UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

Ecem BAYINDIR ~ Juste BOTTHY ~ Traoré ISSAKA MOUSSA ~ Kenny NUNGU

PROJET SQL - Sorbonne Data Analytics

M.GRAND

Sommaire

1ère partie : Base de données Sakila

1) Trouvez tous les acteurs dont le nom de famille contient les lettres “gen”

```
SELECT * FROM actor WHERE last_name LIKE '%gen%';
```

actor_id	first_name	last_n...	last_update
14	VIVIEN	BERG...	2006-02-15 04:34:33
41	JODIE	DEGE...	2006-02-15 04:34:33
107	GINA	DEGE...	2006-02-15 04:34:33
166	NICK	DEGE...	2006-02-15 04:34:33
NULL	NULL	NULL	NULL

2) Trouvez tous les acteurs dont le nom de famille contient les lettres “li”

```
SELECT * FROM actor WHERE last_name LIKE '%li%';
```

actor_id	first_name	last_n...	last_update
15	CUBA	OLIVIER	2006-02-15 04:34:33
34	AUDREY	OLIVIER	2006-02-15 04:34:33
72	SEAN	WILLI...	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	CHAP...	2006-02-15 04:34:33
96	GENE	WILLIS	2006-02-15 04:34:33
137	MORGAN	WILLI...	2006-02-15 04:34:33
164	HUMPHREY	WILLIS	2006-02-15 04:34:33
172	GROUCHO	WILLI...	2006-02-15 04:34:33
NULL	NULL	NULL	NULL

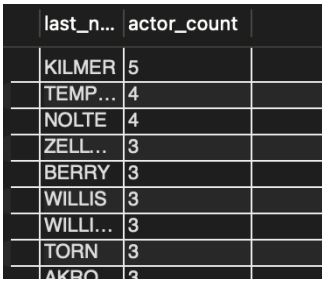
3) Listez les noms de famille de tous les acteurs, ainsi que le nombre d’acteurs portant chaque nom de famille

```
SELECT last_name, COUNT(*) AS actor_count
FROM actor
GROUP BY last_name
ORDER BY actor_count DESC;
```

last_n...	actor_count
KILMER	5
NOLTE	4
TEMP...	4
AKRO...	3
ALLEN	3
BERRY	3

4) **Listez les 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 deux acteurs**

```
SELECT last_name, COUNT(*) AS actor_count
FROM actor
GROUP BY last_name
HAVING COUNT(*) >= 2
ORDER BY actor_count DESC;
```



last_n...	actor_count
KILMER	5
TEMP...	4
NOLTE	4
ZELL...	3
BERRY	3
WILLIS	3
WILLI...	3
TORN	3
AKRO	3

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

```
SELECT staff.staff_id, staff.last_name, staff.first_name,
SUM(payment.amount) AS total_amount
FROM staff
JOIN payment ON staff.staff_id = payment.staff_id
WHERE payment.payment_date between '2005-08-01' AND '2005-08-31'
GROUP BY staff.staff_id;
```

staff_id	last_name	first_name	total_amou...
1	Hillyer	Mike	11853.65
2	Stephens	Jon	12216.49

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

```
SELECT f.title
FROM film f
WHERE f.title LIKE 'K%' OR f.title LIKE 'Q%'
AND f.language_id = (SELECT l.language_id FROM language l WHERE
l.name = 'English');
```

title
KANE EXORCIST
KARATE MOON
KENTUCKIAN GIANT
KICK SAVANNAH
KILL BROTHERHOOD
KILLER INNOCENT
KING EVOLUTION
KISS GLORY

KING EVOLUTION
KISS GLORY
KISSING DOLLS
KNOCK WARLOCK
KRAMER CHOCOLATE
KWAI HOMEWARD
QUEEN LUKE
QUEST MUSSOLINI
QUILLS BULL

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

```
SELECT c.first_name, c.last_name, c.email
FROM customer c
JOIN address a ON c.address_id = a.address_id
JOIN city ci ON a.city_id = ci.city_id
JOIN country co ON ci.country_id = co.country_id
WHERE co.country = 'Canada';
```

first_name	last_name	email
DERRICK	BOURQUE	DERRICK.BOURQUE@sakilacustomer.org
DARRELL	POWER	DARRELL.POWER@sakilacustomer.org
LORETTA	CARPENTER	LORETTA.CARPENTER@sakilacustomer.org
CURTIS	IRBY	CURTIS.IRBY@sakilacustomer.org
TROY	QUIGLEY	TROY.QUIGLEY@sakilacustomer.org

