Projet SQL

Lydia NEGHAD, Pierre PASSERINI, Haby NDIAYE et Thanh-Thao CHU

1ère partie : Base de données Sakila

1) Trouvez tous les acteurs dont le nom de famille contient les lettres "gen".

```
SELECT * FROM sakila.actor
WHERE last_name LIKE "%gen%" ;
```

Result Grid | Filter Rows: | Edit:

actor_id	first_name	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) Trouvez tous les acteurs dont le nom de famille contient les lettres "li".

```sql
SELECT * FROM sakila.actor
WHERE last_name LIKE "%li%" ;
```

Result Grid | Filter Rows: | Edit:

actor_id	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.

Limit to 1000 rows

```sql
SELECT last_name, COUNT(*) AS number_of_actors
FROM actor
GROUP BY last_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

last_name	COUNT(*)
AKROYD	3
ALLEN	3
ASTAIRE	1
BACALL	1
BAILEY	2
BALE	1
BALL	1
BARRYMORE	1
BASINGER	1
BENING	2
BERGEN	1
BERGMAN	1
BERRY	3
BIRCH	1
BLOOM	1
BOLGER	2
BRIDGES	1

4) Lister 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.

```sql
SELECT last_name, COUNT(*) AS nombre_acteurs
FROM actor
GROUP BY last_name
HAVING count(*)>= 2
ORDER BY nombre_acteurs DESC
```

Result Grid | Filter Rows: | Export: | Wrap

last_name	nombre_acteurs
KILMER	5
NOLTE	4
TEMPLE	4
AKROYD	3
ALLEN	3
HOPKINS	3
DAVIS	3
BERRY	3
HARRIS	3
GARLAND	3
DEGENERES	3
HOFFMAN	3
GUINESS	3
JOHANSS	3

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

SQL File 3* testSQL

Limit to 1000 rows

```sql
/*5. Utilisez JOIN pour afficher le montant total perçu par chaque membre du personnel en août 2005*/
SELECT staff.staff_id, SUM(amount) AS montant_total
FROM payment
INNER JOIN staff on payment.staff_id = staff.staff_id
WHERE payment_date BETWEEN "2005-08-01" AND "2005-08-31"
GROUP BY staff.staff_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

staff_id	montant_total
1	11853.65
2	12216.49

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

```sql
SELECT film.title
FROM film
LEFT JOIN language ON film.language_id = language.language_id
WHERE language.name = 'English' AND (film.title LIKE 'K%' OR film.title LIKE 'Q%')
ORDER BY film.title DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

title
QUILLS BULL
QUEST MUSSOLINI
QUEEN LUKE
KWAI HOMEWARD
KRAMER CHOCOLATE
KNOCK WARLOCK
KISSING DOLLS
KISS GLORY
KING EVOLUTION
KILLER INNOCENT
KILL BROTHERHOOD
KICK SAVANNAH
KENTUCKIAN GIANT
KARATE MOON
KANE EXORCIST

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

```sql
SELECT first_name, last_name, email FROM customer
LEFT JOIN address ON customer.address_id = address.address_id
LEFT JOIN city ON address.city_id = city.city_id
LEFT JOIN country ON city.country_id = country.country_id
WHERE country.country = 'Canada'
ORDER BY customer.last_name, customer.first_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

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

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

```sql
SELECT store.store_id,
       CONCAT(YEAR(payment.payment_date), '-', LPAD(MONTH(payment.payment_date), 2, '0')) AS mois_annee,
       SUM(payment.amount) AS total_ventes
FROM payment
JOIN staff ON payment.staff_id = staff.staff_id
JOIN store ON staff.store_id = store.store_id
WHERE YEAR(payment.payment_date) = 2005
GROUP BY store.store_id, YEAR(payment.payment_date), MONTH(payment.payment_date), mois_annee
ORDER BY store.store_id, mois_annee;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

store_id	mois_annee	total_ventes
1	2005-05	2621.83
1	2005-06	4774.37
1	2005-07	13998.56
1	2005-08	11853.65
2	2005-05	2201.61
2	2005-06	4855.52
2	2005-07	14370.35
2	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)

SQL File 3*

Limit to 1000 rows

