UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Projet SQL
Requêtes Sakila – Technique en Entreprise

Hoang Thuy Duong Vu
Johan Ghré
Céline Chen
Inès Hummel

Octobre 2024

Projet réalisé sous la direction de M. Benoît Grand

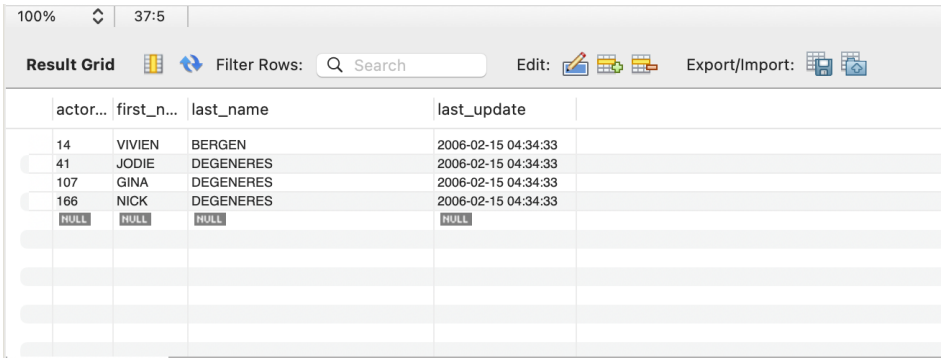
Code source du Projet

Table de matière

1 1ère partie – Base de données Sakila

1. Tous les acteurs dont le nom de famille contient les lettres ‘gen’

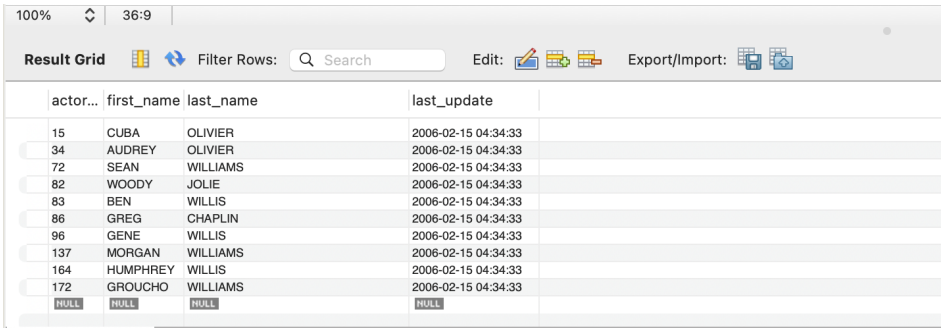
```
SELECT
    *
FROM
    SAKILA.ACTOR
WHERE
    LOWER(LAST_NAME) LIKE '%gen%';
```



actor...	first_n...	last_name	last_update
14	VIVIEN	BERGEN	2006-02-15 04:34:33
41	JODIE	DEGENERES	2006-02-15 04:34:33
107	GINA	DEGENERES	2006-02-15 04:34:33
166	NICK	DEGENERES	2006-02-15 04:34:33
NULL	NULL	NULL	NULL

2. Tous les acteurs dont le nom de famille contient les lettres ‘li’

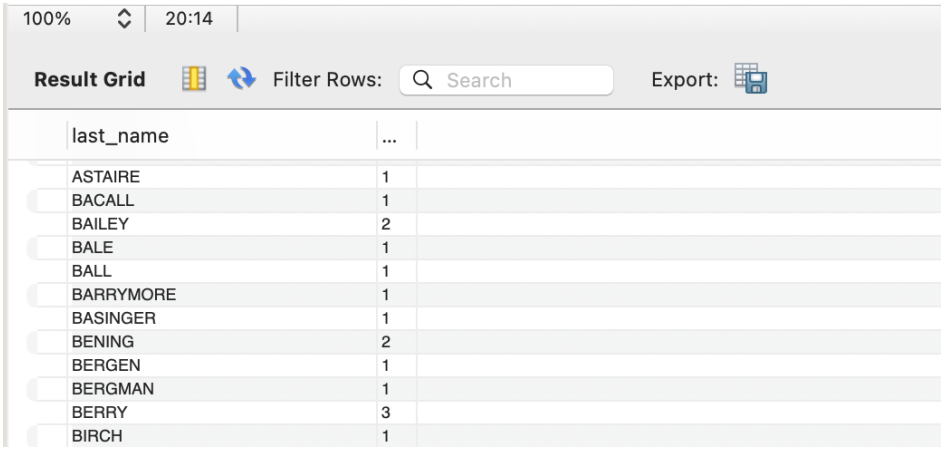
```
SELECT
    *
FROM
    SAKILA.ACTOR
WHERE
    LOWER(LAST_NAME) LIKE '%li%';
```



