Projet SQL

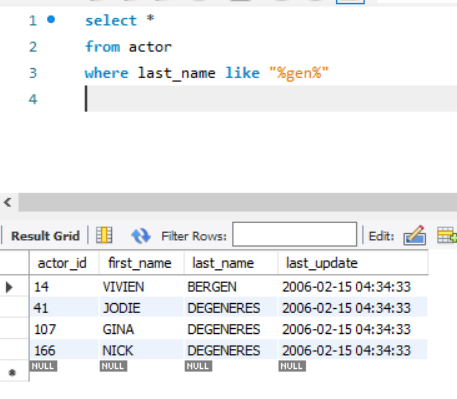
# 1ère partie : Base de données Sakila

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

**select \***

**from actor**

**where last\_name like "%gen%"**

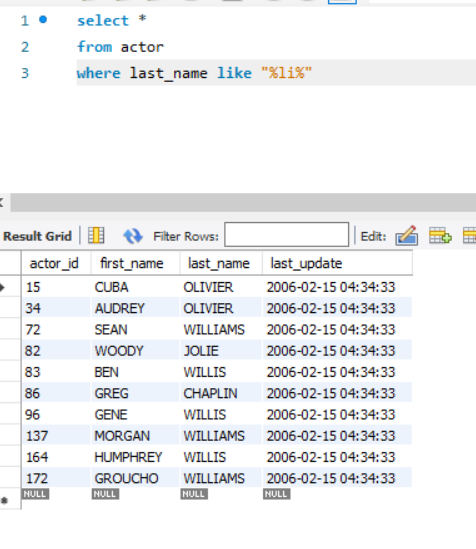


1. Trouvez tous les acteurs dont le nom de famille contient les lettres "li".

**select \***

**from actor**

**where last\_name like "%li%"**

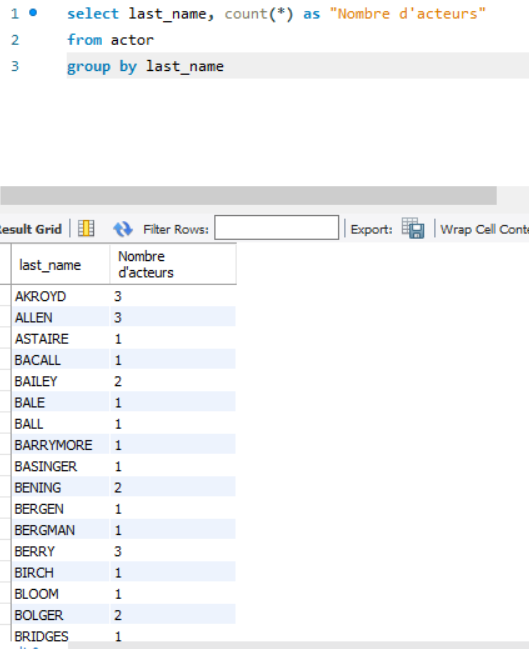


1. Liste des noms de famille de tous les acteurs, ainsi que le nombre d'acteurs portant chaque nom de famille.

**select last\_name, count(\*) as "Nombre d'acteurs"**

**from actor**

**group by last\_name**



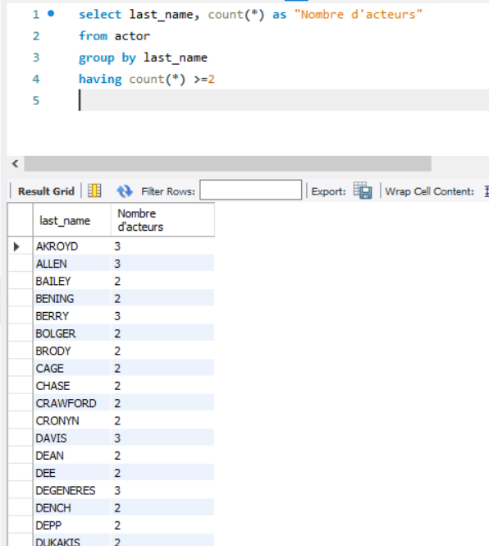
1. Lister les noms de famille des acteurs et le nombre d'acteurs qui portent chaque nom de famille, mais seulement pour les noms qui sont portés par au moins deux acteurs.

