Core SQL Skill Areas to Focus For These Assessments:

🔍 1. Multiple Table Joins (INNER, LEFT)

Usage in your problems:

- Joining `Orders`, `Customers`, `Products`, `Order_Items`
- Joining `Students`, `Enrollments`, `Courses`, `Departments`
- Joining `Appointments`, `Doctors`, `Departments`

Must Know:

- Syntax for multi-table joins:

sql

CopyEdit

```
FROM Table1 A
JOIN Table2 B ON A.key = B.key
```

- Difference between `INNER JOIN` and `LEFT JOIN`.

🔍 2. GROUP BY and Aggregation Functions

Usage in your problems:

- Counting number of orders per customer.
- Counting number of enrollments per student.

- Finding maximum or minimum using `MAX()`, `COUNT()`, `SUM()`.

Must Know:

- Aggregate functions: `COUNT()`, `SUM()`, `AVG()`, `MAX()`, `MIN()`
- `GROUP BY` after aggregation:

sql

CopyEdit

```
SELECT customer_id, COUNT(order_id) AS order_count
FROM Orders
GROUP BY customer_id;
```

🔍 3. Subqueries (Single Row & Multi-Row)

Usage in your problems:

- To get max value in a subquery (e.g., highest order, max enrollment).
- To fetch departments or users based on condition from subqueries.

Must Know:

- Scalar subqueries (return single value):

sql

CopyEdit

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

- Subqueries returning multiple rows:

sql

CopyEdit

```
WHERE product_id IN (SELECT product_id FROM ...)
```

🔍 4. Window Functions (RANK, DENSE_RANK, ROW_NUMBER)

Usage in your problems:

- **To rank students/customers based on their counts.**

Must Know:

sql

CopyEdit

```
RANK() OVER (ORDER BY count DESC)

DENSE_RANK() OVER (ORDER BY count DESC)
```

- **Difference between `RANK()` and `DENSE_RANK()`.**

🔍 5. Filtering Using WHERE, IN, NOT IN, EXISTS

Usage in your problems:

- **Excluding 'Jane Smith' or 'Dr. John Williams'.**
- **Filtering based on subquery results.**

Must Know:

- `WHERE column IN (subquery)`
- `WHERE NOT EXISTS (subquery)`
- **Exclusion conditions (`<>`, `!=`).**

🔍 6. Sorting Results Using ORDER BY

Usage in your problems:

- **Sorting by `appointment_date`.**
- **Sorting by ranking columns (`RANK()` output).**

Must Know:

sql

CopyEdit

```
ORDER BY column_name [ASC|DESC]
```

🔍 7. Distinct vs Duplicates Handling

Usage in your problems:

- **`DISTINCT` for unique product lists.**
- **Avoiding duplicate doctors/customers.**

Must Know:

sql

CopyEdit

```
SELECT DISTINCT column FROM table;
```

🔍 8. Aliasing Tables and Columns

Usage in your problems:

- **Aliasing for better readability in multi-joins.**

sql

CopyEdit

```
FROM Orders o JOIN Customers c ON o.customer_id = c.customer_id
```

🔍 9. Derived Columns / Calculated Fields

Usage possibility:

- **Calculating `order_count`, `course_count`, etc.**

Must Know:

sql

CopyEdit

```
COUNT(course_id) AS course_count
```

✅ How to Practice These on Single Datasets (Multiple Tables):

If you use one large dataset (e.g., Retail or University) — all these SQL areas can be tested as:

Area	Possible Single Dataset Example
Joins	Join `Orders` + `Customers` + `Products`
Aggregation + Group By	Count orders per `customer_id` or sales per `product_id`

Subqueries	**Find the customer with the max total order**
Window Functions	**Rank customers by order count**
Filtering / IN / NOT IN	**Customers who bought same product as 'John Doe'**
Sorting	**Sort by latest `order_date`**
DISTINCT	**Unique product categories bought**
Alias	**Use `o`, `c`, `p` for `Orders`, `Customers`, `Products`**
Derived Columns	**Calculate `order_value` or `course_count`**

✅ Summary: Areas to Master for Real SQL Assessment:

Area	Must Be Comfortable With
Multi-table Joins	**2-4 table joins with ON condition**
Group By + Aggregation Functions	**`COUNT()`, `SUM()`, `AVG()`, `MAX()`, `MIN()`**
Subqueries	**Scalar and IN/EXISTS types**

Window Functions	**`RANK()`, `DENSE_RANK()`, `ROW_NUMBER()`**
Filtering using WHERE / IN	**With subquery and direct conditions**
Sorting using ORDER BY	**Ascending/descending order**
DISTINCT usage	**To remove duplicates**
Aliasing	**Tables (`o`, `c`, `e`) and Columns (`total_count`)**
Derived Columns	**Calculated fields using aggregation or expression**

🎯 Pro Tip to Students:

> **"*If you can solve all these using a single dataset with multiple tables (like Retail, Hospital, or University) — you are fully prepared for any SQL assessment — no matter how the question is framed.*"**

Hospital Database Scenario
The hospital management wants to find the patient who had the **longest hospital stay**. You need to return:

- `patient_id`
- `patient_name`
- `admission_days` (total days admitted)

✅ SQL Query:

sql
CopyEdit

```
SELECT
    p.patient_id,
    p.patient_name,
    a.admission_days
FROM
    Admissions a
JOIN
    Patients p ON a.patient_id = p.patient_id
WHERE
    a.admission_days = (
        SELECT MAX(admission_days) FROM Admissions
    );
```

✅ Explanation:

Part	What it does
`JOIN`	Connects `Admissions` and `Patients` tables.
`MAX(admission_days)`	Finds the highest number of admission days (longest stay).
`WHERE`	Filters the row where admission days equals the max value.
Result	Gives the patient with the longest hospital stay.