actor...	first_name	last_name	last_update
15	CUBA	OLIVIER	2006-02-15 04:34:33
34	AUDREY	OLIVIER	2006-02-15 04:34:33
72	SEAN	WILLIAMS	2006-02-15 04:34:33
82	WOODY	JOLIE	2006-02-15 04:34:33
83	BEN	WILLIS	2006-02-15 04:34:33
86	GREG	CHAPLIN	2006-02-15 04:34:33
96	GENE	WILLIS	2006-02-15 04:34:33
137	MORGAN	WILLIAMS	2006-02-15 04:34:33
164	HUMPHREY	WILLIS	2006-02-15 04:34:33
172	GROUCHO	WILLIAMS	2006-02-15 04:34:33
NULL	NULL	NULL	NULL

3. Liste des noms de famille de tous les acteurs, ainsi que le nombre d’acteurs portant chaque nom de famille

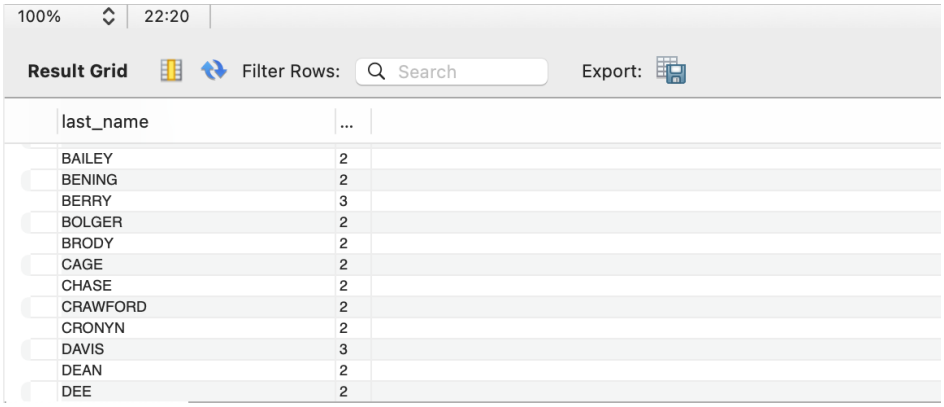
```sql
SELECT
    LAST_NAME,
    COUNT(*)
FROM
    SAKILA.ACTOR
GROUP BY
    LAST_NAME;
```



last_name	...
ASTAIRE	1
BACALL	1
BAILEY	2
BALE	1
BALL	1
BARRYMORE	1
BASINGER	1
BENING	2
BERGEN	1
BERGMAN	1
BERRY	3
BIRCH	1

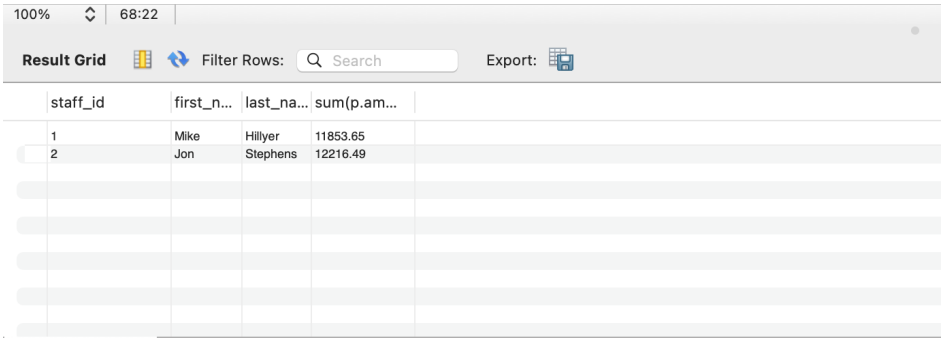
4. Liste des noms de famille des acteurs et le nombre d'acteurs qui portent chaque nom de famille, mais seulement pour les noms qui sont portés par au moins 2 acteurs

```sql
SELECT
    LAST_NAME,
    COUNT(*)
FROM
    SAKILA.ACTOR
GROUP BY
    LAST_NAME
HAVING
    COUNT(*)  >= 2;
```



