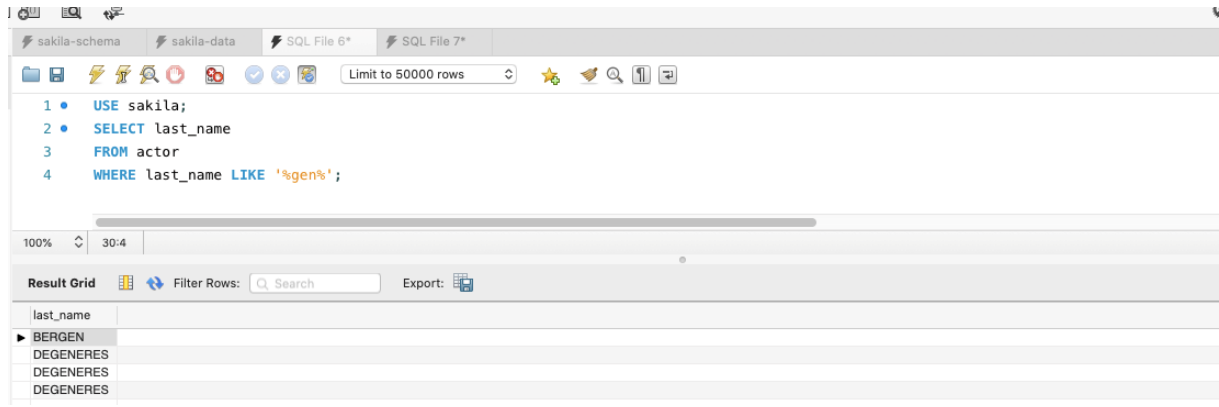


Projet SQL

1ère partie : Base de données Sakila

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



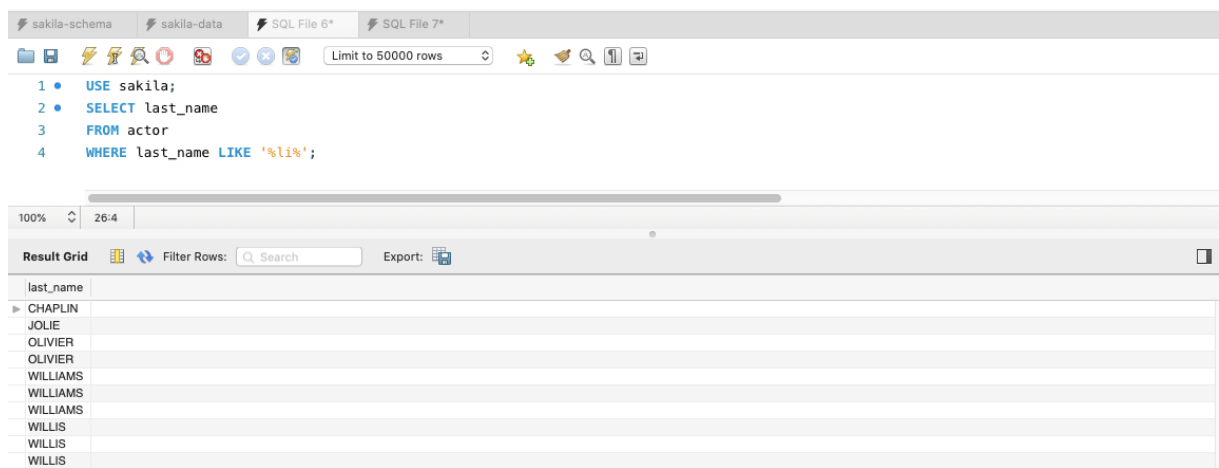
The screenshot shows a SQL IDE interface with a query editor and a result grid. The query is as follows:

```
1 • USE sakila;
2 • SELECT last_name
3 • FROM actor
4 • WHERE last_name LIKE 'gen%';
```

