**PABLO CHICO MERINO**

**PROYECTO 2: SQL**

**Análisis de la Base de Datos de una Tienda de Películas 🎬**

Este repositorio reúne **64 ejercicios resueltos en SQL** sobre una base de datos relacional de una tienda de alquiler de películas. El objetivo es **practicar y dominar consultas** desde nivel básico hasta avanzado, cubriendo escenarios analíticos reales: ventas, inventario, comportamiento de clientes, catálogo y rendimiento del negocio audiovisual. Los ejercicios principales se encuentran en queries/EJERCICIOS\_SQL.sql (1–64).

EJERCICIOS SQL

**Tabla de contenidos**

* [Descripción del proyecto](#descripci%C3%B3n-del-proyecto)
* [Stack y compatibilidad](#stack-y-compatibilidad)
* [Estructura del repositorio](#estructura-del-repositorio)
* [Instalación y puesta en marcha](#instalaci%C3%B3n-y-puesta-en-marcha)
  + - [PostgreSQL](#postgresql)
    - [MySQL](#mysql)
* [Guía de ejecución](#gu%C3%ADa-de-ejecuci%C3%B3n)
* [Qué aprendemos](#qu%C3%A9-aprender%C3%A1s)
* [Resultados y ejemplos](#resultados-y-ejemplos)
* [Buenas prácticas y estilo SQL](#buenas-pr%C3%A1cticas-y-estilo-sql)
* [Pruebas y validación](#pruebas-y-validaci%C3%B3n)
* [Rendimiento y optimización](#rendimiento-y-optimizaci%C3%B3n)
* [Roadmap / Próximos pasos](#roadmap--pr%C3%B3ximos-pasos)
* [Contribuciones](#contribuciones)
* [Autores y agradecimientos](#autores-y-agradecimientos)

**Descripción del proyecto**

A partir de un esquema tipo “**Sakila**”/tienda de alquiler de películas, se plantean y resuelven consultas que abarcan:

* **SELECT, WHERE, ORDER BY, LIMIT, OFFSET**
* **Funciones de agregación**: SUM, AVG, COUNT, MIN, MAX, VARIANCE, STDDEV
* **Agrupación y filtrado**: GROUP BY, HAVING
* **Relaciones**: INNER | LEFT | FULL JOIN, CROSS JOIN (con discusión de utilidad)
* **Subconsultas** (incluidas **correlacionadas**)
* **Vistas (VIEW)**, **CTE** (Common Table Expressions) y **tablas temporales**
* **Dimensiones de negocio**: catálogo (films, actores, categorías), clientes, inventario, alquileres, pagos e ingresos

Los 64 ejercicios están comentados y numerados, con salidas esperadas cuando es útil para el aprendizaje.

EJERCICIOS SQL

**Stack y compatibilidad**

* **Recomendado**: **PostgreSQL 14+**  
  El fichero de ejercicios usa **comillas dobles** para identificadores ("tabla"."columna"), típica de Postgres. Algunas funciones agregadas como VARIANCE y STDDEV están soportadas nativamente.

EJERCICIOS SQL

* **Alternativa**: **MySQL 8.0**
* **Cliente SQL**: DBeaver (o psql/mysql CLI).

**Estructura del repositorio**

TiendaPeliculas\_SQL/

├─ data/

│ ├─ schema.sql # Esquema (film, actor, rental, etc.)

│ └─ data\_sample.sql # Carga de datos de ejemplo

│

├─ queries/

│ ├─ EJERCICIOS\_SQL.sql # Ejercicios 1–64 comentados

│ ├─ consultas\_basicas.sql # SELECT, WHERE, ORDER BY, LIMIT

│ └─ consultas\_avanzadas.sql # JOIN, GROUP BY, HAVING, subconsultas, CTE

│

├─ results/

│ ├─ ejemplos\_salida.csv # Resultados esperados (selectos)

│ └─ capturas/ # Screenshots de cliente SQL

│

└─ README.md

Sugerencia: añade un docs/ con un **diagrama ER** (docs/ERD.png) para facilitar la navegación del modelo de datos.

**Instalación y puesta en marcha**

**PostgreSQL**

1. Crear base de datos:

CREATE DATABASE tienda\_peliculas;

1. Cargar esquema y datos:

psql -d tienda\_peliculas -f data/schema.sql

psql -d tienda\_peliculas -f data/data\_sample.sql

1. Abrir los ejercicios:

* Con DBeaver/psql, ejecuta bloques desde queries/EJERCICIOS\_SQL.sql.