5. Utilisez JOIN pour afficher le montant total perçu par chaque membre du personnel en août 2005

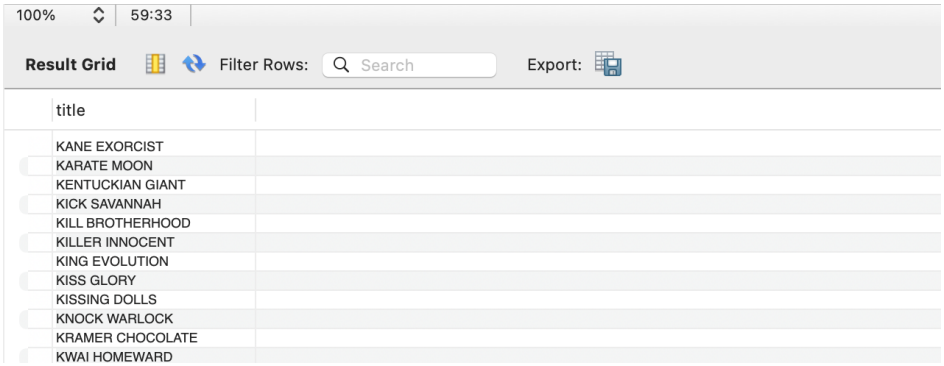
```
SELECT
    P.STAFF_ID,
    S.FIRST_NAME,
    S.LAST_NAME,
    SUM(P.AMOUNT)
FROM
    SAKILA.STAFF S
    JOIN SAKILA.PAYMENT P
    ON S.STAFF_ID = P.STAFF_ID
WHERE
    DATE_FORMAT(P.PAYMENT_DATE, '%Y-%m') = '2005-08'
GROUP BY
    P.STAFF_ID, S.FIRST_NAME, S.LAST_NAME;
```



6. Afficher les titres des films commençant par les lettres K et Q dont la langue est l'anglais

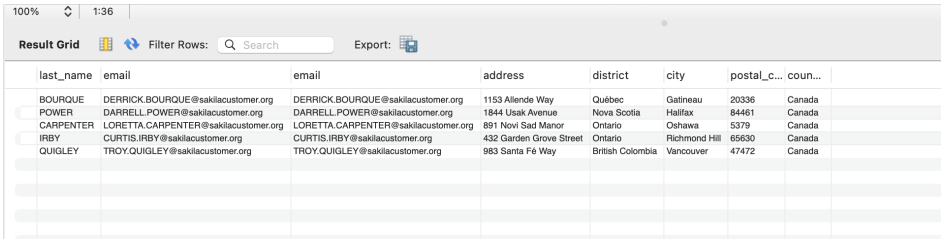
```
SELECT DISTINCT F.TITLE
FROM SAKILA.FILM F, SAKILA.LANGUAGE L
```

```
WHERE
    LOWER(F.TITLE) LIKE 'k%' OR
    LOWER(F.TITLE) LIKE 'q%'
    AND F.LANGUAGE_ID = L.LANGUAGE_ID
    AND L.NAME = 'English';
```



7. Affichez les noms et les adresses électroniques de tous les clients canadiens

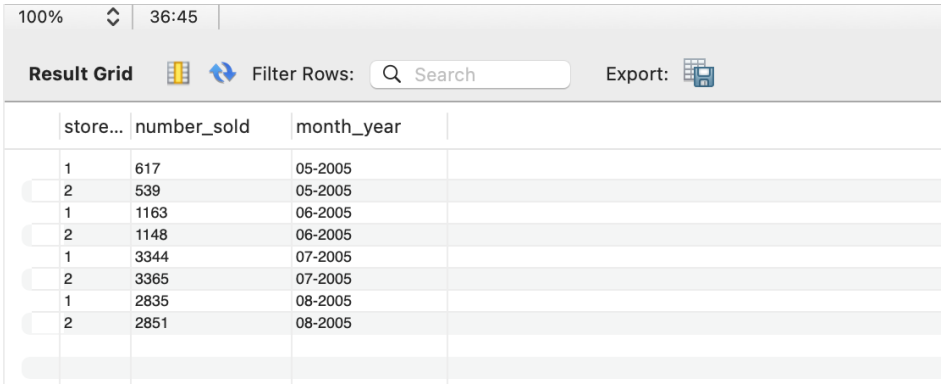
```
SELECT
    C.LAST_NAME,
    C.EMAIL,
    A.ADDRESS,
    A.DISTRICT,
    CI.CITY,
    A.POSTAL_CODE,
    CO.COUNTRY
FROM
    SAKILA.CUSTOMER C,
    SAKILA.CITY CI,
    SAKILA.ADDRESS A,
    SAKILA.COUNTRY CO
WHERE
    C.ADDRESS_ID = A.ADDRESS_ID
    AND CI.CITY_ID = A.CITY_ID
    AND CO.COUNTRY_ID = CI.COUNTRY_ID
    AND CO.COUNTRY = 'Canada';
```