The result grid displays the following data:

last_name
BERGEN
DEGENERES
DEGENERES
DEGENERES

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



The screenshot shows a SQL IDE interface with a query editor and a result grid. The query is as follows:

```
1 • USE sakila;
2 • SELECT last_name
3 • FROM actor
4 • WHERE last_name LIKE '%li%';
```

The result grid displays the following data:

last_name
CHAPLIN
JOLIE
OLIVIER
OLIVIER
WILLIAMS
WILLIAMS
WILLIAMS
WILLIS
WILLIS
WILLIS

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

SQL Query Editor Interface:

```

1  USE sakila;
2  • SELECT last_name, COUNT(last_name)
3  FROM actor
4  GROUP BY last_name
5  ORDER BY COUNT(last_name) DESC;

```

Result Grid:

last_name	COUNT(last_name)
KILMER	5
NOLTE	4
TEMPLE	4
AKROYD	3
ALLEN	3
BERRY	3
DAVIS	3
DEGENERES	3
GARLAND	3
GUINNESS	3
HARRIS	3
HOFFMAN	3
HOPKINS	3
JOHANSSON	3
KEITEL	2

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 Query Editor Interface:

```

1  USE sakila;
2  • SELECT last_name, COUNT(last_name) AS actor_count
3  FROM actor
4  GROUP BY last_name
5  HAVING actor_count >= 2
6  ORDER BY actor_count DESC;
7
8

```

Result Grid:

last_name	actor_count
KILMER	5
TEMPLE	4
NOLTE	4
ZELLWEGER	3
BERRY	3
WILLIS	3
WILLIAMS	3
TORN	3
AKROYD	3
PECK	3
ALLEN	3
DAVIS	3
KEITEL	2

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

sakila-schema sakila-data SQL File 6* SQL File 7* Limit to 50000 rows

```

1 • USE sakila;
2 • SELECT staff.staff_id, staff.first_name, staff.last_name, SUM(payment.amount) AS total_amount
3 FROM payment
4 JOIN staff ON payment.staff_id = staff.staff_id
5 WHERE MONTH(payment.payment_date) = 8 AND YEAR(payment.payment_date) = 2005
6 GROUP BY staff.staff_id, staff.first_name, staff.last_name
7 ORDER BY total_amount DESC;
8
9

```

100% 1:9

Result Grid Filter Rows: Search Export:

	staff_id	first_name	last_name	total_amount
2	Jon	Stephens	12216.49	
1	Mike	Hillier	11853.65	

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

sakila-schema sakila-data SQL File 6* SQL File 7* Limit to 50000 rows

```

1 • USE sakila;
2 • SELECT film.title
3 FROM film
4 JOIN language ON film.language_id = language.language_id
5 WHERE (film.title LIKE 'K%' OR film.title LIKE 'Q%')
6 AND language.name = 'English';
7

```

100% 1:6

Result Grid Filter Rows: Search Export:

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

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

The screenshot shows a SQL IDE with a query window and a result grid. The query is as follows:

```

1 SELECT customer.first_name, customer.last_name, customer.email
2 FROM sakila.customer
3 INNER JOIN sakila.address ON customer.address_id = address.address_id
4 INNER JOIN sakila.city ON address.city_id = city.city_id
5 INNER JOIN sakila.country ON city.country_id = country.country_id
6 WHERE country.country = 'Canada';
7
8
9

```

The result grid shows the following data:

first_name	last_name	email
DERRICK	BOURQUE	DERRI...
DARRELL	POWER	DARRE...
LORETTA	CARPENTER	LORET...
CURTIS	IRBY	CURTIS...
TROY	QUIGLEY	TROY.Q...