```sql
SELECT f.title AS film_title,
       c.first_name AS customer_first_name,
       c.last_name AS customer_last_name,
       a.phone AS customer_phone,
       a.address AS customer_address
FROM rental AS r
JOIN inventory AS i ON r.inventory_id = i.inventory_id
JOIN film AS f ON i.film_id = f.film_id
JOIN customer AS c ON r.customer_id = c.customer_id
JOIN address AS a ON c.address_id = a.address_id
WHERE r.return_date IS NULL;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

film_title	customer_first_name	customer_last_name	customer_phone	customer_address
DETECTIVE VISION	LEE	HAWKS	270456873752	1661 Abha Drive
IGBY MAKER	GLORIA	COOK	347487831378	1668 Saint Louis Place
CURTAIN VIDEOTAPE	LUCY	WHEELER	49677664184	624 Oshawa Boulevard
MONSTER SPARTACUS	KATIE	ELLIOTT	940830176580	447 Surakarta Loop
PRIDE ALAMO	DAISY	BATES	816436065431	661 Chisinau Lane
MOTIONS DETAILS	CHRISTINE	ROBERTS	539758313890	1447 Imus Way
ENCINO ELF	CLAUDIA	FULLER	630424482919	346 Skikda Parkway
OPPOSITE NECKLACE	SYLVIA	ORTIZ	765345144779	241 Mosul Lane
STREAK RIDGEMONT	DARYL	LARUE	954786054144	1208 Tama Loop
DEEP CRUSADE	MARGARET	MOORE	380657522649	613 Korolev Drive
DANCES NONE	JESSIE	BANKS	352679173732	1229 Valencia Parkway
CONVERSATION DOWNHILL	RAFAEL	ABNEY	82671830126	48 Maracaibo Place
SHOCK CABIN	STEPHANIE	MITCHELL	42384721397	42 Brindisi Place
WEDDING APOLLO	REGINA	BERRY	201705577290	475 Atšinsk Way
WINDOW SIDE	JEREMY	HURTADO	600264533987	1133 Rizhao Avenue

Result 1

2ème partie : Test Technique (Type entreprise)

1) How can SQL queries be optimized?

- **Use indexes** : An index acts like a "bookmark" in a table, allowing for quick data retrieval. Creating an index on frequently used columns in searches (like in WHERE or JOIN clauses) speeds up the query.
- **Avoid SELECT * :** Instead of selecting all columns with SELECT *, only ask for the columns you need (e.g., SELECT student_name, city FROM student). This reduces the amount of data returned and speeds up execution.
- **Use LIMIT** to reduce the number of rows : If you only need part of the results, use LIMIT to limit the number of rows (e.g., LIMIT 10).
- **Filter data as early as possible** : Place filtering conditions in the WHERE clause to reduce the number of rows processed by the query
- **Use UNION ALL instead of UNION** : The main difference between UNION and UNION ALL is that : UNION is only keeps unique records and UNION ALL is keeps all records, including duplicates.
- **Normalization database tables** : This is the process of organizing data in a database. It includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.
- CTE means Common Table Expression. It's a **SELECT** query that returns a temporary result set that you can use within another SQL query.They are often used to break up complex queries to make them simpler.

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

To remove duplicate rows, you can use the DELETE command with a subquery to identify duplicates. However, this can be complex for a beginner. Here's a simpler solution:

- Identify duplicates using **SELECT DISTINCT** to see unique data.
- The **UNION** statement removes duplicate rows from the query results.
- In cases where duplicates happen in a few columns and it is necessary to have unique items, we may use **CONCAT** to create temporary row IDs, then use **SELECT DISTINCT** on this column to remove duplicates
- Using **ROW_NUMBER()** with CTEs (if supported) **:** This method assigns a unique row number to each row and deletes duplicates based on this.

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

Both WHERE and HAVING clauses filter data, but they are used differently:

WHERE : Used to filter rows before aggregation (such as SUM or COUNT) is applied. It's for simple conditions on columns (e.g., WHERE age > 18).

HAVING : Used after aggregation to filter grouped results. For example, if you want to find subjects (courses) with at least three students, you would use HAVING after a GROUP BY. Unlike the WHERE clause, **HAVING** can be used with aggregate functions. Common aggregate functions include COUNT(), SUM(), MIN(), and MAX().

4) What is the difference between normalization and denormalization ?

- **Normalization** : This is the process of structuring data to reduce redundancy and avoid anomalies. By dividing data into multiple related tables, you ensure each piece of data is stored only once. This makes the database more efficient and easier to maintain.
- **Denormalization** : This is the opposite of normalization. It involves combining tables to reduce the number of JOINs needed. Denormalization is sometimes used to improve read performance, though it may cause data duplication.

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

- **DELETE** : Removes specific rows in a table. You can specify which rows to delete with a WHERE clause. DELETE can be rolled back if you're in a transactional mode.
- **TRUNCATE** : Deletes all rows from a table without any filtering option. It's faster than DELETE as it doesn't check each row individually. TRUNCATE is often irreversible and

completely empties the table.