last_name	email	email	address	district	city	postal_c...	coun...
BOURQUE	DERRICK.BOURQUE@sakilacustomer.org	DERRICK.BOURQUE@sakilacustomer.org	1153 Allende Way	Québec	Gatineau	20336	Canada
POWER	DARRELL.POWER@sakilacustomer.org	DARRELL.POWER@sakilacustomer.org	1844 Usak Avenue	Nova Scotia	Halifax	84461	Canada
CARPENTER	LORETTA.CARPENTER@sakilacustomer.org	LORETTA.CARPENTER@sakilacustomer.org	891 Novi Sad Manor	Ontario	Oshawa	5379	Canada
IRBY	CURTIS.IRBY@sakilacustomer.org	CURTIS.IRBY@sakilacustomer.org	432 Garden Grove Street	Ontario	Richmond Hill	65630	Canada
QUIGLEY	TROY.QUIGLEY@sakilacustomer.org	TROY.QUIGLEY@sakilacustomer.org	983 Santa Fé Way	British Colombia	Vancouver	47472	Canada

8. Quelles sont les ventes de chaque magasin pour chaque mois de 2005 (CONCAT)

```
SELECT
    S.STORE_ID,
    COUNT(P.PAYMENT_ID) AS NUMBER_SOLD,
    CONCAT('0', MONTH(P.PAYMENT_DATE), '-', YEAR(P.PAYMENT_DATE)) AS MONTH_YEAR
FROM
    SAKILA.PAYMENT P, SAKILA.STAFF ST, SAKILA.STORE S
WHERE
    P.STAFF_ID = ST.STAFF_ID
    AND ST.STORE_ID = S.STORE_ID
    AND YEAR(P.PAYMENT_DATE) = 2005
GROUP BY
    CONCAT('0', MONTH(P.PAYMENT_DATE), '-', YEAR(P.PAYMENT_DATE)), S.STORE_ID
ORDER BY
    CONCAT('0', MONTH(P.PAYMENT_DATE), '-', YEAR(P.PAYMENT_DATE)) ASC;
```

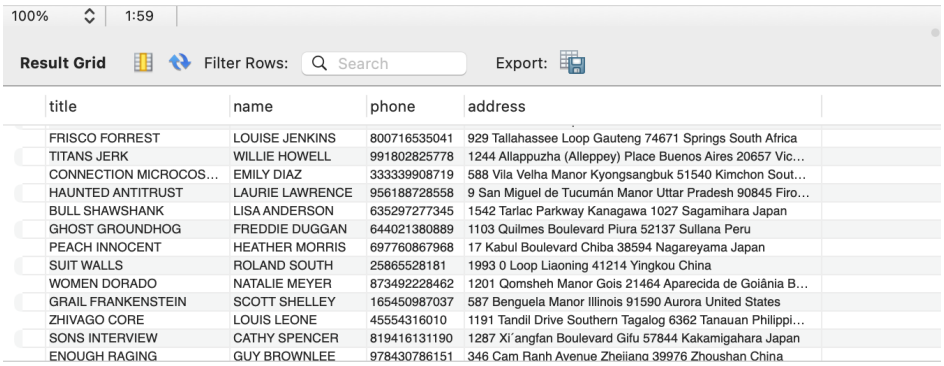


store...	number_sold	month_year
1	617	05-2005
2	539	05-2005
1	1163	06-2005
2	1148	06-2005
1	3344	07-2005
2	3365	07-2005
1	2835	08-2005
2	2851	08-2005

9. Trouvez le titre du film, le nom du client, le numéro de téléphone du client et l'adresse du client pour tous les DVD en circulation (qui n'ont pas prévu d'être rendus)

```
SELECT DISTINCT
    F.TITLE,
    CONCAT(C.FIRST_NAME, ' ', C.LAST_NAME) AS NAME, A.PHONE,
```

```
    CONCAT(A.ADDRESS, ' ', A.DISTRICT, ' ', A.POSTAL_CODE, ' ', CI.CITY, ' ', CO.COUNTRY
FROM
    SAKILA.RENTAL R
    JOIN SAKILA.CUSTOMER C
        ON R.CUSTOMER_ID = C.CUSTOMER_ID
    JOIN SAKILA.ADDRESS A
        ON A.ADDRESS_ID = C.ADDRESS_ID
    JOIN SAKILA.INVENTORY I
        ON I.INVENTORY_ID = R.INVENTORY_ID
    JOIN SAKILA.FILM F
        ON F.FILM_ID = I.FILM_ID
    JOIN SAKILA.CITY CI
        ON CI.CITY_ID = A.CITY_ID
    JOIN SAKILA.COUNTRY CO
        ON CO.COUNTRY_ID = CI.COUNTRY_ID
WHERE R.RETURN_DATE IS NULL;
```



