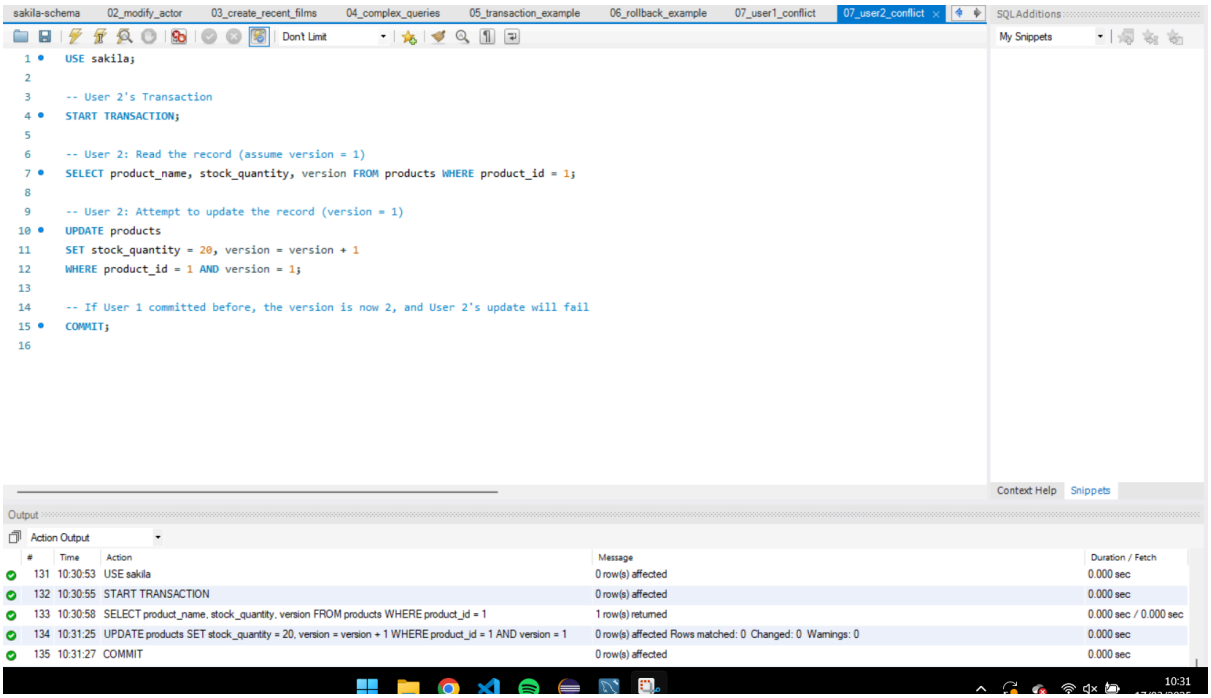
Then, the user changes the values of stock and commits.



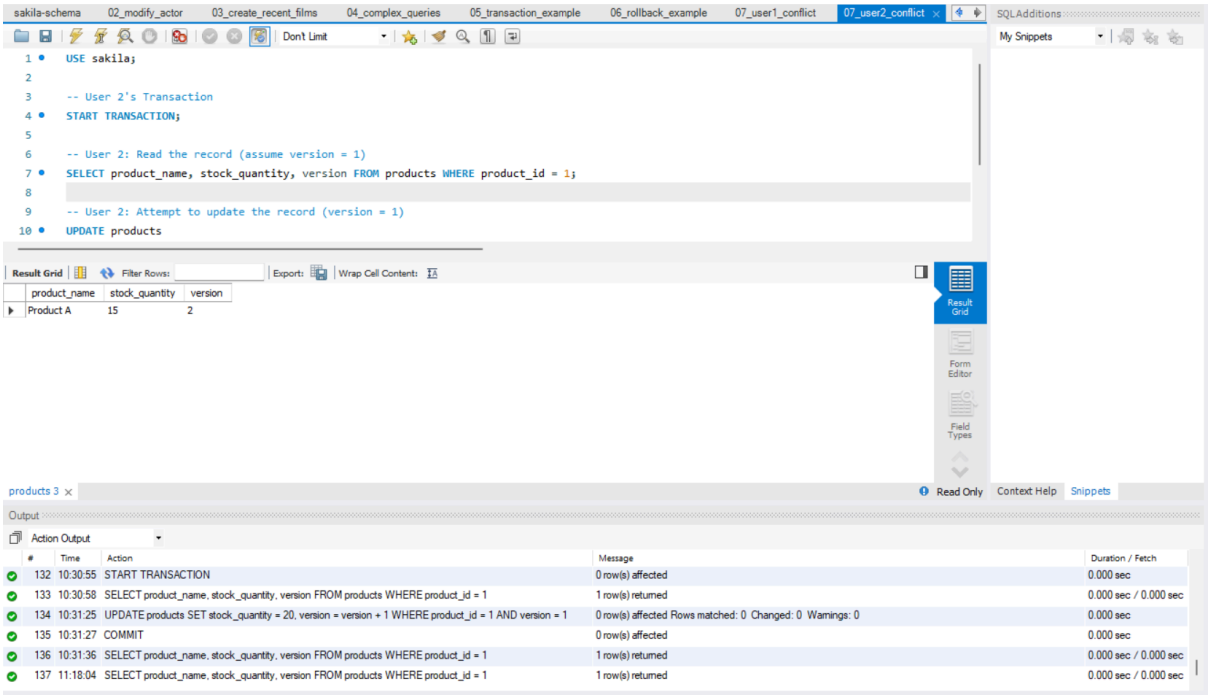
Verification of the changed values:



Commitment of the changes done by the second user.



When retrieving the information about the new table, we can see that the value has not changed because when the first change is done, the second one is unable to be done



Potential integrity Issues in the Salika DB.

Orphaned Rentals and Payments: If a customer is deleted from the `customer` table, any related records in `rental` and `payment` tables could remain, which would cause orphaned records.
Inventory and Film Consistency: The `inventory` table references films in the `film` table. If a film record is deleted, it could leave records in `inventory` that point to a non-existent film.
Staff and Store Consistency: If a staff member is deleted, all references to that staff in `rental` and `payment` tables should be handled properly to maintain data integrity.

Solutions for Potential Integrity Issues in the Sakila DB.

Foreign Key Constraints:

Ensures referential integrity between tables, ensuring that all records in child tables (`rental`, `payment`, `inventory`) point to valid records in the parent tables (`customer`, `staff`, `film`, `store`).

Uses `ON DELETE CASCADE` to automatically delete child records when the referenced parent record is deleted.

Triggers:

Prevents inconsistencies such as renting a film that doesn't exist in the inventory or deleting a customer with active rentals.

Logs deleted records in the `inventory` table for auditing purposes.

Challenges.

- Connecting the database to the mysql server:
At first, I thought that I had to run the model file directly into MySQL Workbench, that is when I tried to run the data and schema files in workbench so I could use the table structure and the data without the model.

- The statements not returning values:
While I was executing a retrieval type of query, I stumbled upon a statement that returned no value, which I found strange, so I searched online what could be the cause of it, and I found that this could be due to the table either having no values within the conditions applied, or directly not having any value into the table. So I retrieved all the data from that table and it returned nothing, coming to the conclusion that while the statement was properly written, the values that I was looking for simply did not exist.

-

External References.

- https://www.tutorialspoint.com/sql/sql-transactions.htm
- https://www.codecademy.com/article/sql-commands
- https://www.geeksforgeeks.org/difference-between-pessimistic-approach-and-optimistic-approach-in-dbms/