**DatabaseManagementAssignment:SakilaDatabaseinMySQLWorkbench**

# Objective

This assignment aims to enhance your understanding of database management by performingvariousoperationsonthe**Sakila**databaseusing**MySQLWorkbench**.Youwill modify,insert,delete,andupdaterecords,designcomplexqueries,workwithtransactions, and ensure data integrity and consistency.

# AssignmentTasks

## 1. IdentifyingToolsandStatementsforModifyingDatabaseContent

- Researchanddocumentdifferent**SQLstatements**usedtomodifydatabasecontent (INSERT, UPDATE, DELETE, ALTER, etc.).
- DescribethefunctionalitiesofMySQLWorkbenchtoolssuchas**SQLEditor, Schema Inspector, and Query Builder**.
- **Deliverable:**AsectioninthefinalreportsummarizingSQLstatementsandMySQL Workbench tools.

```
mysql> exit
Bye

C:\Users\DAM2_Diurno\Alex\sakila-db>echo "INSERT: Agrega registros..." > 01_sql_summary.txt

C:\Users\DAM2_Diurno\Alex\sakila-db>echo "UPDATE: Modifica registros existentes..." >> 01_sql_summary.txt

C:\Users\DAM2_Diurno\Alex\sakila-db>echo "DELETE: Elimina registros..." >> 01_sql_summary.txt
```

## 2. DataInsertion,Deletion,andUpdate

- Usingthe**actor**table,insertanewrecordwithfictitiousdata.
- Updateanexistingrecordbychangingthelastnameofanactor.
- Deleteanactorfromthetable.
- **Deliverable:**ProvidetheexecutedSQLstatementsandtheirresults.
    - **SQLFile:**`02_modify_actor.sql`
    - **Screenshots:**`02_modify_actor_screenshots/`

\

```
mysql> USE sakila;
Database changed
mysql>
mysql> -- Insertar un nuevo actor
mysql> INSERT INTO actor (first_name, last_name, last_update)
    -> VALUES ('John', 'Doe', NOW());
Query OK, 1 row affected (0.01 sec)

mysql>
mysql> -- Actualizar el apellido de un actor existente
mysql> UPDATE actor SET last_name = 'Smith' WHERE first_name = 'John' AND last_name = 'Doe';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> -- Eliminar un actor
mysql> DELETE FROM actor WHERE first_name = 'John' AND last_name = 'Smith';
Query OK, 1 row affected (0.00 sec)

mysql>
```

```
C:\Users\DAM2_Diurno\Alex\sakila-db>echo -- Modificación de actor > 02_modify_actor.sql

C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p sakila < 02_modify_actor.sql
Enter password: ****

C:\Users\DAM2_Diurno\Alex\sakila-db>
```

## 3. CreatingaTablefromaQueryResult

- Executeaquerytoretrieveallmoviesreleasedafter2005fromthe**film**table.
- Storetheresultinanewtablecalled**recent_films**.
- **Deliverable:**TheSQLscriptusedandascreenshotofthenewlycreatedtable.
  - **SQLFile:**`03_create_recent_films.sql`
  - **Screenshot:**`03_recent_films_screenshot.png`

```
C:\Users\DAM2_Diurno\Alex\sakila-db>echo -- recent films > 03_create_recent_films.sql

C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p sakila < 03_create_recent_films.sql
Enter password: ****

C:\Users\DAM2_Diurno\Alex\sakila-db>
```

```
mysql> USE sakila
Database changed
mysql> CREATE TABLE recent_films AS
    -> SELECT * FROM film WHERE release_year > 2005;
ERROR 1050 (42S01): Table 'recent_films' already exists
mysql>
```

```
C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p sakila < C:\Users\DAM2_Diurno\Alex\sakila-db\03_create_recent_fi
lms.sql
Enter password: ****

C:\Users\DAM2_Diurno\Alex\sakila-db>
```

## 4. DesigningComplexSQLScripts

- WriteanSQLscriptthat:
  - Listsallcustomerswhohaverentedafilminthelast30days.
  - Identifiesthemostrentedfilminthedatabase.
  - Displaysthetotalrevenuegeneratedperstore.
- **Deliverable:**TheSQLscriptwithcommentsexplainingeachquery.
  - **SQLFile:**`04_complex_queries.sql`