100% 1:59

Result Grid Filter Rows: Search Export:

title	name	phone	address
FRISCO FORREST	LOUISE JENKINS	800716535041	929 Tallahassee Loop Gauteng 74671 Springs South Africa
TITANS JERK	WILLIE HOWELL	991802825778	1244 Allappuzha (Alleppey) Place Buenos Aires 20657 Vic…
CONNECTION MICROCOS…	EMILY DIAZ	333339908719	588 Vila Velha Manor Kyongsangbuk 51540 Kimchon Sout…
HAUNTED ANTITRUST	LAURIE LAWRENCE	956188728558	9 San Miguel de Tucumán Manor Uttar Pradesh 90845 Firo…
BULL SHAWSHANK	LISA ANDERSON	635297277345	1542 Tarlac Parkway Kanagawa 1027 Sagamihara Japan
GHOST GROUNDHOG	FREDDIE DUGGAN	644021380889	1103 Quilmes Boulevard Piura 52137 Sullana Peru
PEACH INNOCENT	HEATHER MORRIS	697760867968	17 Kabul Boulevard Chiba 38594 Nagareyama Japan
SUIT WALLS	ROLAND SOUTH	25865528181	1993 0 Loop Liaoning 41214 Yingkou China
WOMEN DORADO	NATALIE MEYER	873492228462	1201 Qomsheh Manor Gois 21464 Aparecida de Goiânia B…
GRAIL FRANKENSTEIN	SCOTT SHELLEY	165450987037	587 Benguela Manor Illinois 91590 Aurora United States
ZHIVAGO CORE	LOUIS LEONE	45554316010	1191 Tandil Drive Southern Tagalog 6362 Tanauan Philippi…
SONS INTERVIEW	CATHY SPENCER	819416131190	1287 Xi´angfan Boulevard Gifu 57844 Kakamigahara Japan
ENOUGH RAGING	GUY BROWNLEE	978430786151	346 Cam Ranh Avenue Zhejiang 39976 Zhoushan China

2 2è partie – Test technique (type Entreprise)

1. How can SQL queries be optimized ?

   12 best practices :

   - Use indexes effectively
   - Avoid `SELECT *` and retrieve only necessary columns
   - Optimize `JOIN` operations
   - Minimize the use of subqueries
   - Avoid redundant or unnecessary data retrieval
   - Utilize stored procedures

- Consider partitioning and sharding
- Normalize database tables
- Monitor query performance
- Use `UNION ALL` instead of `UNION`
- Optimize subquery performance
- Leverage cloud database-specific features

2. How do you remove duplicate rows from a table ?

   - **Method 1 :** Run the following script :

     ```
     SELECT DISTINCT *
     INTO duplicate_table
     FROM original_table
     GROUP BY key_value
     HAVING COUNT(key_value) > 1;

     DELETE original_table
     WHERE key_value
     IN (SELECT key_value
     FROM duplicate_table);

     INSERT original_table
     SELECT *
     FROM duplicate_table;

     DROP TABLE duplicate_table;
     ```

     This script takes the following actions in the given order:

     – Moves one instance of any duplicate row in the original table to a duplicate table.
     – Deletes all rows from the original table that are also located in the duplicate table.
     – Moves the rows in the duplicate table back into the original table.
     – Drops the duplicate table.

   - **Method 2 :** The `ROW_NUMBER` function that was introduced in Microsoft SQL Server 2005 makes this operation much simpler:

```
DELETE T
FROM
(
    SELECT *,
     DupRank = ROW_NUMBER() OVER (
        PARTITION BY key_value
        ORDER BY (SELECT NULL)
    )
FROM original_table
) AS T
WHERE DupRank > 1;
```

- **OTHER METHODS:**
  - SQL delete duplicate Rows using Group By and Having clause –> Use the SQL GROUP BY clause to identify the duplicate rows. The Group By clause groups data as per the defined columns and we can use the COUNT function to check the occurrence of a row;
  - SQL delete duplicate Rows using Common Table Expressions (CTE) –> Use Common Table Expressions commonly known as CTE to remove duplicate rows in SQL Server. It is available starting from SQL Server 2005;
  - RANK function to SQL delete duplicate rows –> Use the SQL RANK function to remove the duplicate rows as well. SQL RANK function gives unique row ID for each row irrespective of the duplicate row;
  - Use SSIS package to SQL delete duplicate rows –> Use Sort Operator in an SSIS package for removing duplicating rows.