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

The screenshot shows a SQL IDE with a query window and a result grid. The query is as follows:

```

27 SELECT s.store_id,
28        CONCAT(MONTHNAME(p.payment_date), ' ', YEAR(p.payment_date)) AS month_year,
29        SUM(p.amount) AS total_sales
30 FROM payment p
31 JOIN staff st ON p.staff_id = st.staff_id
32 JOIN store s ON st.store_id = s.store_id
33 WHERE YEAR(p.payment_date) = 2005
34 GROUP BY s.store_id, CONCAT(MONTHNAME(p.payment_date), ' ', YEAR(p.payment_date))
35 ORDER BY store_id, month_year desc;
36
37

```

The result grid shows the following data:

store_id	month_year	total_sales
1	May 2005	2621.83
1	June 2005	4774.37
1	July 2005	13998.56
1	August 2005	11853.65
2	May 2005	2201.61
2	June 2005	4855.52
2	July 2005	14370.35
2	August 2005	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)

The screenshot shows a SQL IDE with a query editor at the top and a result grid at the bottom. The query is as follows:

```

1 • SELECT film.title,
2       customer.first_name,
3       customer.last_name,
4       address.phone,
5       address.address
6 FROM sakila.rental
7 INNER JOIN sakila.inventory ON rental.inventory_id = inventory.inventory_id
8 INNER JOIN sakila.film ON inventory.film_id = film.film_id
9 INNER JOIN sakila.customer ON rental.customer_id = customer.customer_id
10 INNER JOIN sakila.address ON customer.address_id = address.address_id
11 WHERE rental.return_date IS NULL;
12
13

```

The result grid below shows the following data:

title	first_name	last_name	phone	address
HUNGER ROOF	GREGORY	MAULDIN	80303246192	507 Smolensk Loop
FRISCO FORREST	LOUISE	JENKINS	800716535041	929 Tallahassee Loop
TITANS JERK	WILLIE	HOWELL	991802825778	1244 Allappuzha (Alleppey) Place
CONNECTION MICROCOSMOS	EMILY	DIAZ	333339908719	588 Vila Velha Manor
HAUNTED ANTTITRUST	LAURIE	LAWRENCE	956188728558	9 San Miguel de Tucumán Manor
BULL SHAWSHANK	LISA	ANDERSON	635297277345	1542 Tarlac Parkway
GHOST GROUNDHOG	FREDDIE	DUGGAN	644021380889	1103 Quilmes Boulevard
PEACH INNOCENT	HEATHER	MORRIS	697760867968	17 Kabul Boulevard
SUIT WALLS	ROLAND	SOUTH	25865528181	1993 0 Loop
WOMEN DORADO	NATALIE	MEYER	873492228462	1201 Qomshesh Manor

2ème partie : Test technique (type entreprise)

1. How can SQL queries be optimized?

Pour optimiser les requêtes SQL, nous pouvons adopter plusieurs méthodes à savoir : l'indexation avec de bonnes clauses pour accélérer la recherche des données, l'utilisation de bon type de jointures pour éviter les opérations inutiles, la réduction des charges de données en sélectionnant les colonnes nécessaires, utiliser de préférence la fonction d'agrégation GROUP BY et la condition de filtrage WHERE pour réduire le jeu de résultats. Penser toutefois à limiter si possible, l'utilisation de GROUP BY et ORDER BY sur des grandes tables car cela peut être consommateur de ressources.

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

Pour supprimer les lignes en doublon d'une table, nous pouvons utiliser la méthode DELETE avec « ROW_NUMBER() » pour indiquer la ligne :

The screenshot shows a SQL IDE with a query editor. The query is as follows:

```

1 WITH duplicates AS (
2   SELECT *, ROW_NUMBER() OVER(PARTITION BY column_name ORDER BY id) AS row_num
3   FROM table_name
4 )
5 DELETE FROM table_name
6 WHERE id IN (
7   SELECT id FROM duplicates WHERE row_num > 1
8 );
9

```

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

Les principales différences entre les clauses SQL HAVING et WHERE sont :

La clause WHERE ne renvoie que les lignes qui correspondent à ses critères et utilisée avant l'agrégation des données. Cependant, la fonction HAVING peut être utilisée avec des fonctions agrégées en regroupant plusieurs lignes en fonction des certains critères pour former une valeur unique. Les fonctions agrégées courantes sont COUNT(), SUM(), MIN(), MAX() etc.