**select last\_name, count(\*) as "Nombre d'acteurs"**

**from actor**

**group by last\_name**

**having count(\*) >=2**



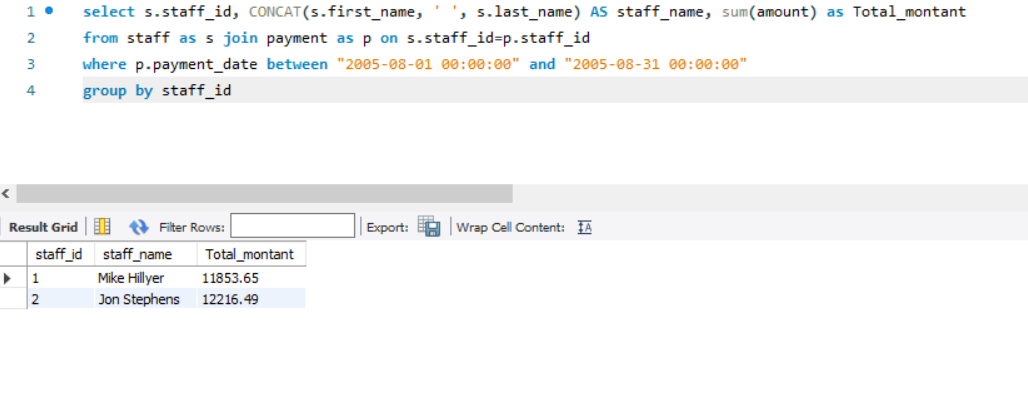
1. Utilisez JOIN pour afficher le montant total perçu par chaque membre du personnel en août 2005.

