SQL Interview Prep Notes 📚

1. CASE STATEMENTS

Basic Syntax

```sql
SELECT
  column_1,
  CASE
    WHEN condition_1 THEN result_1
    WHEN condition_2 THEN result_2
    ELSE default_result
  END AS new_column_name
FROM table_name;
```

Use Cases

A. Categorizing Data in SELECT

```sql
-- Example: Categorize by followers
SELECT
  character,
  followers,
  CASE
    WHEN followers >= 700000 THEN 'Highly Popular'
    WHEN followers BETWEEN 300000 AND 699999 THEN 'Moderately Popular'
    ELSE 'Less Popular'
  END AS popularity_category
FROM marvel_avengers;
```

B. CASE in WHERE Clause (Conditional Filtering)

```sql
```

```sql
-- Example: Different filters per platform
SELECT actor, character, platform
FROM marvel_avengers
WHERE
  CASE
    WHEN platform = 'Instagram' THEN followers >= 500000
    WHEN platform = 'Twitter' THEN followers >= 200000
    ELSE followers >= 100000
  END;
```

C. CASE with COUNT()

sql

```sql
-- Count based on conditions
SELECT
  platform,
  COUNT(CASE WHEN followers >= 500000 THEN 1 ELSE NULL END) AS popular_count,
  COUNT(CASE WHEN followers < 500000 THEN 1 ELSE NULL END) AS less_popular_count
FROM marvel_avengers
GROUP BY platform;
```

💡 **TIP:** Use `1` for TRUE cases and `NULL` for FALSE - COUNT ignores NULLs!

D. CASE with SUM()

sql

```sql
-- Sum based on conditions
SELECT
  platform,
  SUM(CASE WHEN engagement_rate >= 8.0 THEN followers ELSE 0 END) AS high_engagement_sum,
  SUM(CASE WHEN engagement_rate < 8.0 THEN followers ELSE 0 END) AS low_engagement_sum
FROM marvel_avengers
GROUP BY platform;
```

E. CASE with AVG()

sql

```sql
-- Average based on conditions
SELECT
  platform,
  AVG(CASE WHEN engagement_rate >= 8.0 THEN followers ELSE NULL END) AS avg_high_engagement
FROM marvel_avengers
GROUP BY platform;
```

💡 **TIP:** Use `NULL` in ELSE for AVG (not 0), otherwise 0s affect the average!

Interview Problem Pattern: Device Viewership

```sql
-- Calculate laptop vs mobile views
SELECT
  SUM(CASE WHEN device_type = 'laptop' THEN 1 ELSE 0 END) AS laptop_views,
  SUM(CASE WHEN device_type IN ('tablet', 'phone') THEN 1 ELSE 0 END) AS mobile_views
FROM viewership;
```

---

2. SQL JOINS

The 3 Essential Parts of a JOIN

1. **SELECT** - columns to display
2. **FROM + JOIN** - which tables to combine
3. **ON** - how tables relate to each other

Basic JOIN Syntax

```sql
SELECT *
FROM table1
JOIN table2
  ON table1.common_column = table2.common_column;
```

Example: Spotify Use Case

```sql
```

```sql
-- Join artists and songs tables
SELECT
  artists.artist_name,
  songs.name AS song_name
FROM artists
JOIN songs
  ON artists.artist_id = songs.artist_id;
```

Types of JOINS

A. INNER JOIN (Default JOIN)

Returns **ONLY matching rows** from both tables

```sql
SELECT
  orders.order_id,
  goodreads.book_title
FROM goodreads
INNER JOIN orders
  ON goodreads.book_id = orders.book_id
WHERE goodreads.price >= 20;
```

Visual: Only the overlapping middle section

B. LEFT JOIN

Returns **ALL rows from LEFT table** + matching rows from right table

```sql
SELECT
  orders.order_id,
  deliveries.delivery_status
FROM orders
LEFT JOIN deliveries
  ON orders.order_id = deliveries.order_id;
```

Result: All orders shown, NULL for undelivered orders

C. RIGHT JOIN (Rarely Used)

Returns **ALL rows from RIGHT table** + matching rows from left table

```sql
SELECT
  orders.order_id,
  deliveries.delivery_status
FROM deliveries
RIGHT JOIN orders
  ON deliveries.order_id = orders.order_id;
```

💡 **TIP:** RIGHT JOIN is rarely used - just swap tables and use LEFT JOIN instead!

D. FULL OUTER JOIN

Returns **ALL rows from BOTH tables**, with NULLs where no match

```sql
SELECT
  orders.order_id,
  deliveries.delivery_status
FROM orders
FULL OUTER JOIN deliveries
  ON orders.order_id = deliveries.order_id;
```

⚠️ **WARNING:** Can introduce duplicate rows - use carefully!

