

Interview Questions Summary



Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Shippings

shipping_id	status	customer
1	Pending	2
2	Pending	4
3	Delivered	3
4	Pending	5
5	Delivered	1

how many different type of question can be asked in interview from above table Customers [-]

customer_id

[int]

first_name

[varchar(100)]

last_name

[varchar(100)]

age

[int]

country

[varchar(100)]

Orders [-]

order_id

[integer]

item

[varchar(100)]

amount

[integer]

customer_id

[integer]

Shippings [-]

shipping_id

[integer]

status

```
[integer]
customer
[integer]
```



Based on the given tables and their structures, here are some different types of questions that could be asked in an interview:

General Understanding and Interpretation:

1. Table Structure and Relationships:

- Explain the relationships between the `Customers`, `Orders`, and `Shippings` tables.
- What type of relationship exists between `Customers` and `Orders`?

2. Table Schema and Data Types:

- Describe the schema of the `Customers` table.
- What data types are used in the `Orders` table?

Data Retrieval and Querying:

3. Simple Queries:

- Retrieve all customers who are from the USA.
- List all orders along with their corresponding customer names.

4. Joins and Aggregations:

- Write a query to find the total amount spent by each customer.
- Retrieve the list of all items ordered by customers from the UK.

5. Filtering and Conditions:

- Find all orders that have an amount greater than 500.
- List all pending shipments.

6. Nested Queries:

- Write a query to find the names of customers who have pending shipments.
- Retrieve the customer details for those who have ordered a keyboard.

Data Analysis:

7. Statistical Queries:

- Calculate the average age of customers.
- Determine the total number of orders placed by customers from each country.

8. Trends and Patterns:

- Identify the most frequently ordered item.
- Find out which customer has spent the most on orders.

Data Modification and Integrity:

9. Insertions and Updates:

- Write a SQL query to add a new customer to the `Customers` table.
- Update the status of all pending shipments to "Delivered."

10. Constraints and Indexing:

- Discuss how you would add a foreign key constraint between the `Orders` and `Customers` tables.

- Explain the benefits of indexing the `customer_id` in the `Orders` table.

Advanced Topics:

11. Normalization and Optimization:

- Discuss the normalization forms applied to these tables.
- Suggest any optimizations for these table structures to improve query performance.

12. Stored Procedures and Triggers:

- Write a stored procedure to retrieve all orders for a given customer ID.
- Explain how you would implement a trigger to automatically update the shipping status when an order is placed.

Real-World Scenarios:

13. Business Logic:

- How would you handle a scenario where a customer has multiple pending shipments?
- Describe how you would generate a report of orders and their shipping statuses for a given month.

14. Error Handling:

- What strategies would you use to handle errors during data insertion?
- Discuss how you would ensure data integrity across these tables.

These questions cover a wide range of topics, including understanding table structures, writing SQL queries, data analysis, and database management principles.



give all solutions



Sure, here are the solutions for the proposed questions:

General Understanding and Interpretation:

1. Table Structure and Relationships:

- The `Customers` table contains customer details.
- The `Orders` table records orders placed by customers, with a foreign key (`customer_id`) referencing the `customer_id` in the `Customers` table.
- The `Shipments` table records the shipping status of orders, with a foreign key (`customer_id`) referencing the `customer_id` in the `Customers` table.
- There is a one-to-many relationship between `Customers` and `Orders` (one customer can have many orders).
- There is also a one-to-many relationship between `Customers` and `Shipments` (one customer can have many shipments).

2. Table Schema and Data Types:

- The `Customers` table has the following schema:

css

```
customer_id [int]
first_name [varchar(100)]
last_name [varchar(100)]
age [int]
country [varchar(100)]
```

- The `Orders` table has the following schema:

css

```
order_id [integer]
item [varchar(100)]
amount [integer]
customer_id [integer]
```

Data Retrieval and Querying:

3. Simple Queries:

- Retrieve all customers who are from the USA:

sql

```
SELECT * FROM Customers WHERE country = 'USA';
```

- List all orders along with their corresponding customer names:

sql

```
SELECT Orders.order_id, Orders.item, Orders.amount, Customers.first_name,
Customers.last_name
FROM Orders
JOIN Customers ON Orders.customer_id = Customers.customer_id;
```

4. Joins and Aggregations:

- Find the total amount spent by each customer:

sql

```
SELECT customer_id, SUM(amount) AS total_spent
FROM Orders
GROUP BY customer_id;
```

- Retrieve the list of all items ordered by customers from the UK:

sql

```
SELECT Orders.item
FROM Orders
```

```
JOIN Customers ON Orders.customer_id = Customers.customer_id
WHERE Customers.country = 'UK';
```

