RESEARCH.-

I didn't know how to write in the condition that it was greater than something

CÓMO HAGO QUE LA CONDICIÓN DEL WHERE SEA MAYOR DE 200 update actor
set first_name = "KYLIAN", last_name = "MBAPPE"
where actor_id = 201;

Para hacer que la condición del `WHERE` en un `UPDATE` afecte a todos los registros donde `actor_id` sea **mayor de 200**, debes usar el operador `>` **(mayor que)** en la condición.

✅ SQL correcto:

sql Copiar Editar

```sql
UPDATE actor
SET first_name = "KYLIAN", last_name = "MBAPPE"
WHERE actor_id > 200;
```

I didn't know how to store a result in a new table

Store the result in a new table called recent_films

To **store the result** of the query (movies released after 2005) in a new table called `recent_films`, follow these steps:

1️⃣ Create the `recent_films` table

You can create the table with the same structure as `film`, but you can choose to store only relevant columns.

sql Copiar Editar

```sql
CREATE TABLE recent_films AS
SELECT * FROM film WHERE release_year > 2005;
```

I didn't know how to do this part

Write an SQL script that:
- Lists all customers who have rented a film in the last 30 days.
- Identifies the most rented film in the database.
- Displays the total revenue generated per store.

Here's an SQL script that accomplishes the three tasks:

1️⃣ List all customers who have rented a film in the last 30 days

sql Copiar Editar

```sql
SELECT DISTINCT c.customer_id, c.first_name, c.last_name, r.rental_date
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
WHERE r.rental_date >= NOW() - INTERVAL 30 DAY;
```

Then I tried to understand why it is called c.customer_id

Por qué pone c.customer_id

📌 ¿Por qué se usa `c.customer_id` en la consulta SQL?

En la consulta SQL sobre la base de datos **Sakila**, se usa `c.customer_id` porque:

1. `c` **es un alias para la tabla** `customer`
   - En la consulta se define `customer c`, lo que significa que `c` representa la tabla `customer`.
   - Esto permite escribir `c.customer_id` en lugar de `customer.customer_id`, haciendo la consulta más corta y clara.
2. **Evita ambigüedad en la consulta**
   - Como se están uniendo varias tablas (`customer` y `rental`), ambas pueden tener una columna con el mismo nombre (`customer_id`).
   - Es necesario indicar de qué tabla proviene para evitar errores.

I asked about transactions and their importance, but I didn't understand it so much

Explain transactions and their importance in database management.
- Using MySQL Workbench, perform a transaction that:
  - Inserts a new rental record.
  - Updates the inventory to reflect the rental.
  - Commits the transaction.

Transactions and Their Importance in Database Management

A **transaction** in database management is a sequence of one or more operations that are executed as a **single unit of work**. Transactions follow the **ACID properties** to ensure data integrity:

1. **Atomicity** – Ensures that either all operations within a transaction are completed successfully or none are executed.
2. **Consistency** – Guarantees that the database remains in a valid state before and after a transaction.
3. **Isolation** – Ensures that concurrent transactions do not interfere with each other.
4. **Durability** – Ensures that once a transaction is committed, its changes are permanent.

Transactions are essential in **MySQL** (and other databases) for **data integrity** and **error handling**. They prevent partial updates that can lead to inconsistencies, especially in multi-step processes like financial transactions, booking systems, and inventory mana ent.

I asked about the scenario where a transaction is partially executed but later rolled back due to an error, because I did not how to start it and

Demonstrate a scenario where a transaction is partially executed but later rolled back
due to an error (e.g., an attempt to rent a movie that is out of stock).

Scenario: Rolling Back a Transaction Due to an Error

In a video rental system like **Sakila**, we need to ensure that customers can only rent movies that are available in stock. If an attempt is made to rent a movie that is **out of stock**, we should **rollback the transaction** to maintain data consistency.

Steps in the Scenario:

1. **Start a transaction.**
2. **Check if a movie is available for rental.**
3. **Insert a new rental record if stock is available.**
4. **Update the inventory to reflect the rental.**
5. **If stock is unavailable, rollback the transaction.**

I asked about types of locking and I understood both, but I do not how to write it so I asked about it again

Research and document different types of record-locking mechanisms (pessimistic
vs. optimistic locking).
● Test a scenario where two users attempt to update the same record simultaneously

Types of Record-Locking Mechanisms

In database management systems (DBMS), **record locking** is a mechanism used to prevent multiple users from accessing or modifying the same record simultaneously, which could lead to data corruption or inconsistency. Record-locking mechanisms are essential for maintaining data integrity in multi-user environments. Two primary types of record-locking mechanisms are **Pessimistic Locking** and **Optimistic Locking**.