3. What are the main differences between HAVING and WHERE SQL clauses ?

   The main difference between WHERE and HAVING clause is that the WHERE clause allows you to filter data from specific rows (individual rows) from a table based on certain conditions.

   In contrast, the HAVING clause allows you to filter data from a group of rows in a query based on conditions involving aggregate values.

Where	Having
filters by each row	filters by each group
processed before any grouping	processed after any grouping
cannot have aggregate functions	can have aggregate functions
can be used in SELECT, INSERT, UPDATE, DELETE statements	can only be used in SELECT statements
written before GROUP BY clause	written after GROUP BY clause

4. What is the difference between normalization and denormalization ?

   The goal of normalization is to minimize data redundancy and dependency by organizing data into well-structured tables.

   Denormalization involves combining tables that have been normalized to improve query performance and simplify data retrieval.

	Normalization	Denormalization
Implementation	Decomposes data into different tables to reduce redundancy	Combines data to improve the access time
Query execution speed	Speed of update, delete and write operations is higher	Speed of read operations is higher, but that of update and write operations is slower
Memory consumption	Memory consumption is less as data redundancy is less	Memory consumption is more as redundancy is introduced
Number of tables	Number of tables is more on account of decomposition of data	Combines data and hence number of tables are less
Data integrity	Data integrity is maintained	Data integrity might not be maintained

5. What are the key differences between the DELETE and TRUNCATE SQL commands ?

DELETE is a SQL command that removes one or multiple rows from a table using conditions.

TRUNCATE is a SQL command that removes all the rows from a table without using any condition.

Truncate	Delete
It removes all rows from a table + faster + does not use as much undo space as a delete	It is used to remove rows from table. A WHERE clause can be used to only remove some rows
It is a DDL command so this command change structure of table	It is a DML command. It only remove rows from a table, leaving the table structure untouched
You cannot rollback in Truncate	In DELETE, you can rollback
In SQL, the auto increment counter gets reset with truncate	The auto increment counter cannot get reset with delete

6. What are some ways to prevent duplicate entries when making a query?

   To prevent duplicate entries when making a query, you can consider the following approaches:

   1. **Use `DISTINCT`**: When selecting data from a database, include the `DISTINCT` keyword in your SQL query. This will return only unique rows based on the columns you specify.

```
SELECT DISTINCT column1, column2 FROM table_name;
```

   2. **Use Grouping**: Utilize the `GROUP BY` clause to group records by specific columns. This will aggregate results and can be combined with aggregate functions to eliminate duplicates.

```
SELECT column1, COUNT(*) FROM table_name GROUP BY column1;
```

   3. **Implement Primary Keys and Unique Constraints**: In your database schema, define primary keys and unique constraints on columns where duplicates should not occur. This will automatically prevent the insertion of duplicate records.

   4. **Use `JOIN` with Care**: When joining tables, ensure you are joining on the correct keys and using appropriate conditions to avoid unintended duplication of rows.

5. **Window Functions**: Use window functions to rank or number rows based on specific criteria, then filter out duplicates based on that ranking.

```
SELECT * FROM (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY column1 ORDER BY column2) as rn
    FROM table_name
) AS temp
WHERE rn = 1;
```

7. What are the different types of relationships in SQL?

   In SQL and relational database design, there are three main types of relationships that define how tables are related to each other:

   1. **One-to-One (1:1)**:
      - In a one-to-one relationship, a row in one table is linked to a single row in another table, and vice versa.
      - This type of relationship is often used to split a table for normalization or to separate optional data.
      - **Example**: A `Users` table and a `UserProfiles` table, where each user has exactly one profile.
   2. **One-to-Many (1:N)**:
      - In a one-to-many relationship, a row in one table can be associated with multiple rows in another table, but a row in the second table is linked to only one row in the first table.
      - This is the most common type of relationship in relational databases.
      - **Example**: A `Customers` table and an `Orders` table, where each customer can have multiple orders, but each order is associated with only one customer.
   3. **Many-to-Many (M:N)**:
      - In a many-to-many relationship, multiple rows in one table can be related to multiple rows in another table. This type of relationship requires a junction (or linking) table to facilitate the relationship.
      - **Example**: A `Students` table and a `Courses` table, where a student can enroll in multiple courses, and a course can have multiple students. A junction table called `Enrollments` could be created to link students and courses.
   4. **Additional Concepts**:

- **Self-Referencing Relationships**: A table may relate to itself in a one-to-one or one-to-many manner. For example, an `Employees` table where each employee can have a manager who is also an employee.
- **Foreign Keys**: Relationships are typically enforced using foreign keys, which are fields in one table that refer to the primary key of another table. This ensures referential integrity.