5. Filtering and Conditions:

- Find all orders that have an amount greater than 500:

sql

```
SELECT * FROM Orders WHERE amount > 500;
```

- List all pending shipments:

sql

```
SELECT * FROM Shippings WHERE status = 'Pending';
```

6. Nested Queries:

- Find the names of customers who have pending shipments:

sql

```
SELECT first_name, last_name
FROM Customers
WHERE customer_id IN (SELECT customer FROM Shippings WHERE status = 'Pending');
```

- Retrieve the customer details for those who have ordered a keyboard:

sql

```
SELECT DISTINCT Customers.*
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Orders.item = 'Keyboard';
```

Data Analysis:

7. Statistical Queries:

- Calculate the average age of customers:

sql

```
SELECT AVG(age) AS average_age FROM Customers;
```

- Determine the total number of orders placed by customers from each country:

```
sql
```

```
SELECT country, COUNT(Orders.order_id) AS total_orders
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY country;
```

8. Trends and Patterns:

- Identify the most frequently ordered item:

```
sql
```

```
SELECT item, COUNT(item) AS frequency
FROM Orders
GROUP BY item
ORDER BY frequency DESC
LIMIT 1;
```

- Find out which customer has spent the most on orders:

```
sql
```

```
SELECT Customers.first_name, Customers.last_name, SUM(Orders.amount) AS
total_spent
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.customer_id
ORDER BY total_spent DESC
LIMIT 1;
```

Data Modification and Integrity:

9. Insertions and Updates:

- Add a new customer to the `Customers` table:

```
sql
```

```
INSERT INTO Customers (customer_id, first_name, last_name, age, country)
VALUES (6, 'Alice', 'Smith', 30, 'Canada');
```

- Update the status of all pending shipments to "Delivered":

```
sql
```

```
UPDATE Shippings SET status = 'Delivered' WHERE status = 'Pending';
```

10. Constraints and Indexing:

- Add a foreign key constraint between the `Orders` and `Customers` tables:

```
sql
```

```
ALTER TABLE Orders
ADD CONSTRAINT fk_customer
FOREIGN KEY (customer_id)
REFERENCES Customers(customer_id);
```

- Benefits of indexing the `customer_id` in the `Orders` table:
Indexing the `customer_id` in the `Orders` table will improve the performance of queries that filter or join based on this column. It allows the database to quickly locate the relevant rows in the `Orders` table that correspond to a specific customer, reducing the time required for query execution.

Advanced Topics:

11. Normalization and Optimization:

- The given tables seem to be in at least the 2nd normal form (2NF) as they do not have partial dependencies. Each non-key attribute is fully functionally dependent on the primary key. To further normalize, ensure there are no transitive dependencies (3NF). Review the schema to ensure that each non-key attribute is directly dependent on the primary key.
- To optimize these tables:
 - Ensure that foreign key relationships are indexed.
 - Avoid storing redundant data.
 - Regularly analyze query performance and adjust indexing strategies accordingly.

12. Stored Procedures and Triggers:

- Write a stored procedure to retrieve all orders for a given customer ID:

```
sql
```

```
CREATE PROCEDURE GetCustomerOrders (IN customerId INT)
BEGIN
  SELECT * FROM Orders WHERE customer_id = customerId;
END;
```

- Implement a trigger to automatically update the shipping status when an order is placed:

```
sql
```

```
CREATE TRIGGER update_shipping_status
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
  UPDATE Shippings
  SET status = 'Pending'
  WHERE customer = NEW.customer_id;
END;
```

Real-World Scenarios:

13. Business Logic:

- Handle a scenario where a customer has multiple pending shipments:
You could write a query to prioritize shipments based on certain criteria (e.g., earliest order

date). Alternatively, consolidate shipments where possible to streamline the process.

- Generate a report of orders and their shipping statuses for a given month:

sql

```
SELECT Orders.order_id, Orders.item, Orders.amount, Shippings.status
FROM Orders
JOIN Shippings ON Orders.customer_id = Shippings.customer
WHERE MONTH(Orders.order_date) = 1 AND YEAR(Orders.order_date) = 2024;
```

14. Error Handling:

- Handle errors during data insertion:
Use transactions to ensure atomicity. If an error occurs during the transaction, roll back the changes to maintain data integrity.

sql

```
START TRANSACTION;
-- Insert data
INSERT INTO Customers (customer_id, first_name, last_name, age, country) VALUES
(7, 'Eve', 'Johnson', 35, 'Australia');
-- Check for errors
IF (some_error_condition) THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
```

- Ensure data integrity across these tables:
Implement foreign key constraints to enforce referential integrity. Use triggers and stored procedures to handle complex business rules and maintain consistency. Regularly back up the database and perform data validation checks.