1. **Pessimistic Locking**

Pessimistic locking assumes that conflicts between users accessing the same data are likely to occur, so it locks records to prevent any other user from modifying the data until the lock is released. This type of locking involves "locking" the record at the beginning of the transaction and holding the lock until the transaction is completed. This prevents other users from updating or sometimes even reading the locked data, depending on the type of lock implemented.

Haz queries de eso

Claro, a continuación te proporcionaré ejemplos de consultas (queries) SQL para simular un escenario con **bloqueo pesimista** y **bloqueo optimista**.

1. Escenario de Bloqueo Pesimista

En este ejemplo, se utilizará un **bloqueo exclusivo** (también conocido como `SELECT FOR UPDATE`) para bloquear un registro mientras se está editando.

Asd

Then I asked about the scenario that task ask for

Test a scenario where two users attempt to update the same record simultaneously

Prueba de un Escenario en el que Dos Usuarios Intentan Actualizar el Mismo Registro Simultáneamente

Vamos a crear un escenario en el que dos usuarios intentan actualizar el mismo registro al mismo tiempo en una base de datos. Para este ejemplo, utilizaremos tanto **bloqueo pesimista** como **bloqueo optimista**. A continuación, te explicaré los pasos de ambas situaciones y cómo podríamos implementarlo.

Base de Datos de Ejemplo

Supongamos que tenemos una tabla llamada `cuentas_bancarias` que contiene la información de las cuentas bancarias de los usuarios.

Estructura de la tabla `cuentas_bancarias`:

sql Copiar Editar

```sql
CREATE TABLE cuentas_bancarias (
    id INT PRIMARY KEY,
    saldo DECIMAL(10, 2),
    moneda VARCHAR(10),
    version INT -- Columna para bloquear de manera optimista
);
```

There was a problema, my information source starts queries with “begin” and it doesnt works on my queries, so I asked about another way to start transaction

Me da error el begin transaction

El error que estás obteniendo con `BEGIN TRANSACTION` puede estar relacionado con la configuración de tu base de datos o el motor de base de datos que estás utilizando. Algunas bases de datos utilizan diferentes comandos para iniciar una transacción. A continuación te proporcionaré soluciones para los sistemas de bases de datos más comunes.

1. MySQL o MariaDB:

En MySQL, el comando correcto para iniciar una transacción es `START TRANSACTION` o `BEGIN`.

sql Copiar Editar

```sql
-- Iniciar una transacción
START TRANSACTION;

-- O alternativamente
BEGIN;

-- Aquí va el resto de las operaciones
```

Después de las operaciones que realizas en la transacción, puedes usar `COMMIT` para confirmar los cambios, o `ROLLBACK` si deseas revertir la transacción.

To finish, I asked about potential issues that Sakila database would have

Identify potential data integrity issues in the Sakila database.
● Implement foreign key constraints and triggers to maintain data consistency.

Identificación de Posibles Problemas de Integridad de Datos en la Base de Datos Sakila

La base de datos **Sakila** es un esquema de ejemplo que simula una tienda de alquiler de películas, y contiene diversas tablas relacionadas entre sí, como `film`, `customer`, `rental`, `payment`, `actor`, entre otras. Dado que las bases de datos reales pueden sufrir problemas de **integridad de datos**, es importante identificar posibles áreas en las que puedan ocurrir estos problemas y tomar medidas para asegurarse de que los datos se mantengan consistentes.

A continuación, se presentan algunos **posibles problemas de integridad de datos** en la base de datos Sakila y cómo podemos implementar **restricciones de clave foránea (foreign key)** y **triggers** para mantener la consistencia de los datos.

1. Problema: Registros Huérfanos (Orphan Records)

Un **registro huérfano** ocurre cuando hay registros en una tabla secundaria que no tienen una correspondencia en la tabla principal. Por ejemplo, si un alquiler (`rental`) hace referencia a un cliente (`customer`) que ha sido eliminado, pero el alquiler sigue existiendo, esto sería un problema de integridad referencial.

References: https://chatgpt.com/