**select s.staff\_id, CONCAT(s.first\_name, ' ', s.last\_name) AS staff\_name, sum(amount) as Total\_montant**

**from staff as s join payment as p on s.staff\_id=p.staff\_id**

**where p.payment\_date between "2005-08-01 00:00:00" and "2005-08-31 00:00:00"**

**group by staff\_id**

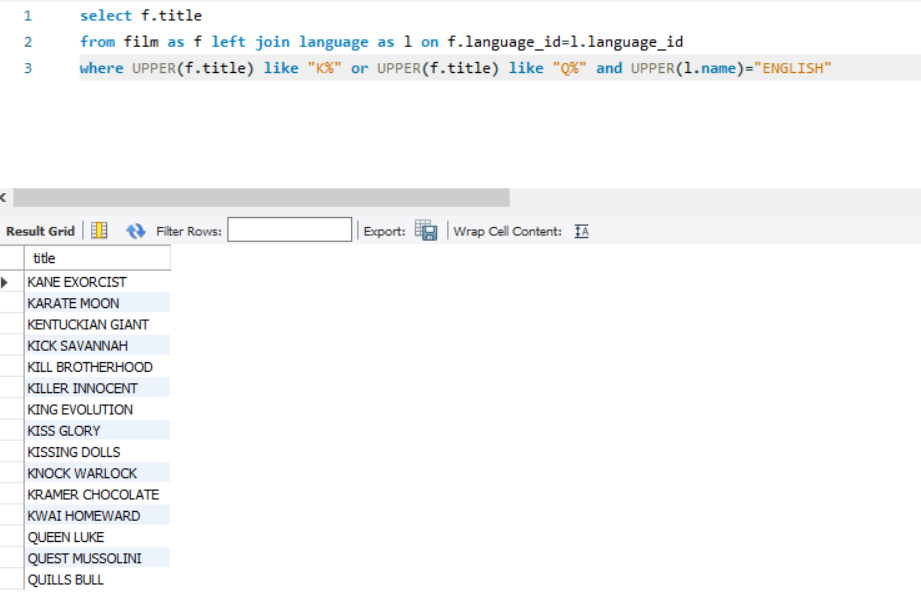


1. Afficher les titres des films commençant par les lettres K et Q dont la langue est l'anglais.

**select f.title**

**from film as f left join language as l on f.language\_id=l.language\_id**

**where UPPER(f.title) like "K%" or UPPER(f.title) like "Q%" and UPPER(l.name)="ENGLISH"**



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

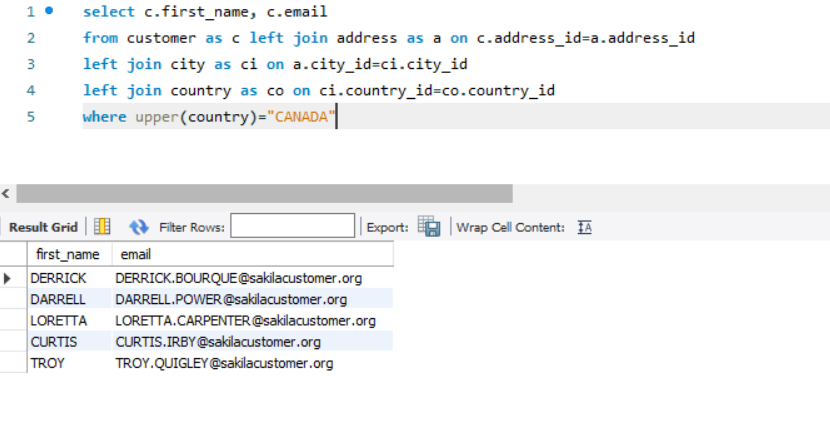
**select c.first\_name, c.email**

**from customer as c left join address as a on c.address\_id=a.address\_id**

**left join city as ci on a.city\_id=ci.city\_id**

**left join country as co on ci.country\_id=co.country\_id**

**where upper(country)="CANADA"**



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

**select sto.store\_id,concat(month(p.payment\_date),"-",year(p.payment\_date)) as month\_year, sum(p.amount) as "somme des ventes"**

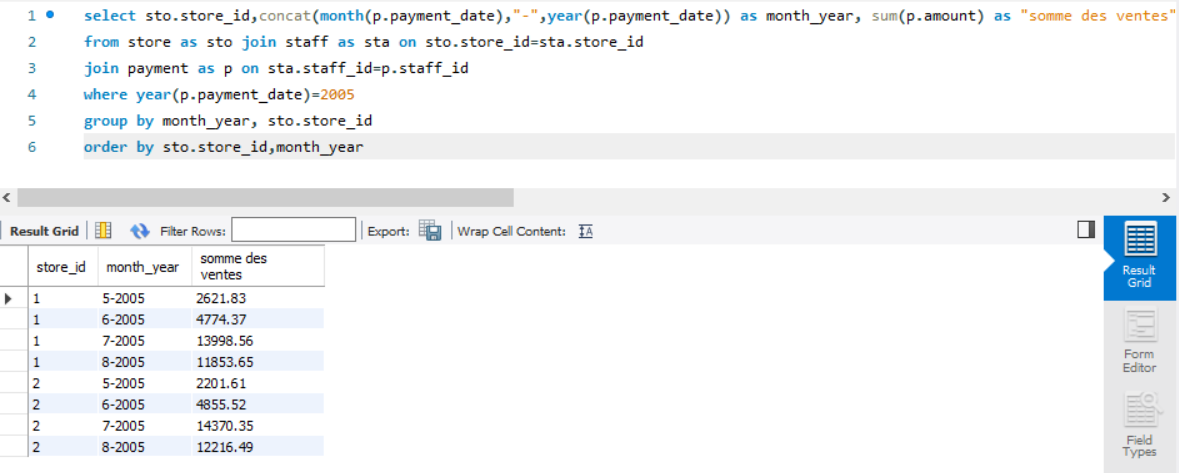
**from store as sto join staff as sta on sto.store\_id=sta.store\_id**

**join payment as p on sta.staff\_id=p.staff\_id**

**where year(p.payment\_date)=2005**

**group by month\_year, sto.store\_id**

**order by sto.store\_id,month\_year**



1. 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 f.title, concat(c.first\_name," ",c.last\_name) as nom\_client,a.phone, a.address**

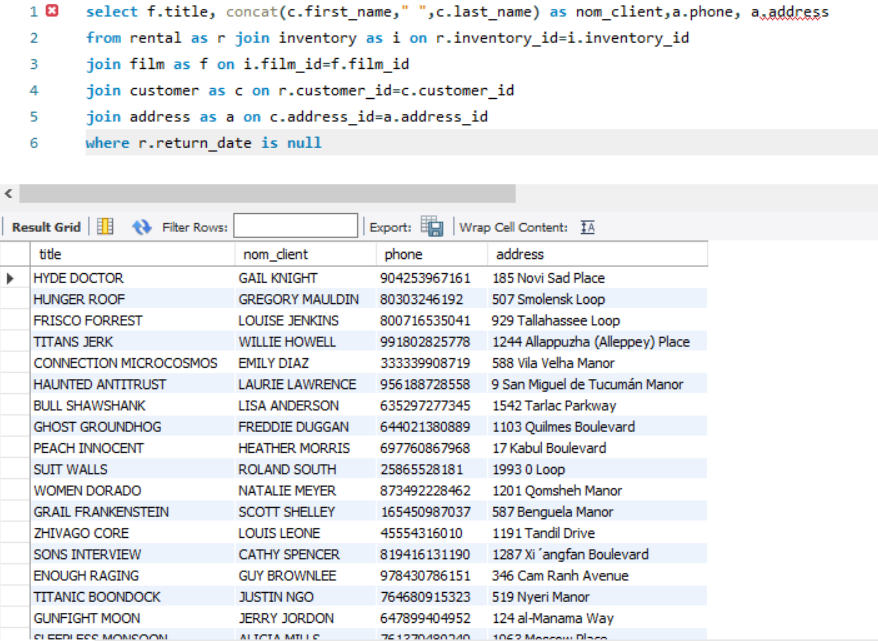
**from rental as r join inventory as i on r.inventory\_id=i.inventory\_id**