These solutions provide a comprehensive set of answers to the different types of questions that can be asked based on the given tables.



```
select Customers.first_name, Customers.country from Customers join
Shippings on Shippings.shipping_id==Customers.customer_id
where status=='Pending'; and this queries are same SELECT first_name, last_name
FROM Customers
WHERE customer_id IN (SELECT customer FROM Shippings WHERE status = 'Pending');
```



There are a couple of issues in your provided SQL query, including syntax errors and incorrect join conditions. Here are the corrected versions and clarifications:

1. Corrected Query:


```
sql
```

```
SELECT Customers.first_name, Customers.country
FROM Customers
JOIN Shippings ON Shippings.customer = Customers.customer_id
WHERE Shippings.status = 'Pending';
```

Explanation:

- Corrected the join condition to match `Shippings.customer` with `Customers.customer_id`.
- Used single equals `=` for comparisons in SQL.
- The column `status` in the `WHERE` clause should use a single equals `=`.

2. The second query you provided is:

```
sql
```

```
SELECT first_name, last_name
FROM Customers
WHERE customer_id IN (SELECT customer FROM Shippings WHERE status = 'Pending');
```

Explanation:

- This query retrieves the first name and last name of customers who have pending shipments using a subquery.

Comparing the Queries:

- Both queries aim to retrieve information about customers with pending shipments.
- The first query directly joins the `Customers` and `Shippings` tables and filters by status.
- The second query uses a subquery to achieve the same result.

Key Points:

- Both queries are correct and should return the same results but in different formats (one includes country, the other includes last name).
- The first query might be more efficient as it uses a join instead of a subquery, depending on the database engine and indexing.

To ensure completeness:

If you want the second query to also return the `country`, you would write:

```
sql
```

```
SELECT first_name, last_name, country
FROM Customers
WHERE customer_id IN (SELECT customer FROM Shippings WHERE status = 'Pending');
```

Conclusion:

Both queries are logically equivalent for finding customers with pending shipments but differ in their approach and the specific columns they return.



also i want to revise my all sql concept and any other variation interviewer can asked plus more on join like self, right, full left



Revising SQL concepts and understanding different types of joins can greatly help in interview preparation. Here's a comprehensive overview:

Basic SQL Concepts:

1. Data Types:

- `INT`, `VARCHAR`, `CHAR`, `TEXT`, `DATE`, `BOOLEAN`, etc.

2. DDL (Data Definition Language):

- `CREATE`, `ALTER`, `DROP`, `TRUNCATE`

3. DML (Data Manipulation Language):

- `SELECT`, `INSERT`, `UPDATE`, `DELETE`

4. DCL (Data Control Language):

- `GRANT`, `REVOKE`

5. TCL (Transaction Control Language):

- `COMMIT`, `ROLLBACK`, `SAVEPOINT`

Basic SQL Queries:

1. Selecting Data:

sql

```
SELECT column1, column2 FROM table WHERE condition;
```

2. Inserting Data:

sql

```
INSERT INTO table (column1, column2) VALUES (value1, value2);
```

3. Updating Data:

sql

```
UPDATE table SET column1 = value1 WHERE condition;
```

4. Deleting Data:

```
sql
```

```
DELETE FROM table WHERE condition;
```

Advanced SQL Queries:

1. Aggregate Functions:

- `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`

```
sql
```

```
SELECT COUNT(*) FROM table;
```

2. Group By:

```
sql
```

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

3. Having Clause:

```
sql
```

```
SELECT column1, COUNT(*) FROM table GROUP BY column1 HAVING COUNT(*) > 1;
```

4. Order By:

```
sql
```

```
SELECT column1, column2 FROM table ORDER BY column1 ASC/DESC;
```

5. Joins:

- **Inner Join:**

```
sql
```

```
SELECT table1.column1, table2.column2
FROM table1
INNER JOIN table2 ON table1.common_column = table2.common_column;
```

- **Left Join:**

sql

```
SELECT table1.column1, table2.column2
FROM table1
LEFT JOIN table2 ON table1.common_column = table2.common_column;
```

o Right Join:

sql

```
SELECT table1.column1, table2.column2
FROM table1
RIGHT JOIN table2 ON table1.common_column = table2.common_column;
```

o Full Outer Join:

sql

```
SELECT table1.column1, table2.column2
FROM table1
FULL OUTER JOIN table2 ON table1.common_column = table2.common_column;
```

o Self Join:

sql

```
SELECT A.column1, B.column1
FROM table A, table B
WHERE A.common_column = B.common_column;
```

6. Subqueries:

sql

```
SELECT column1
FROM table
WHERE column2 = (SELECT MAX(column2) FROM table);
```

Practical SQL Concepts:

1. Normalization:

- o 1NF, 2NF, 3NF, BCNF
- o Ensuring data redundancy is minimized.