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

```
SELECT CONCAT(a.address, ',', a.district) AS store_address,
DATE_FORMAT(p.payment_date, '%Y-%m') AS year_months,
SUM(p.amount) AS total_sales
FROM payment p
JOIN staff st ON p.staff_id = st.staff_id
JOIN store s ON st.store_id = s.store_id
JOIN address a ON s.address_id = a.address_id
WHERE YEAR(p.payment_date) = 2005
GROUP BY s.store_id, year_months
ORDER BY year_months, s.store_id;
```

store_address	year_months	total_sales
47 MySakila Drive,Alberta	2005-05	2621.83
28 MySQL Boulevard,QLD	2005-05	2201.61
47 MySakila Drive,Alberta	2005-06	4774.37
28 MySQL Boulevard,QLD	2005-06	4855.52
47 MySakila Drive,Alberta	2005-07	13998.56
28 MySQL Boulevard,QLD	2005-07	14370.35
47 MySakila Drive,Alberta	2005-08	11853.65
28 MySQL Boulevard,QLD	2005-08	12216.49

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 c.customer_id, f.title, c.first_name, c.last_name, a.phone,
a.address
FROM customer c
JOIN address a ON a.address_id = c.address_id
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
WHERE r.return_date > DATE_ADD(r.rental_date, INTERVAL
f.rental_duration DAY);
```

customer_id	title	first_name	last_name	phone	address
518	ACADEMY DINOSAUR	GABRIEL	HARDER	158326114853	680 A Coruña (La Coruña) Manor
279	ACADEMY DINOSAUR	DIANNE	SHELTON	117592274996	600 Bradford Street
411	ACADEMY DINOSAUR	NORMAN	CURRIER	598912394463	1445 Carmen Parkway
581	ACADEMY DINOSAUR	VIRGIL	WOFFORD	294449058179	760 Miyakonojo Drive
359	ACADEMY DINOSAUR	WILLIE	MARKHAM	296394569728	1623 Kingstown Drive
597	ACADEMY DINOSAUR	FREDDIE	DUGGAN	644021380889	1103 Quilmes Boulevard
487	ACADEMY DINOSAUR	HECTOR	POINDEXTER	589377568313	185 Mannheim Lane
271	ACE GOLDFINGER	PENNY	NEAL	271149517630	1675 Xi´angfan Manor
82	ACE GOLDFINGER	KATHRYN	COLEMAN	821972242086	927 Bahía Blanca Parkway
139	ACE GOLDFINGER	AMBER	DIXON	33173584456	1029 Dzerzinsk Manor
180	ACE GOLDFINGER	STACY	CUNNINGHAM	104150372603	1410 Benin City Parkway
533	ACE GOLDFINGER	JESSIE	MILAM	383353187467	1332 Gaziantep Lane

---

2ème partie : Test Technique (type entreprise)

1) **How can SQL queries be optimized?**

- **Avoid SELECT *:** Select only the columns you need instead of using SELECT * to reduce data transfer and improve performance.
- **Indexing:** Create indexes on columns that are frequently used in WHERE, JOIN, and ORDER BY clauses to speed up data retrieval.
- **Use JOIN Wisely:** Use appropriate joins and ensure that you're joining on indexed columns.
- **Limit Result Set:** Use LIMIT or TOP to restrict the number of returned rows if only a subset is needed.

- **Avoid Subqueries:** Where possible, replace subqueries with joins, as joins are often more efficient.
- **Use aliases:** Using aliases is often recommended for readability, especially when multiple tables are involved.

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

- Use the DISTINCT keyword in the SELECT statement to return only unique rows
- Combine DISTINCT with aggregate functions like COUNT(), MIN(), MAX() to get distinct counts, min/max values
- Create a temporary table with unique constraint and insert data from the original table
- Use the ROW_NUMBER() or RANK() window function to identify and remove duplicate rows

Example :