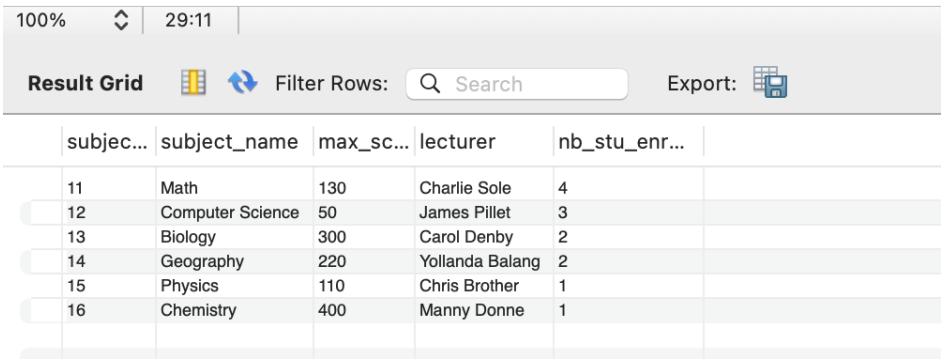
8. SQL code example + Queries :

    1. Give an example of the SQL code that will insert the 'Input data' into the two tables. You must ensure that the student table includes the correct [dbo].[Master].[id] in the [dbo].[student].[Master_id] column.

```
WITH new_master AS (
    INSERT INTO [dbo].[Master] (name, some_column)
    VALUES ('Example Name', 'Example Value')
        OUTPUT inserted.id
)
INSERT INTO [dbo].[student] (Master_id, student_name, student_column)
SELECT id, 'Student Name', 'Student Value'
FROM new_master;
```

    2. SQL code that shows courses', subject names and the number of students taking the course only if the course has three or more students on the course.

```
SELECT sub.*,
COUNT(stu.subject_id) AS nb_stu_enrolled

FROM student stu
JOIN subject sub
ON stu.subject_id = sub.subject_id
GROUP BY stu.subject_id
ORDER BY stu.subject_id ASC;
```



subjec...	subject_name	max_sc...	lecturer	nb_stu_enr...
11	Math	130	Charlie Sole	4
12	Computer Science	50	James Pillet	3
13	Biology	300	Carol Denby	2
14	Geography	220	Yollanda Balang	2
15	Physics	110	Chris Brother	1
16	Chemistry	400	Manny Donne	1

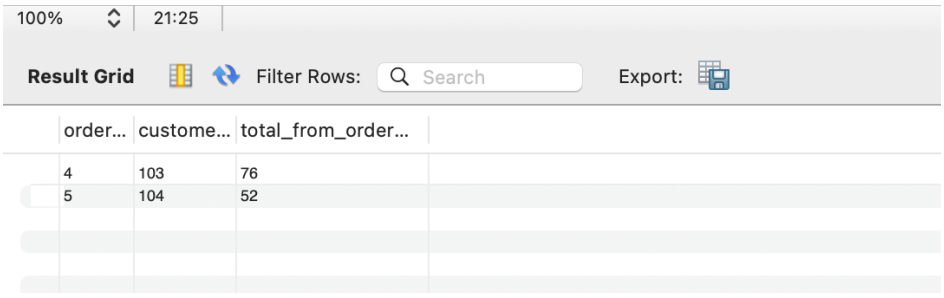
9. 2 parts :

HYPOTHESIS :

- Total = `Orders.total`
- Total *from the orders table* = Total amount spent *(by each customer from the orders table)* = `sum((OrderItems.price * OrderItems.quantity))`

1. Retrieve the order_id , customer_id, and total from the orders table where the **total** is greater than 400

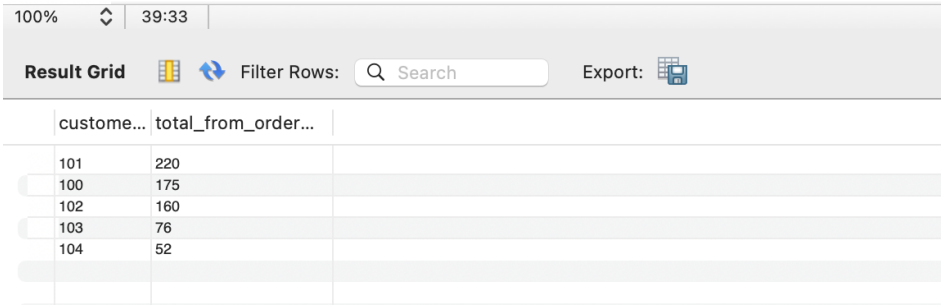
```
SELECT o.order_id,
o.customer_id,
SUM((i.price * i.quantity)) AS total_from_orders_table

FROM orders o
JOIN order_items i
ON o.order_id = i.order_id
WHERE o.total > 400
GROUP BY o.order_id;
```