2. Indexing:

- o Creating indexes to speed up query performance.

sql

```
CREATE INDEX index_name ON table(column);
```

3. Views:

- Virtual tables.

sql

```
CREATE VIEW view_name AS SELECT column1, column2 FROM table WHERE condition;
```

4. Stored Procedures:

sql

```
CREATE PROCEDURE procedure_name (parameters)
BEGIN
  -- SQL Statements
END;
```

5. Triggers:

sql

```
CREATE TRIGGER trigger_name
BEFORE/AFTER INSERT/UPDATE/DELETE ON table
FOR EACH ROW
BEGIN
  -- SQL Statements
END;
```

Interview Question Variations:

1. Complex Queries:

- Writing queries that combine multiple joins, subqueries, and aggregate functions.

2. Scenario-Based Questions:

- Designing database schema for a given scenario.
- Writing SQL queries based on real-world problems.

3. Performance Tuning:

- Explaining how to optimize slow queries.
- Discussing indexing strategies and query optimization techniques.

4. Data Integrity:

- Implementing constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL).
- Ensuring data consistency and handling anomalies.

More Examples:

1. Inner Join Example:

sql

```
SELECT Customers.first_name, Orders.item
FROM Customers
```

```
INNER JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

2. Left Join Example:

```
sql
```

```
SELECT Customers.first_name, Orders.item
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

3. Right Join Example:

```
sql
```

```
SELECT Customers.first_name, Orders.item
FROM Customers
RIGHT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

4. Full Outer Join Example:

```
sql
```

```
SELECT Customers.first_name, Orders.item
FROM Customers
FULL OUTER JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

5. Self Join Example:

```
sql
```

```
SELECT A.first_name, B.first_name
FROM Customers A, Customers B
WHERE A.customer_id <> B.customer_id AND A.last_name = B.last_name;
```

Practice Problems:

1. Find the total amount spent by customers who are older than 25 years:

```
sql
```

```
SELECT Customers.first_name, SUM(Orders.amount) AS total_spent
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Customers.age > 25
GROUP BY Customers.first_name;
```

2. List all customers who have not placed any orders:

sql

```
SELECT first_name, last_name
FROM Customers
WHERE customer_id NOT IN (SELECT customer_id FROM Orders);
```

3. Get the details of the highest amount order:

sql

```
SELECT * FROM Orders
WHERE amount = (SELECT MAX(amount) FROM Orders);
```

By mastering these concepts and variations, you'll be well-prepared for a wide range of SQL interview questions.



Practice Problems: [more](#)



Sure! Here are more practice problems to help you further hone your SQL skills:

Practice Problems:

1. Find the names of customers who have placed an order for more than one type of item:

sql

```
SELECT Customers.first_name, Customers.last_name
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.first_name, Customers.last_name
HAVING COUNT(DISTINCT Orders.item) > 1;
```

2. Retrieve the list of customers and their total expenditure, including customers who haven't placed any orders (using LEFT JOIN):

sql

```
SELECT Customers.first_name, Customers.last_name, COALESCE(SUM(Orders.amount), 0) AS
total_spent
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.first_name, Customers.last_name;
```

3. List the first names of customers who have both 'Pending' and 'Delivered' shipments:

sql

```
SELECT Customers.first_name
FROM Customers
JOIN Shippings ON Customers.customer_id = Shippings.customer
GROUP BY Customers.first_name
HAVING SUM(CASE WHEN Shippings.status = 'Pending' THEN 1 ELSE 0 END) > 0
      AND SUM(CASE WHEN Shippings.status = 'Delivered' THEN 1 ELSE 0 END) > 0;
```

4. Find the details of customers who have placed exactly three orders:

sql

```
SELECT Customers.*
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.customer_id
HAVING COUNT(Orders.order_id) = 3;
```

5. List all the customers who have not received any 'Delivered' shipments:

sql

```
SELECT DISTINCT Customers.first_name, Customers.last_name
FROM Customers
WHERE Customers.customer_id NOT IN (
  SELECT Shippings.customer
  FROM Shippings
  WHERE Shippings.status = 'Delivered'
);
```

6. Retrieve the order details for orders that are associated with 'Pending' shipments:

sql

```
SELECT Orders.*
FROM Orders
JOIN Shippings ON Orders.customer_id = Shippings.customer
WHERE Shippings.status = 'Pending';
```