4. What is the difference between normalization and denormalization?

La normalisation organise les tables pour réduire la redondance et la dépendance des données. Elle implique de diviser une base de données en plusieurs tables distinctes et de définir des relations entre elles, créant ainsi une structure plus efficace. En revanche, **la dénormalisation** combine certaines de ces tables afin de simplifier les requêtes. Bien qu'elle puisse introduire une certaine redondance, la dénormalisation améliore souvent les performances en réduisant le nombre de jointures nécessaires.

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

- L'instruction DELETE est une commande **DML (Data Manipulation Language)**. Elle permet de supprimer une ou plusieurs lignes (enregistrements) d'une table. En revanche la commande TRUNCATE est une commande **DDL (Data Definition Language)**. Elle est utilisée sur les tables pour supprimer tous les enregistrements à la fois.
- L'instruction DELETE utilise la condition WHERE contrairement à la commande TRUNCATE qui n'utilise aucune condition.
- Dans une requête DELETE, un journal **est créé pour chaque ligne du journal des transactions** . De cette façon, nous pouvons récupérer les enregistrements en utilisant ROLLBACK **avant COMMIT** . De plus, lorsque nous utilisons TRUNCATE, nous pouvons récupérer les enregistrements en utilisant ROLLBACK. La différence est que TRUNCATE **enregistre uniquement la désallocation de la page où les données sont stockées**.

DIFFÉRENCES	
TRUNCATE	DELETE
<ul style="list-style-type: none"> • Elle supprime toutes les lignes d'une table, ce qui est plus rapide et n'utilise pas autant d'espace d'annulation qu'une suppression. • Il s'agit d'une commande DDL qui modifie la structure de la table. • Il n'est pas possible de revenir en arrière avec TRUNCATE • En SQL, le compteur d'incrément automatique est réinitialisé avec TRUNCATE. 	<ul style="list-style-type: none"> • Elle est utilisée pour supprimer des lignes d'un tableau. Une clause WHERE peut être utilisée pour ne supprimer que certaines lignes. • Il s'agit d'une commande DML. Elle ne supprime que les lignes d'une table, en laissant la structure intacte. • Dans DELETE, on peut revenir en arrière • Le compteur d'incrément automatique ne peut pas être réinitialisé par la suppression.

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

Pour éviter les entrées en double lors de l'élaboration d'une requête nous pouvons utiliser plusieurs stratégies :

- Utiliser DISTINCT pour éliminer les doublons dans les résultats.
- Définir des contraintes d'unicité sur les colonnes devant rester uniques (les clés primaires)
- Vérifier l'existence de doublons avant l'insertion en utilisant des requêtes conditionnelles ou des triggers
- Utiliser la fonction ROW_NUMBER() pour identifier et gérer les doublons
- Nettoyer régulièrement les données pour supprimer les doublons existants.
- Mettre en place des audits réguliers

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

Les types de relations en SQL sont :

- La relation **Un à un** : Une relation un-à-un dans une base de données se produit lorsque chaque ligne de la table 1 n'a qu'une seule ligne associée dans la table 2.
- La relation **Un-à-plusieurs** : C'est le type de relation le plus couramment utilisé. Une relation un-à-plusieurs se produit lorsqu'un enregistrement de la table 1 est lié à un ou plusieurs enregistrements de la table 2. En revanche, un enregistrement de la table 2 ne peut pas être lié à plus d'un enregistrement de la table 1

- La relation **plusieurs à plusieurs** : Une relation plusieurs-à-plusieurs se produit lorsque plusieurs enregistrements d'une table sont liés à plusieurs enregistrements d'une autre table.
- Les relations **autoréférencées** : Une relation d'auto-référencement (également appelée relation récursive) dans une base de données se produit lorsqu'une colonne d'une table est liée à une autre colonne de la même table. Dans une telle relation, une seule table est concernée.
- La relation **plusieurs-à-un** : Moins distinguée par de nombreux experts car, similaire à la relation un-à-plusieurs .