Conditional JOINS (Multiple Conditions)

Single Condition

```sql
SELECT g.book_title, o.quantity
FROM goodreads AS g
INNER JOIN orders AS o
  ON g.book_id = o.book_id
    AND o.quantity > 2;
```

Multiple Conditions

```sql
```

```sql
SELECT g.book_title, g.author, o.order_date
FROM goodreads AS g
INNER JOIN orders AS o
  ON g.book_id = o.book_id
    AND g.year_released > 2015
    AND o.quantity > 1;
```

Joining 3+ Tables

sql

```sql
SELECT
  g.book_title,
  o.order_date,
  d.delivery_status
FROM goodreads g
JOIN orders o
  ON g.book_id = o.book_id
    AND g.book_rating > 4.0
INNER JOIN deliveries d
  ON o.order_id = d.order_id
    AND d.delivery_status = 'Delivered';
```

Interview Problem Pattern: Top Cities by Orders

sql

```sql
-- Find top 3 cities with most completed trades
SELECT
  users.city,
  COUNT(trades.order_id) AS total_orders
FROM trades
JOIN users
  ON trades.user_id = users.user_id
WHERE trades.status = 'Completed'
GROUP BY users.city
ORDER BY total_orders DESC
LIMIT 3;
```

Handling NULLs with JOINs

sql

```sql
-- Find Facebook pages with ZERO likes
SELECT pages.page_id
FROM pages
LEFT JOIN likes
  ON pages.page_id = likes.page_id
WHERE likes.page_id IS NULL;
```

💡 **KEY INSIGHT:** LEFT JOIN + IS NULL = find records with no match!

---

3. DATE & TIME FUNCTIONS

A. Current Date/Time Functions

sql

```sql
SELECT
  CURRENT_DATE,          -- 2023-08-27
  CURRENT_TIME,          -- 07:35:15
  CURRENT_TIMESTAMP,     -- 2023-08-27 07:35:15
  NOW()                  -- Same as CURRENT_TIMESTAMP
FROM messages;
```

B. Comparing Dates

sql

```sql
-- Greater than
WHERE sent_date > '2022-08-10 00:00:00'

-- Greater than or equal to
WHERE sent_date >= '2022-08-10 00:00:00'

-- Equal to
WHERE sent_date = '2022-08-10 17:03:00'

-- Between dates
WHERE sent_date BETWEEN '2022-01-01' AND '2022-12-31'
```

C. EXTRACT() - Get Specific Parts

sql

```sql
SELECT
  sent_date,
  EXTRACT(YEAR FROM sent_date) AS year,
  EXTRACT(MONTH FROM sent_date) AS month,
  EXTRACT(DAY FROM sent_date) AS day,
  EXTRACT(HOUR FROM sent_date) AS hour,
  EXTRACT(MINUTE FROM sent_date) AS minute
FROM messages;
```

Alternative: `DATE_PART('year', sent_date)` - same result!

D. DATE_TRUNC() - Round Down Dates

```sql
SELECT
  sent_date,
  DATE_TRUNC('month', sent_date) AS month_start,  -- 2022-08-01 00:00:00
  DATE_TRUNC('day', sent_date) AS day_start,      -- 2022-08-03 00:00:00
  DATE_TRUNC('hour', sent_date) AS hour_start     -- 2022-08-03 16:00:00
FROM messages;
```

Use Case: Group by month/day/hour for time-series analysis

E. INTERVAL - Add/Subtract Time

```sql
SELECT
  sent_date,
  sent_date + INTERVAL '2 days' AS plus_2days,
  sent_date - INTERVAL '3 days' AS minus_3days,
  sent_date + INTERVAL '2 hours' AS plus_2hours,
  sent_date - INTERVAL '10 minutes' AS minus_10mins
FROM messages;
```

F. TO_CHAR() - Format Dates as Strings

```sql
```

```sql
SELECT
  sent_date,
  TO_CHAR(sent_date, 'YYYY-MM-DD') AS iso_date,          -- 2022-08-03
  TO_CHAR(sent_date, 'Month DDth, YYYY') AS long_format,  -- August 03rd, 2022
  TO_CHAR(sent_date, 'Mon DD, YYYY') AS short_format,     -- Aug 03, 2022
  TO_CHAR(sent_date, 'Day') AS day_of_week                -- Wednesday
FROM messages;
```

Common Formats:

- `'YYYY-MM-DD HH24:MI:SS'` - ISO format
- `'YYYY-MM-DD HH:MI:SS AM'` - 12-hour format
- `'Month DDth, YYYY'` - Long month name

G. Casting - Convert Strings to Date/Timestamp

```sql
SELECT
  -- String to DATE
  '2023-08-27'::DATE AS casted_date,
  TO_DATE('2023-08-27', 'YYYY-MM-DD') AS converted_date,

  -- String to TIMESTAMP
  '2023-08-27 10:30:00'::TIMESTAMP AS casted_timestamp,
  TO_TIMESTAMP('2023-08-27 10:30:00', 'YYYY-MM-DD HH:MI:SS') AS converted_timestamp
FROM messages;
```