7. Find the customers who have spent more than the average amount on their orders:

sql

```
SELECT Customers.first_name, Customers.last_name, SUM(Orders.amount) AS total_spent
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.customer_id
HAVING SUM(Orders.amount) > (SELECT AVG(amount) FROM Orders);
```

8. List all items ordered by customers along with the shipping status of their orders:

sql

```
SELECT Customers.first_name, Orders.item, Shippings.status
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
JOIN Shippings ON Customers.customer_id = Shippings.customer;
```

9. Get the count of customers from each country who have placed at least one order:

sql

```
SELECT Customers.country, COUNT(DISTINCT Customers.customer_id) AS customer_count
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.country;
```

10. Find the names of customers who share the same last name as any other customer:

sql

```
SELECT A.first_name, A.last_name
FROM Customers A
JOIN Customers B ON A.last_name = B.last_name AND A.customer_id <> B.customer_id;
```

11. Get the list of customers and their most recent order amount:

sql

```
SELECT Customers.first_name, Customers.last_name, Orders.amount
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Orders.order_id IN (
    SELECT MAX(order_id)
    FROM Orders
    GROUP BY customer_id
);
```

12. List customers who have ordered both 'Keyboard' and 'Mouse':

sql

```
SELECT Customers.first_name, Customers.last_name
FROM Customers
JOIN Orders 01 ON Customers.customer_id = 01.customer_id AND 01.item = 'Keyboard'
JOIN Orders 02 ON Customers.customer_id = 02.customer_id AND 02.item = 'Mouse';
```

13. Find the average age of customers who have placed orders for more than 5000:

sql

```
SELECT AVG(Customers.age) AS average_age
FROM Customers
```

```
JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Orders.amount > 5000;
```

14. Get the details of the customer who placed the most expensive order:

sql

```
SELECT Customers.*
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Orders.amount = (SELECT MAX(amount) FROM Orders);
```

15. List customers who do not have any orders in the 'Orders' table:

sql

```
SELECT Customers.first_name, Customers.last_name
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id
WHERE Orders.order_id IS NULL;
```

16. Get the total number of orders for each item:

sql

```
SELECT item, COUNT(*) AS total_orders
FROM Orders
GROUP BY item;
```

Advanced Practice Problems:

1. Calculate the total amount spent by each customer and categorize them as 'High', 'Medium', or 'Low' spender based on their total expenditure:

sql

```
SELECT Customers.first_name, Customers.last_name,
SUM(Orders.amount) AS total_spent,
CASE
    WHEN SUM(Orders.amount) > 10000 THEN 'High'
    WHEN SUM(Orders.amount) BETWEEN 5000 AND 10000 THEN 'Medium'
    ELSE 'Low'
END AS spender_category
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.first_name, Customers.last_name;
```

2. Find the customer who has the highest number of pending shipments:

sql

```
SELECT Customers.first_name, Customers.last_name, COUNT(Shippings.shipping_id) AS
pending_shipments
```

```
FROM Customers
JOIN Shippings ON Customers.customer_id = Shippings.customer
WHERE Shippings.status = 'Pending'
GROUP BY Customers.first_name, Customers.last_name
ORDER BY pending_shipments DESC
LIMIT 1;
```

3. List the orders and their shipping status, including those that do not have a shipping record:

sql

```
SELECT Orders.order_id, Orders.item, Orders.amount, COALESCE(Shippings.status, 'No Shipping Record') AS shipping_status
FROM Orders
LEFT JOIN Shippings ON Orders.customer_id = Shippings.customer;
```

4. Determine the total number of customers, orders, and shipments in the system:

sql

```
SELECT
  (SELECT COUNT(*) FROM Customers) AS total_customers,
  (SELECT COUNT(*) FROM Orders) AS total_orders,
  (SELECT COUNT(*) FROM Shippings) AS total_shipments;
```

5. Find customers who have orders that haven't been shipped yet:

sql

```
SELECT DISTINCT Customers.first_name, Customers.last_name
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
LEFT JOIN Shippings ON Orders.customer_id = Shippings.customer
WHERE Shippings.shipping_id IS NULL;
```

6. Retrieve the shipping status of orders along with customer details, even if no shipping record exists:

sql

```
SELECT Customers.first_name, Customers.last_name, Orders.item,
COALESCE(Shippings.status, 'Not Shipped') AS shipping_status
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
LEFT JOIN Shippings ON Orders.customer_id = Shippings.customer;
```

By working through these practice problems, you will be able to reinforce your understanding of SQL and be well-prepared for a variety of interview questions.

ChatGPT can make mistakes. Check important info.