**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**



# 2ème partie : Test technique (type entreprise)

1. How can SQL queries be optimized?

Il permet d’optimiser le stockage et le rangement des données afin de les mobiliser plus rapidement à travers une vue

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

DELETE FROM table

WHERE (noms\_colonnes) IN (

SELECT noms\_colonnes

FROM table

GROUP BY noms\_colonnes

HAVING COUNT(\*) > 1

);

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

Le Having est un attribut après le groupby alors que le where est avant et permet de filtrer avant de grouper

1. What is the difference between normalization and denormalization?

La normalisation est utilisée pour évider les redondances et optimiser le stockage des données qui se fait avec des grandes tables mais en plus petites tables. Ainsi on définit des relations entre ces tables et on a un schéma normalisé, permettant un stockage optimal des données

La dénormalisation vise à limiter le nombre de relation pour optimiser au maximum la performance des requêtes. Moins il y aura de relation, plus les requêtes seront performantes.

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

La différence est dans le traitement qu’elles utilisent pour fonctionner. La fonction DELETE va supprimer les lignes une à une en tenant une journalisation des opérations alors que TRUNCATE est une commande par définition qui va effacer toutes les lignes d’un coup ? DELETE permet d’ailleurs de supprimer des lignes en fonction de caractéristiques alors que TRUNCATE non.

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

La première est d’inclure DISTINCT dans le select pour ne pas avoir de doublons dans le résultat.

La deuxième est d’associer une clé primaire à chaque table, qui garantira l’unicité des lignes

La troisième peut etre de regrouper les lignes que l’on souhaite et ainsi avoir un résultat unique avec le group by

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

**Relation 1-1 :** Dans cette relation, chaque occurrence dans une table correspondra exactement à une autre occurrence dans une autre table. Cela signifie qu'il existe une correspondance unique entre les enregistrements des deux tables. Par exemple, une table "Utilisateur" pourrait avoir une relation 1-1 avec une table "Profil", où chaque utilisateur a un seul profil associé et chaque profil est lié à un seul utilisateur.

**Relation 1-N :** Dans cette relation, une occurrence dans une table peut être associée à plusieurs occurrences dans une autre table, mais chaque occurrence dans la seconde table est associée à une seule occurrence dans la première table. Par exemple, une table "Département" pourrait avoir une relation 1-N avec une table "Employé", où un département peut avoir plusieurs employés, mais chaque employé appartient à un seul département.

**Relation N-N :** Dans cette relation, plusieurs occurrences dans une table peuvent être associées à plusieurs occurrences dans une autre table. Cela signifie qu'il n'y a pas de contrainte sur le nombre d'occurrences associées entre les deux tables. Par exemple, une table "Étudiant" pourrait avoir une relation N-N avec une table "Cours", où un étudiant peut suivre plusieurs cours et un cours peut avoir plusieurs étudiants inscrits. Pour modéliser cette relation, une table de jointure intermédiaire est généralement utilisée pour mapper les associations entre les deux tables.

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.

**Insertion des données dans la table "subject"**

**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');**

**Insertion des données dans la table "student"**

**INSERT INTO student (student\_id, student\_name, city, subject\_id)**

**VALUES**

**(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);**

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.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

1. 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 |

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

**Select product\_id, sum(quantity) as “total\_quantity”**

**From “Order items”**

**Groupby 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 |

1. 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).

**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 |

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.

**Insertion des clients**

**insert into customers (name, address, city, country)**

**select distinct**

**customer\_name,**

**customer\_address,**

**customer\_city,**

**customer\_country**

**from customer\_orders;**

**Insertion des commandes en liaison avec les clients**

**insert into orders (customer\_id, order\_date, total)**

**select**

**c.id,**

**order\_date,**

**total**

**from customer\_orders co**

**inner join customers c on co.customer\_name = c.name**

**and co.customer\_address = c.address**

**and co.customer\_city = c.city**

**and co.customer\_country = c.country;**

**Insertion des details des commandes**

**insert into orderdetails (order\_id, product, quantity, price)**

**select**

**o.id,**

**product,**

**quantity,**

**price**

**from customer\_orders co**

**inner join orders o on co.customer\_name = (select name from customers where id = o.customer\_id)**

**and co.order\_date = o.order\_date**

**and co.total = o.total;**