UnderstandingTransactions

- Explaintransactionsandtheirimportanceindatabasemanagement.
- UsingMySQLWorkbench,performatransactionthat:
  - Insertsanewrentalrecord.
  - Updatestheinventorytoreflecttherental.
  - Commitsthetransaction.
- **Deliverable:**
  - **SQLFile:**`05_transaction_example.sql`
  - **Sectioninthefinalreportcoveringtransactions.**

```
mysql> USE sakila;
Database changed
mysql>
mysql> -- Clientes que alquilaron en los últimos 30 días
mysql> SELECT DISTINCT customer_id FROM rental WHERE rental_date >= NOW() - INTERVAL 30 DAY;
Empty set (0.00 sec)

mysql>
mysql> -- Película más alquilada
mysql> SELECT film_id, COUNT(*) as rentals FROM rental
    -> JOIN inventory USING(inventory_id)
    -> GROUP BY film_id
    -> ORDER BY rentals DESC
    -> LIMIT 1;
+---------+---------+
| film_id | rentals |
+---------+---------+
|     103 |      34 |
+---------+---------+
1 row in set (0.01 sec)

mysql>
mysql> -- Ingresos totales por tienda
mysql> SELECT store_id, SUM(amount) as total_revenue FROM payment
    -> JOIN staff USING(staff_id)
    -> GROUP BY store_id;
+----------+---------------+
| store_id | total_revenue |
+----------+---------------+
|        1 |      33482.50 |
|        2 |      33924.06 |
+----------+---------------+
2 rows in set (0.02 sec)

mysql>
```

```
C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p sakila < 04_complex_queries.sql
Enter password: ****

C:\Users\DAM2_Diurno\Alex\sakila-db>
```

```
C:\Users\DAM2_Diurno\Alex\sakila-db>echo -- transaction example > 05_transaction_example.sql

C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE sakila;
Database changed
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO rental (rental_date, inventory_id, customer_id, return_date, staff_id, last_update)
    -> VALUES (NOW(), 1, 1, NULL, 1, NOW());
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> UPDATE inventory SET last_update = NOW() WHERE inventory_id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql>
```

```
C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p sakila < 05_transaction_example.sql
Enter password: ****
```

## 5. RollingBackTransactions

- Demonstrateascenariowhereatransactionispartiallyexecutedbutlaterrolledback due to an error (e.g., an attempt to rent a movie that is out of stock).
- **Deliverable:**
  - **SQLFile:**`06_rollback_example.sql`
  - **Sectioninthefinalreportcoveringrollbacktransactions.**

```
mysql> USE sakila;
Database changed
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> INSERT INTO rental (rental_date, inventory_id, customer_id, return_date, staff_id, last_update)
    -> VALUES (NOW(), 9999, 1, NULL, 1, NOW());
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`sakila`.`rental`, CONSTRAINT `
fk_rental_inventory` FOREIGN KEY (`inventory_id`) REFERENCES `inventory` (`inventory_id`) ON DELETE RESTRICT ON UPDAT
E CASCADE)
mysql>
mysql> -- Si la película no existe, hacer rollback
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)
```

```
C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p sakila < 06_rollback_example.sql
Enter password: ****
```

## 6. UnderstandingRecordLockingPolicies

- Researchanddocumentdifferenttypesofrecord-lockingmechanisms(pessimistic vs. optimistic locking).
- Testascenariowheretwousersattempttoupdatethesamerecordsimultaneously.
- **Deliverable:**
  - **Sectioninthefinalreportcoveringrecordlockingpolicies.**

```
C:\Users\DAM2_Diurno\Alex\sakila-db>echo "Pessimistic Locking: Bloquea el registro antes de modificarlo" > 07_locking_policies.txt

C:\Users\DAM2_Diurno\Alex\sakila-db>echo "Optimistic Locking: Permite modificar pero verifica que no haya cambios previos" >> 07_locking_policies.txt
```

## 7. EnsuringDataIntegrityandConsistency

- IdentifypotentialdataintegrityissuesintheSakiladatabase.
- Implementforeignkeyconstraintsandtriggerstomaintaindataconsistency.
- **Deliverable:**
  - **SQLFile:**`08_data_integrity.sql`
  - **Sectioninthefinalreportcoveringdataintegrity.**

```
C:\Users\DAM2_Diurno\Alex\sakila-db>echo -- data integrity > 08_data_integrity.sql
```

```
mysql> USE sakila;
Database changed
mysql>
mysql> -- Asegurar integridad con una clave foránea
mysql> ALTER TABLE rental ADD CONSTRAINT fk_inventory FOREIGN KEY (inventory_id) REFERENCES inventory (inventory_id);

Query OK, 16045 rows affected (0.52 sec)
Records: 16045  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Crear un trigger para evitar alquileres de películas sin stock
mysql> DELIMITER $$
mysql> CREATE TRIGGER before_rental_insert
    -> BEFORE INSERT ON rental
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE stock INT;
    ->     SELECT COUNT(*) INTO stock FROM inventory WHERE inventory_id = NEW.inventory_id;
    ->     IF stock = 0 THEN
    ->         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No stock available';
    ->     END IF;
    -> END;
    -> $$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

```
C:\Users\DAM2_Diurno\Alex\sakila-db>mysql -u root -p sakila < 08_data_integrity.sql
Enter password: ****
```

# SubmissionRequirements

- **FolderStructure:**
  - `Deliverables/`
    - `SQL_Scripts/`(Containsall`.sql`files)
    - `Screenshots/`(Containsscreenshotsofqueryexecutions,if applicable)
    - `Final_Report/`(ContainsasingleconsolidatedreportinPDF format)
- **FinalReport:**Allpreviouslyseparatereportsshouldnowbeconsolidatedintoa single PDF file:
  - **Filename:**`Surname_Name_Final_Report_Sakila.pdf`
  - **Folder:**`Deliverables/Final_Report/`
  - Thereportmustcontainallresearch,explanations,queryresults,and screenshots.

---

## Notes:

- EnsurethatallSQLstatementsaretestedbeforesubmission.
- Thefinalreportshouldincludereflectionsonchallengesfacedandhowtheywere overcome.
- Citeanyexternalreferencesusedinyourresearch.