```
WITH DuplicateRows AS (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY column1, column2 ORDER
BY id) AS row_num
    FROM my_table
)
DELETE FROM my_table
WHERE id IN (
    SELECT id
    FROM DuplicateRows
    WHERE row_num > 1
);
```

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

WHERE: Used to filter records before any groupings are made. It cannot be used with aggregate functions (like SUM, COUNT, etc…).

HAVING: Used to filter records after any group by is made, and it can be used with aggregate functions.

4) What is the difference between normalization and denormalization?

Normalization is a strategy used to design databases in a way that reduces unnecessary duplication of data and minimizes dependencies between data elements. This is done by organizing data into multiple related tables.

Characteristics :

- **Lowers Redundancy**
- **Enhances Data Integrity**
- **Structured Relationships**: Establishes connections between tables using primary and foreign keys to ensure data is organized logically

Denormalization is the process of combining tables to simplify the structure of a database. While this can lead to some repetition of data, it may also improve performance by making queries faster with lesser joins.

Characteristics :

- **Adds Redundancy**
- **Enhances Query Speed**: Simplifies queries by reducing the need for multiple joins
- **Optimized for read heavy systems** : Better suited for applications where data is primarily accessed rather than modified

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

DELETE removes rows with optional WHERE filtering and can be rolled back.
TRUNCATE removes all rows instantly without a WHERE clause and cannot be rolled back in all databases.
DELETE is slower for large tables, TRUNCATE is faster for removing all rows from a table.

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

- Use unique constraints on columns
- Use DISTINCT in queries where duplicates are unwanted
- Implement primary keys or unique indexes on key fields

- Use the ROW_NUMBER() or RANK() window functions to identify and handle duplicate rows

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

5 types of relationships:

- **one-to-one:** A single record in one table is associated with a single record in another table
- **one-to-many:** A single record in one table is associated with multiple records in another table
- **many-to-one:** Multiple records in one table are associated with a single record in another table.
- **many-to-many:** Multiple records in one table are associated with multiple records in another table, requiring a junction/bridge table
- **self referencing relationships:** A record in a table is related to another record in the same table, often used to represent hierarchical data

8) 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.

```
INSERT INTO master (id, master_name) VALUES (...);
INSERT INTO student (student_id, student_name, city, subject_id,
master_id) VALUES (..., (SELECT id FROM master WHERE master_name =
'...'));
```

Then give an example of the 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 s.subject_name, COUNT(st.student_id) AS num_students
FROM subject s JOIN student st ON s.subject_id = st.subject_id
GROUP BY s.subject_name
HAVING COUNT(st.student_id) >= 3;
```

Table: subject			
subject_id	subject_name	max_score	lecturer
11	Math	130	Charlie Sole
12	Computer Science	50	James Pillet

13	Biology	300	Carol Denby
14	Geography	220	Yollanda Balang
15	Physics	110	Chris Brother
16	Chemistry	400	Manny Donne

Table: student			
student_id	student_name	city	subject_id
2001	Olga Thorn	New York	11
2002	Sharda Clement	San Francisco	12
2003	Bruce Shelkins	New York	13
2004	Fabian Johnson	Boston	15
2005	Bradley Camer	Stanford	11
2006	Sofia Mueller	Boston	16
2007	Rory Pietman	New Haven	12
2008	Carly Walsh	Tulsa	14
2011	Richard Curtis	Boston	11
2012	Cassey Ledgers	Stanford	11
2013	Harold Ledgers	Miami	13
2014	Davey Bergman	San Francisco	12
2015	Darcey Button	Chicago	14

9) Write a query to retrieve the order_id , customer_id, and total from the orders table where the total is greater than 400.

```
SELECT order_id, customer_id, total
FROM orders
WHERE total > 400;
```

Then do a query to 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.

```
SELECT customer_id, SUM(total) AS total_spent
FROM orders
GROUP BY customer_id
ORDER BY total_spent DESC;
```

Table: Orders

order_id	customer_id	order_date	total
1	100	01/01/2021	200
2	101	02/02/2021	300
3	102	03/03/2021	400
4	103	04/04/2021	500
5	104	05/05/2021	600

Table: Order items

order_id	product_id	quantity	price
1	10	2	50
1	11	3	25
2	12	4	30
2	13	5	20
3	14	6	15
3	15	7	10
4	16	8	5
4	17	9	4
5	18	10	3
5	19	11	2

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

```
SELECT product_id, SUM(quantity) AS total_quantity_sold
FROM Order_items
GROUP BY product_id;
```

Table: Order items

order_id	order_date	customer_id	product_id	quantity
1	01/01/2022	101	1	2
2	01/01/2022	102	1	1
3	01/01/2022	103	2	5
4	02/01/2022	104	3	3
5	02/01/2022	105	1	2
6	02/01/2022	101	3	1
7	03/01/2022	102	2	4
8	03/01/2022	103	1	2
9	03/01/2022	104	2	1
10	04/01/2022	105	3	2

11) Assume we have a large excel spreadsheet with customer orders data. Each row contains information about a single order, including the customer name, order date, order ID, order quantity, and order total. We want to divide this data into three tables: Customers, Orders, and OrderDetails. Customers will store customer information, Orders will store order information (including customer ID), and OrderDetails will store details about individual order items (including order ID).