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.

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.

8.1:explication de notre structure relationnelle nous avons :

1. Les programmes sont centralisés dans la table Master.
2. Les matières sont associées à des programmes par la clé étrangère Master_id dans subject.
3. Les étudiants sont associés à des programmes et à des matières par les clés étrangères Master_id et subject_id dans student.


```

60 • create table Master (
61     Master_id INT PRIMARY KEY,
62     name VARCHAR(100)
63 );
64
65 /* ajout de la colonne Master_id dans subject et student*/
66
67 • alter table subject
68     add Master_id int;
69
70 • alter table student
71     add Master_id int;
72
73 /*definit Master_id comme clé étrangere pour relier aux tables subject et student*/
74 • alter table subject
75     add constraint fk_subject_master
76     foreign key (Master_id) references Master(Master_id);
77
78 • alter table student
79     add constraint fk_student_master
80     foreign key (Master_id) references Master(Master_id);
81
82 /*insertion des données dans master */
83
84 • insert into master(master_id, name)
85     values(1,'Informatique');
86
87 /*insertion des données dans subject avec master_id*/
88
89 • insert into subject (subject_id, subject_name, max_score, lecturer, Master_id)
90     VALUES (17, 'Data Analytics', 100, 'Dr. Alice', 1);
91
92 /*insertion des données student avec master_id*/
93 • insert into student (student_id, student_name, city, subject_id, Master_id)
94     VALUES (2016, 'John Doe', 'Los Angeles', 17, 1);
95 • describe student;
96 • show columns from student;
97
98 • select*from student;
99

```

Result Grid					
Filter Rows:					
Edit: Export/Import: Wrap Cell Content:					
student_id	student_name	city	subject_id	Master_id	
2005	Bradley Camer	Stanford	11	NULL	
2006	Sofia Mueller	Boston	16	NULL	
2007	Rory Pietman	New Haven	12	NULL	
2008	Carly Walsh	Tulsa	14	NULL	
2011	Richard Curtis	Boston	11	NULL	
2012	Cassey Ledgers	Stanford	11	NULL	
2013	Harold Ledgers	Miami	13	NULL	
2014	Davey Bergman	San Francisco	12	NULL	
2015	Darcey Button	Chicago	14	NULL	
2016	John Doe	Los Angeles	17	1	
•	NULL	NULL	NULL	NULL	

student 1 x

Output

8.2. 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.

```

100
101  /* 8.2 afficher les cours avec au moins 3 etudiants inscrits */
102  • select subject.subject_name, COUNT(student.student_id) AS nombre_etudiants_inscrits
103      from subject
104      join student ON subject.subject_id = student.subject_id
105      group by subject.subject_name
106      having COUNT(student.student_id) >= 3;
107
108

```

subject_name	nombre_etudiants_inscrits
Math	4
Computer Science	3

9. Write a query to retrieve the order_id , customer_id, and total from the orders table where the total 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.

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

```

151  • SELECT order_id, customer_id, total
152      FROM Orders
153      WHERE total > 400;

```

order_id	customer_id	total
4	103	500.00
5	104	600.00
NULL	NULL	NULL

9.2. 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.

```

155  /* 9.2 . Récupérer le customer_id et le montant total dépensé par chaque client, trié par montant total en ordre décroissant */
156
157  • SELECT Orders.customer_id AS customer_id, SUM(Order_items.quantity * Order_items.price) AS Total_spending
158      FROM Orders
159      INNER JOIN Order_items ON Orders.order_id=Order_items.order_id
160      GROUP BY Orders.customer_id
161      ORDER BY Total_spending DESC;