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

To prevent duplicates when inserting data, here are some simple methods :

- **Use PRIMARY KEY** : To prevent the entry of duplicate records into a table, we can define a PRIMARY KEY or a UNIQUE Index on the relevant fields. These database constraints ensure that each entry in the specified column or set of columns is unique.
- **Use UNIQUE constraints** : Define a column or set of columns as UNIQUE to ensure that values in this column or combination of columns are unique.

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

In relational databases, there are three main types of relationships between tables :

- **One-to-One Relationship** : Each record in one table is linked to a single record in another table. For example, a student might have a single locker, and each locker belongs to only one student.

- **One-to-Many Relationship** : One record in a table can be linked to multiple records in another table. For example, one professor (a record in the professor table) can teach multiple courses (records in the course table).

- **Many-to-Many Relationship** : Multiple records in one table can be linked to multiple records in another table. For example, a student can enroll in multiple courses, and each course can have multiple students. This usually requires a junction table to manage the links between the two main tables.

- **Many-to-One Relationship** : A many-to-one relationship is essentially the reverse of a one-to-many relationship. It occurs when multiple records in one table are associated with a single record in another table.

- **Self-Referencing Relationship** : A self-referencing relationship is when a record in a table is related to another record within the same table. This is 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.

A. <u>Table Creation :</u>

- I created two tables : subject and student.

- Since the instruction requires ensuring that the student table includes the correct Master_id, which references another table (Master), I also created an additional table called Master.

```
CREATE TABLE Master (
    id INT PRIMARY KEY,
    master_name VARCHAR(100)
);

CREATE TABLE subject (
    subject_id INT PRIMARY KEY,
    subject_name VARCHAR(100),
    max_score INT,
    lecturer VARCHAR(100)
);

CREATE TABLE student (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(100),
    city VARCHAR(100),
    subject_id INT,
    Master_id INT,
    FOREIGN KEY (subject_id) REFERENCES subject(subject_id),
    FOREIGN KEY (Master_id) REFERENCES Master(id)
);
```

B. <u>Inserting Data into the Master Table :</u>

The Master table is created to satisfy the requirement that each student has a valid **Master_id**. I inserted a few example IDs into Master to make sure there are valid references available for the **Master_id** column in student :

```
INSERT INTO Master (id)
VALUES
(1),
(2),
(3);

```

C. <u>Inserting Data into the Subject Table :</u>

The **subject** table stores information about each course, including the course name, maximum score, and lecturer :

```
INSERT INTO subject (subject_id, subject_name, max_score, lecturer)
VALUES
(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');
```

D. Inserting Data into the Student Table, including Master_id :

Each student is linked to a **subject_id** for the course they are taking and to a **Master_id**, which references the Master table :

```
INSERT INTO student (student_id, student_name, city, subject_id, Master_id)
VALUES
(2001, 'Olga Thorn', 'New York', 11, 1),
(2002, 'Sharda Clement', 'San Francisco', 12, 2),
(2003, 'Bruce Shelkins', 'New York', 13, 1),
(2004, 'Fabian Johnson', 'Boston', 15, 3),
(2005, 'Bradley Camer', 'Stanford', 11, 2),
(2006, 'Sofia Mueller', 'Boston', 16, 1),
(2007, 'Rory Pietman', 'New Haven', 12, 3),
(2008, 'Carly Walsh', 'Tulsa', 14, 1),
(2011, 'Richard Curtis', 'Boston', 11, 2),
(2012, 'Cassey Ledgers', 'Stanford', 11, 3),
(2013, 'Harold Ledgers', 'Miami', 13, 1),
(2014, 'Davey Bergman', 'San Francisco', 12, 2),
(2015, 'Darcey Button', 'Chicago', 14, 3);
```

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.

```sql
SELECT sub.subject_name AS course_name,
        COUNT(st.student_id) AS student_count
FROM subject AS sub
JOIN student AS st ON sub.subject_id = st.subject_id
GROUP BY sub.subject_name, sub.subject_id
HAVING student_count >= 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

course_name	student_count
Math	4
Computer Science	3

Explanation :

- **SELECT sub.subject_name AS course_name** : Selects the name of the course from the subject table and renames it as course_name.
- **COUNT(st.student_id) AS student_count** : Counts the number of students (student_id) enrolled in each course and renames it as student_count.
- **FROM subject AS sub** : Uses the subject table and assigns it an alias sub.
- **JOIN student AS st ON sub.subject_id = st.subject_id** : Joins the subject and student tables based on subject_id to link each course to its enrolled students.
- **GROUP BY sub.subject_id, sub.subject_name** : Groups the results by course ID and course name to count the students per course.
- **HAVING COUNT(st.student_id) >= 3** : Filters the results to show only courses with three or more students enrolled.

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