order...	custome...	total_from_order...
4	103	76
5	104	52

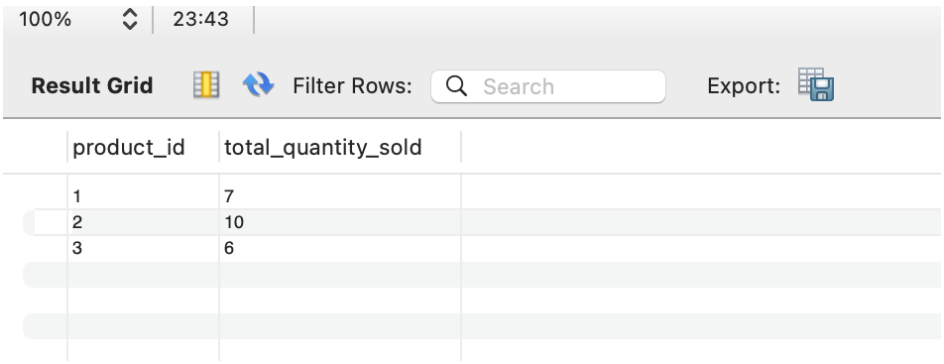
2. Retrieve the customer_id and the **total amount spent** by each customer from the orders table, ordered by the **total amount spent** in descending order

```sql
SELECT o.customer_id,
SUM((i.price * i.quantity)) AS total_from_orders_table

FROM orders o
    JOIN order_items i
        ON o.order_id = i.order_id
GROUP BY o.order_id
ORDER BY o.total DESC;
```



10. Write a query that shows the total quantity sold for each product.

```sql
SELECT o.product_id,
SUM(o.quantity) AS total_quantity_sold

FROM order_items_v2 o
GROUP BY o.product_id;
```



11. Database creation + insertion

```sql
CREATE TABLE Customers
(
    id int,
```

```
    name varchar(100),
    address varchar(100),
    city varchar(100),
    country varchar(100),
    CONSTRAINT customer_pk PRIMARY KEY (id)
);

CREATE TABLE Orders
(
    id int,
    customer_id int,
    order_date datetime,
    total int,
    CONSTRAINT orders_pk PRIMARY KEY (id),
    CONSTRAINT orders_cus_fk FOREIGN KEY (customer_id)
        REFERENCES Customers ON DELETE CASCADE
);

CREATE TABLE OrderDetails
(
    id int,
    order_id int,
    product varchar(100),
    quantity int,
    price float,
    CONSTRAINT order_details_pk PRIMARY KEY (id),
    CONSTRAINT orders_fk FOREIGN KEY (order_id)
        REFERENCES Orders ON DELETE CASCADE
);

-- Insert unique customers into Customers table
INSERT INTO Customers (id, name, address, city, country)
SELECT DISTINCT customer_id, customer_name, customer_addr, customer_city, customer_
FROM customer_orders;

-- Insert orders into Orders table
```

```
INSERT INTO Orders (id, customer_id, order_date, total)
SELECT DISTINCT order_id, customer_id, order_date, order_total
FROM sdacustomer_orders
WHERE order_id IS NOT NULL;

-- Insert order details into OrderDetails table
INSERT INTO OrderDetails (id, order_id, product, quantity, price)
SELECT DISTINCT order_details_id, order_id, product, quantity, price
FROM customer_orders
WHERE order_id IS NOT NULL;
```

3 References

1. ThoughtSpot. (n.d.). Optimizing SQL queries: A guide to data modeling best practices. ThoughtSpot. Retrieved November 1, 2024, from https://www.thoughtspot.com/data-trends/data-modeling/optimizing-sql-queries
2. SQL Shack. (2021, March 30). Different ways to SQL delete duplicate rows from a SQL table. SQL Shack. Retrieved November 1, 2024, from https://www.sqlshack.com/different-ways-to-sql-delete-duplicate-rows-from-a-sql-table/
3. Microsoft. (n.d.). Remove duplicate rows from a SQL Server table. Microsoft Learn. Retrieved November 1, 2024, from https://learn.microsoft.com/en-us/troubleshoot/sql/database-engine/development/remove-duplicate-rows-sql-server-tab