**MySQL**

1. Crear base de datos:

CREATE DATABASE tienda\_peliculas;

1. Importar:

mysql -D tienda\_peliculas < data/schema.sql

mysql -D tienda\_peliculas < data/data\_sample.sql

**Guía de ejecución**

* Ejecuta los ejercicios **en orden** (1→64) o por temática:
  + - **Básicos**: 2–10, 12–16, 35–41, 40
    - **Agregaciones / Grouping**: 7, 13, 20–21, 23–27, 28–31, 41, 47–52, 61–64
    - **JOIN / Relaciones**: 17–20, 29–33, 42–47, 50–56, 58, 61–64
    - **Subconsultas**: 24, 27, 55, 59, 60
    - **Vistas / CTE / Temporales**: 48, 51, 52
* Cada bloque está comentado con su **enunciado** y, cuando procede, notas de interpretación.

EJERCICIOS SQL

**Qué aprendemos**

* Convertir **preguntas de negocio** en consultas relacionales.
* Diferenciar y combinar **JOINs** según el caso de uso.
* Aplicar **agregaciones** y filtros con HAVING.
* Escribir **subconsultas** eficaces (y cuándo evitarlas).
* Estructurar **CTE** para hacer más legible el SQL.
* Crear **vistas** reutilizables para reporting.
* Medir **ingresos**, **actividad de clientes** y **rotación de inventario**.

**Resultados y ejemplos**

**1) Top películas por recaudación (muestra de idea):**

SELECT category.name AS categoria, f.title AS pelicula, SUM(p.amount) AS recaudacion

FROM film f

JOIN inventory i ON f.film\_id = i.film\_id

JOIN rental r ON i.inventory\_id = r.inventory\_id

JOIN payment p ON r.rental\_id = p.rental\_id

JOIN film\_category fc ON f.film\_id = fc.film\_id

JOIN category ON fc.category\_id = category.category\_id

GROUP BY category.name, f.title

ORDER BY recaudacion DESC

LIMIT 10;

**2) Clientes más activos (alquileres):**

SELECT c.first\_name, c.last\_name, COUNT(r.rental\_id) AS total\_alquileres

FROM customer c

JOIN rental r ON c.customer\_id = r.customer\_id

GROUP BY c.customer\_id

ORDER BY total\_alquileres DESC

LIMIT 5;

**3) Actores con más participaciones:**

SELECT a.first\_name, a.last\_name, COUNT(fa.film\_id) AS num\_peliculas

FROM actor a

JOIN film\_actor fa ON a.actor\_id = fa.actor\_id

GROUP BY a.actor\_id

ORDER BY num\_peliculas DESC

LIMIT 10;

En queries/EJERCICIOS\_SQL.sql encontrarás además ejemplos con **CTE**, **vistas** y **subconsultas** para rankings, comparativas temporales y métricas de negocio.

EJERCICIOS SQL

**Buenas prácticas y estilo SQL**

* **Consistencia** en mayúsculas para palabras clave (SELECT, FROM, WHERE). (Aunque en DBeaver corrige a minúsculas).
* **Identificadores** con comillas dobles en Postgres; **backticks** en MySQL.
* Usa **alias descriptivos** y evita SELECT \* en producción.
* Documenta los **supuestos de negocio** en comentarios SQL ( -- ....)

**Pruebas y validación**

* **Resultados esperados**: consulta results/ejemplos\_salida.csv para comparar selectos outputs.
* **Smoke tests**: ejecuta un subconjunto de queries tras cargar datos para validar:
  + - Integridad de joins (sin duplicidades inesperadas).
    - Coherencia de fechas (rental\_date ≤ return\_date cuando aplique).
    - Totales de pagos ≈ suma por cliente/categoría.

**Rendimiento y optimización**

* Asegurar **índices** en claves foráneas frecuentes (p.ej., rental.inventory\_id, inventory.film\_id, payment.rental\_id).
* Revisa planes con EXPLAIN (ANALYZE, BUFFERS) (Postgres) o EXPLAIN ANALYZE (MySQL).
* Considera **materializar vistas** (Postgres: *materialized views*) para informes pesados.
* Evitar subconsultas correlacionadas si un **JOIN + GROUP BY** ofrece el mismo resultado.

**Roadmap / Próximos pasos**

* **Funciones de ventana**: ROW\_NUMBER(), RANK(), DENSE\_RANK(), NTILE()
* **Procedimientos / Triggers** para auditoría de alquileres y stock
* **Versión SQLite** con adaptación de sintaxis
* **Índices compuestos** y **particionado** de tablas de gran volumen
* **Data Quality Checks** automatizados (p. ej., con dbt tests ligeros)

**Contribuciones**

Las contribuciones son bienvenidas

1. **Fork** del repo
2. Crea rama: feature/nueva-consulta
3. Pull Request describiendo:
   * + problema de negocio,
     + enfoque SQL,
     + evidencia (captura/CSV pequeño) y
     + notas de compatibilidad (PG/MySQL).

Incluye tests mínimos (consulta + resultado reducido) en results/.

**Autores y agradecimientos**

**Autor principal**:

**Pablo Chico Merino**  
[GitHub](https://github.com/PabloChico01)