```sql
SELECT order_id, customer_id, total
FROM Orders
WHERE total > 400;
```

order_id	customer_id	total
4	103	500
5	104	600

Explanation:

- **SELECT order_id, customer_id, total** : Selects the specified columns from the Orders table.
- **FROM Orders** : Retrieves data from the Orders table.
- **WHERE total > 400** : Filters the results to include only orders where the total amount is greater than 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(price * quantity) AS Total_price
FROM Orders
JOIN `Order items` ON `Order items`.order_id = orders.order_id
GROUP BY customer_id
ORDER BY Total_price DESC
```

customer_id	Total_price
101	220
100	175
102	160
103	76
104	52

Explanation:

- **SELECT customer_id, SUM(price * quantity) AS Total_price** : Selects the customer_id and calculates the total spent using SUM(price * quantity), renaming it as Total_price.
- **FROM Orders** : Retrieves data from the Orders table.
- **GROUP BY customer_id** : Groups the results by customer_id to calculate the total amount for each customer.
- **ORDER BY Total_price DESC** : Sorts the results by the total amount spent in descending order (from highest to lowest).

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;
```

Explanation :

- **SELECT product_id** : Selects the product_id to identify each product.
- **SUM(quantity) AS total_quantity_sold** : Calculates the sum of the quantity sold for each product and renames it as total_quantity_sold.
- **FROM Order_items** : Uses data from the Order_items table.
- **GROUP BY product_id** : Groups the results by product_id to get one row per product with the total quantity sold.

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 While the instructions did not ask us to insert data, for the purpose of demonstration, we performed a data insertion into the previously created Customers, Order and OrderDetails tables).

A. <u>Creating the Tables :</u>

Limit to 1000 rows

```sql
CREATE TABLE Customers (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    address VARCHAR(255),
    city VARCHAR(100),
    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 VARCHAR(100),
    quantity INT,
    price DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES Orders(id)
);
```

B. <u>Inserting Data :</u>

While the instructions did not ask us to insert data, for the purpose of demonstration, we performed a data insertion into the previously created Customers, Order and OrderDetails tables

- <u>For **Customers** :</u>

```sql
INSERT INTO Customers (id, name, address, city, country)
VALUES
(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');
```

- <u>For **Orders** :</u>

```sql
INSERT INTO Orders (id, customer_id, order_date, total)
VALUES
(1, 1, '2022-01-01', 100),
(2, 1, '2022-01-02', 150),
(3, 2, '2022-01-03', 75),
(4, 3, '2022-01-04', 200),
(5, 4, '2022-01-05', 50);
```

- <u>For **OrderDetails** :</u>

```sql
INSERT INTO OrderDetails (id, order_id, product, quantity, price)
VALUES
(1, 1, 'Widget A', 2, 25),
(2, 1, 'Widget B', 1, 50),
(3, 1, 'Widget C', 1, 75),
(4, 2, 'Widget D', 1, 37.5),
(5, 3, 'Widget A', 2, 25),
(6, 3, 'Widget B', 1, 50),
(7, 3, 'Widget C', 1, 75),
(8, 4, 'Widget D', 1, 200),
(9, 5, 'Widget A', 2, 25);
```

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.

A. <u>Insert Customers into the **Customers** Table</u> :

```
INSERT INTO Customers (id, name, address, city, country)
SELECT DISTINCT customer_id, customer_name, customer_address, customer_city, customer_country
FROM customer_orders;
```

This query inserts unique information about each customer from **customer_orders** into the **Customers** table.

B. <u>Insert Orders into the **Orders** Table</u> :

```
INSERT INTO Orders (id, customer_id, order_date, total)
SELECT DISTINCT order_id, customer_id, order_date, total
FROM customer_orders;
```

This query inserts information about each order into the **Orders** table, linking each order to a customer via **customer_id**.

C. <u>Insert Order Details into the **OrderDetails** Table :</u>

```
INSERT INTO OrderDetails (id, order_id, product, quantity, price)
SELECT id, order_id, product, quantity, price
FROM customer_orders;
```

This query inserts each order line item into **OrderDetails**, using **order_**id to link each item to its respective order in the **Orders** table.

These three queries fulfill the requirement by distributing data from **customer_orders** into the **Customers, Orders**, and **OrderDetails** tables, while maintaining the necessary **customer_id** and **order_id** relationships.