Practice 2:

```
SELECT DISTINCT
  s.student_name
FROM
  Students s
JOIN
  Enrollments e ON s.student_id = e.student_id
WHERE
  e.course_id IN (
    SELECT DISTINCT e2.course_id
    FROM Students s2
    JOIN Enrollments e2 ON s2.student_id = e2.student_id
```

```
      WHERE s2.student_name = 'Alice Johnson'
   )
AND s.student_name != 'Alice Johnson';
```

Scenario:
Management wants to analyze which customers have placed the most orders.
Write a query to assign both a **RANK()** and a **DENSE_RANK()** to each customer based on the **total number of orders they placed** (in descending order).

Return:

- `customer_id`
- `customer_name`
- `order_count` (total number of orders)
- `rank_by_orders` (using RANK())
- `dense_rank_by_orders` (using DENSE_RANK())

Order the result by `rank_by_orders`.

✅ SQL Query:

```
SELECT
    c.customer_id,
    c.customer_name,
    COUNT(o.order_id) AS order_count,
    RANK() OVER (ORDER BY COUNT(o.order_id) DESC) AS rank_by_orders,
    DENSE_RANK() OVER (ORDER BY COUNT(o.order_id) DESC) AS
dense_rank_by_orders
FROM
    Customers c
JOIN
    Orders o ON c.customer_id = o.customer_id
GROUP BY
    c.customer_id,
```

```
    c.customer_name
ORDER BY
    rank_by_orders;
```

✅ Explanation:

Part	Description
`COUNT(o.order_id)`	Counts total orders placed by each customer.
`RANK()`	Assigns a rank with possible gaps if counts are the same.
`DENSE_RANK()`	Assigns dense ranks — no gaps in ranks.
`GROUP BY`	Groups results per customer.
`ORDER BY rank_by_orders`	Sorts the final output based on RANK.

Hostel Management — Rank Hostels by Number of Occupied Rooms

Problem Statement:
Rank each hostel block by the **total number of rooms occupied** using `RANK()` and `DENSE_RANK()`.

Tables:

- `Rooms (room_id, block_id, is_occupied)`
- `Hostel_Blocks (block_id, block_name)`

Expected Output:

- `block_id`
- `block_name`
- `occupied_count`
- `rank_by_occupancy`
- `dense_rank_by_occupancy`

Library — Rank Books by Number of Times Borrowed

Problem Statement:
Rank each book by how many times it has been borrowed, using `RANK()` and `DENSE_RANK()`.

Tables:

- `Borrow_Records (borrow_id, book_id)`
- `Books (book_id, book_title)`

Expected Output:

- `book_id`
- `book_title`
- `borrow_count`
- `rank_by_borrows`
- `dense_rank_by_borrows`

Gym Management — Session Details with Trainer and Room Info

Problem Statement:
List all workout sessions along with the trainer's name and the gym room where the session took place.

Tables:

- `Workout_Sessions (session_id, trainer_id, room_id, session_date)`
- `Trainers (trainer_id, trainer_name)`
- `Gym_Rooms (room_id, room_name)`

Expected Output:

- `session_id`
- `session_date`
- `trainer_name`
- `room_name`

Library — Most Borrowed Genre by Top Member

Problem Statement:
Find the genre(s) most borrowed by the member who has the **highest number of borrow records**.

Tables:

- `Members (member_id, member_name)`
- `Borrow_Records (borrow_id, member_id, book_id)`
- `Books (book_id, genre_id)`
- `Genres (genre_id, genre_name)`

Expected Output:

- `member_id`
- `member_name`
- `genre_name`
- `borrow_count`

University Hostel — Students in Block With Lowest Occupancy

Problem Statement:
Find all students staying in the hostel block with the **lowest number of rooms occupied**.

Tables:

- `Students (student_id, student_name, room_id)`
- `Rooms (room_id, block_id, is_occupied)`

Expected Output:

- `student_id`
- `student_name`
- `block_id`

Gym Management — Members With Gold Membership and Session Details

Problem Statement:
List all gym members who have a **'Gold' membership** and show their attended session dates and room names.

Tables:

- `Members (member_id, member_name, membership_type)`

- `Workout_Sessions (session_id, member_id, room_id, session_date)`
- `Gym_Rooms (room_id, room_name)`

Expected Output:

- `member_id`
- `member_name`
- `room_name`
- `session_date`