We want to insert the customer orders data into the three tables Customers, Orders, and OrderDetails. Write an SQL query that inserts the data into the appropriate tables, and ensures that the customer ID and order ID are maintained across all three tables. The Orders table should have a foreign key reference to the Customers table, and the OrderDetails table should have a foreign key reference to the Orders table. Assume that the source data is stored in a single table named 'customer_orders', and that the schema for each destination table is already defined.

```
# 1. AUTO_INCREMENT for Primary Keys

CREATE TABLE Customers (
id INT PRIMARY KEY,
name VARCHAR(50),
address VARCHAR(100),
city VARCHAR(50),
country VARCHAR(50)
);

CREATE TABLE Orders (
id INT PRIMARY KEY,
customer_id INT,
order_date DATE,
total DECIMAL(10,2),
FOREIGN KEY (customer_id) REFERENCES Customers(id)
);

CREATE TABLE OrderDetails (
id INT PRIMARY KEY,
order_id INT,
product_id INT,
quantity INT,
price DECIMAL(10,2),
FOREIGN KEY (order_id) REFERENCES Orders(id)
);

# 2. Data Integrity Constraints

CREATE TABLE Customers (
id INT AUTO_INCREMENT PRIMARY KEY,
```

```
name VARCHAR(50) NOT NULL,
address VARCHAR(100),
city VARCHAR(50),
country VARCHAR(50)
);

CREATE TABLE Orders (
id INT AUTO_INCREMENT PRIMARY KEY,
customer_id INT NOT NULL,
order_date DATE NOT NULL,
total DECIMAL(10,2) NOT NULL,
FOREIGN KEY (customer_id) REFERENCES Customers(id)
);

CREATE TABLE OrderDetails (
id INT AUTO_INCREMENT PRIMARY KEY,
order_id INT NOT NULL,
product_id INT NOT NULL,
quantity INT NOT NULL,
price DECIMAL(10,2) NOT NULL,
FOREIGN KEY (order_id) REFERENCES Orders(id)
);
```

3. Indexing Foreign Key Columns

```
CREATE INDEX idx_orders_customer_id ON Orders(customer_id);
CREATE INDEX idx_orderdetails_order_id ON OrderDetails(order_id);
```

Customers:

id	name	address	city	country
1	John Smith	123 Main St.	Anytown	USA
2	Jane Doe	456 Oak St.	Somewhere	USA
3	Bob Johnson	789 Pine St.	Anytown	USA
4	Alice Lee	1010 Elm St.	Nowhere	USA
5	David Kim	1234 Maple St.	Anytown	USA

Orders:

id	customer_id	order_date	total
1	1	01/01/2022	100

2	1	02/01/2022	150
3	2	03/01/2022	75
4	3	04/01/2022	200
5	4	05/01/2022	50

OrderDetails:

id	order_id	product	quantity	price
1	1	Widget A	2	25
2	1	Widget B	1	50
3	2	Widget C	1	75
4	2	Widget D	2	37.5
5	3	Widget A	1	25
6	3	Widget B	2	50
7	4	Widget D	1	200
8	5	Widget A	2	25

Sources :

- Emil Drkusic (2020) : “Learn SQL: Types of relations” : https://www.sqlshack.com/learn-sql-types-of-relations/
- Design Gurus Team (2021) : “What is SQL Normalization and Denormalization?” : https://www.designgurus.io/answers/detail/what-is-sql-normalization-and-denormalization
- Shreeya Thakur : “Difference Between DELETE and TRUNCATE Commands In SQL” : https://unstop.com/blog/difference-between-delete-and-truncate-in-sql
- Imarticus Team (2024) : “How To Delete Duplicate Rows In SQL? Detailed Guide With Syntax And Examples” : https://imarticus.org/blog/how-to-delete-duplicate-rows-in-sql/