Interview Problem Pattern: Days Between Posts

```sql
-- Find days between first and last post in 2021
SELECT
  user_id,
  MAX(post_date::DATE) - MIN(post_date::DATE) AS days_between
FROM posts
WHERE EXTRACT(YEAR FROM post_date) = 2021
GROUP BY user_id
HAVING COUNT(post_id) >= 2;
```

💡 **KEY TRICKS:**

- Use `MIN()` and `MAX()` on dates
- Subtract dates to get number of days
- Cast to DATE to ignore time: `post_date::DATE`

Interview Problem Pattern: Confirm Within 24 Hours

```sql
-- Users who confirmed on second day (not first)
SELECT user_id
FROM emails
JOIN texts
  ON emails.email_id = texts.email_id
WHERE texts.signup_action = 'Confirmed'
  AND texts.action_date = emails.signup_date + INTERVAL '1 day';
```

QUICK REFERENCE CHEAT SHEET

When to Use Each Technique

Task	Use This
Categorize data	`CASE` in SELECT
Conditional filtering	`CASE` in WHERE
Count by category	`COUNT(CASE WHEN ... THEN 1 END)`
Sum by category	`SUM(CASE WHEN ... THEN value END)`
Combine 2 tables	`INNER JOIN`
Show all from one table	`LEFT JOIN`
Find missing matches	`LEFT JOIN ... WHERE ... IS NULL`
Get current date/time	`CURRENT_DATE`, `NOW()`
Extract year/month/day	`EXTRACT()` or `DATE_PART()`
Group by month/day	`DATE_TRUNC()`
Add/subtract time	`INTERVAL`
Date difference	`date1 - date2` (returns days)

Common Interview Patterns

1. **Segmentation:** Use CASE to create categories

2. **Pivot/Transpose:** Use CASE with aggregate functions
3. **Multiple tables:** Use appropriate JOIN type
4. **Time-series:** Use DATE_TRUNC + GROUP BY
5. **Date ranges:** Use EXTRACT + WHERE
6. **Missing data:** Use LEFT JOIN + IS NULL

Pro Tips 💡

- **CASE with COUNT:** Use `1` for TRUE, `NULL` for FALSE
- **CASE with AVG:** Use `NULL` in ELSE (not 0)
- **CASE with SUM:** Use `0` in ELSE is fine
- **RIGHT JOIN:** Almost never needed - use LEFT JOIN
- **FULL OUTER JOIN:** Watch for duplicates
- **Date subtraction:** Returns integer (number of days)
- **Always GROUP BY:** When using aggregate functions with categories
- **Aliasing:** Use `AS` to name your columns clearly

Practice Problem Templates

Template 1: Device/Category Split

sql

```sql
SELECT
  SUM(CASE WHEN category = 'A' THEN 1 ELSE 0 END) AS category_a_count,
  SUM(CASE WHEN category = 'B' THEN 1 ELSE 0 END) AS category_b_count
FROM table_name;
```

Template 2: Top N with JOIN

sql

```
SELECT column1, COUNT(*) AS count
FROM table1
JOIN table2 ON table1.id = table2.id
WHERE condition
GROUP BY column1
ORDER BY count DESC
LIMIT n;
```

Template 3: Date Range Analysis

```
sql

SELECT
  DATE_TRUNC('month', date_column) AS month,
  COUNT(*) AS total
FROM table_name
WHERE EXTRACT(YEAR FROM date_column) = 2021
GROUP BY month
ORDER BY month;
```

Template 4: Find Missing Matches

```
sql

SELECT table1.id
FROM table1
LEFT JOIN table2 ON table1.id = table2.id
WHERE table2.id IS NULL;
```

Key Differences: PostgreSQL vs MySQL

Date Functions:

- PostgreSQL: `DATE_TRUNC()`, `EXTRACT()`, `INTERVAL`
- MySQL: `DATE_FORMAT()`, `YEAR()`, `MONTH()`, `DATE_ADD()`

For interviews: Focus on PostgreSQL syntax (more common in data roles)

Final Interview Checklist ✅

- [ ] Can write CASE statements in SELECT and WHERE

- [ ] Know when to use COUNT/SUM/AVG with CASE
- [ ] Understand all 4 JOIN types (focus on INNER and LEFT)
- [ ] Can join 3+ tables with multiple conditions
- [ ] Know how to find missing records (LEFT JOIN + IS NULL)
- [ ] Can extract and format dates
- [ ] Can calculate date differences
- [ ] Can group by time periods (month/day/hour)
- [ ] Remember to use GROUP BY with aggregates
- [ ] Can combine JOINs + CASE + DATE functions in one query

Good luck! 🚀