```

customer_id	Total_spending
101	220.00
100	175.00
102	160.00
103	76.00
104	52.00

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

```

163      /* 10. afficher la quantité total vendu pour chaque produit en faisant une jointure*/
164
165 •   SELECT product_id, SUM(quantity) AS total_quantite
166      FROM Order_items
167      GROUP BY product_id
168      ORDER BY total_quantite DESC;
169
170

```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	product_id	total_quantite			
▶	19	11			
	18	10			
	17	9			
	16	8			
	15	7			
	14	6			
	13	5			
	12	4			
	11	3			
	10	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.

```

233 • ALTER TABLE Orders
234     ADD CONSTRAINT fk_customer
235     FOREIGN KEY (customer_id) REFERENCES Customers(id);
236
237 /*verification des relations*/
238 • SELECT
239     Orders.order_id,
240     Orders.customer_id,
241     Orders.order_date,
242     Orders.total,
243     OrderDetails.product,
244     OrderDetails.quantity,
245     OrderDetails.price
246 FROM Orders
247 JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id;
248
249 /*total des quantités vendues pour chaque produit*/
250
251 • SELECT product, SUM(quantity) AS total_des_quantités_vendues
252 FROM OrderDetails /* contient les détails de chaque commande, y compris les produits et leurs quantités.*/
253 GROUP BY product;
254
255 /* montant total depense par chaque client */
256 • SELECT Customers.id AS customer_id, Customers.name, SUM(Orders.total) AS montant_total
257 FROM Customers
258 JOIN Orders ON Customers.id = Orders.customer_id
259 GROUP BY Customers.id, Customers.name
260 ORDER BY montant_total DESC;
261
262
263
264 • CREATE TABLE CustomerOrders (
265     customer_id INT ,
266     customer_name VARCHAR(100),
267     customer_address VARCHAR(255),
268     customer_city VARCHAR(100),
269     customer_country VARCHAR(100),
270     order_id INT,
271     order_date DATE,
272     order_total DECIMAL(10, 2),
273     product_id INT,
274     product_name VARCHAR(100),
275     quantity INT,
276     price DECIMAL(10, 2),
277     PRIMARY KEY (order_id, product_id)
278 );
279
280
281 • INSERT INTO CustomerOrders (customer_id, customer_name, customer_address, customer_city, customer_country,
282     order_id, order_date, order_total, product_id, product_name, quantity, price)
283 VALUES
284 (100, 'Customer 100', '123 Main St.', 'Anytown', 'USA', 1, '2021-01-01', 200, 10, 'Widget A', 2, 25),
285 (100, 'Customer 100', '123 Main St.', 'Anytown', 'USA', 1, '2021-01-01', 200, 11, 'Widget B', 1, 50),
286 (101, 'Customer 101', '456 Oak St.', 'Somewhere', 'USA', 2, '2021-02-02', 300, 12, 'Widget C', 1, 75),
287 (101, 'Customer 101', '456 Oak St.', 'Somewhere', 'USA', 2, '2021-02-02', 300, 13, 'Widget D', 2, 37.5);
288
289 • select*from CustomerOrders;
290
291

```

Result Grid											
customer_id	customer_name	customer_address	customer_city	customer_country	order_id	order_date	order_total	product_id	product_name	quantity	price
100	Customer 100	123 Main St.	Anytown	USA	1	2021-01-01	200.00	10	Widget A	2	25.00
100	Customer 100	123 Main St.	Anytown	USA	1	2021-01-01	200.00	11	Widget B	1	50.00
101	Customer 101	456 Oak St.	Somewhere	USA	2	2021-02-02	300.00	12	Widget C	1	75.00
101	Customer 101	456 Oak St.	Somewhere	USA	2	2021-02-02	300.00	13	Widget D	2	37.50
* NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

CustomerOrders 8 x

Output