SQL Revision Notes - Quick Reference Guide

📊 DISTINCT - Finding Unique Values

Basic Syntax

```sql
SELECT DISTINCT column_name FROM table_name;
```

Key Concepts

- **Purpose**: Returns only unique/different values (removes duplicates)
- **Use Case**: Data exploration - understand what unique values exist in your dataset

Examples

Single Column:

```sql
-- Find unique pharmaceutical manufacturers
SELECT DISTINCT manufacturer FROM pharmacy_sales;
```

Multiple Columns:

```sql
-- Find unique combinations of user_id and status
SELECT DISTINCT user_id, status FROM trades ORDER BY user_id;
```

⚠️ **Note**: Only write DISTINCT once, not for each column!

COUNT DISTINCT:

```sql
-- Count number of unique users who made trades
SELECT COUNT(DISTINCT user_id) FROM trades;

-- Count distinct products per category (Amazon question)
SELECT category, COUNT(DISTINCT product) AS count
FROM product_spend
GROUP BY category;
```

⚠️ **Important**: DISTINCT goes INSIDE the COUNT() function, not before SELECT!

+−✕÷ SQL Arithmetic Operators

Operator Summary Table

Operator	Description	Example	Result
`+`	Addition	`15 + 5`	20
`-`	Subtraction	`15 - 5`	10
`*`	Multiplication	`15 * 5`	75
`/`	Division	`15 / 5`	3
`%`	Modulus (Remainder)	`14 % 5`	4
`^`	Exponentiation	`15 ^ 2`	225

Real Examples

Addition:

```sql
-- Calculate total compensation
SELECT salary + bonus AS total_compensation FROM employees;
```

Subtraction:

```sql
-- Calculate profit
SELECT revenue - expenses AS profit FROM product_sales;
```

Multiplication:

```sql
-- Calculate revenue
SELECT units_sold * price AS revenue FROM ecomm_orders;
```

Division:

```sql
-- Calculate GDP per capita
SELECT country_gdp / population AS gdp_per_capita FROM econ_stats;
```

Modulus (%):

```sql
```

```
-- Find odd-numbered measurements
SELECT * FROM measurements
WHERE measurement_num % 2 = 1;  -- Remainder of 1 means odd!

-- Find even-numbered measurements
WHERE measurement_num % 2 = 0;  -- Remainder of 0 means even!
```

Order of Operations (PEMDAS)

1. **P**arentheses first
2. **E**xponents (^)
3. **M**ultiplication and **D**ivision (left-to-right)
4. **A**ddition and **S**ubtraction (left-to-right)

Examples:

```sql
SELECT 3 + 7 * 2;          -- Result: 17 (multiplication first)
SELECT (3 + 7) * 2;        -- Result: 20 (parentheses first)
SELECT 10 / 2 + 3 * 4;     -- Result: 17 (10/2=5, 3*4=12, then 5+12)
```

💡 **Pro Tip**: Use parentheses to make your code more readable!

Interview Questions

CVS Pharmacy - Top 3 Most Profitable Medicines:

```sql
-- Hint: Total Profit = Total Sales - Cost of Goods Sold
SELECT drug, (total_sales - cogs) AS total_profit
FROM pharmacy_sales
ORDER BY total_profit DESC
LIMIT 3;
```

JPMorgan Chase - Credit Card Issuance Range:

```sql
```

```
-- Find difference between highest and lowest month
SELECT
  card_name,
  MAX(issued_amount) - MIN(issued_amount) AS difference
FROM monthly_cards_issued
GROUP BY card_name
ORDER BY difference DESC;
```

🔢 SQL Math Functions

ABS() - Absolute Value

```sql
-- Get absolute difference between opening and closing prices
SELECT
  date,
  ticker,
  (close - open) AS difference,
  ABS(close - open) AS abs_difference
FROM stock_prices
WHERE ticker = 'GOOG';
```

Result: Negative differences become positive (e.g., -9.44 → 9.44)

ROUND() - Round Numbers

```sql
-- Round average closing price to 2 decimal places
SELECT
  ticker,
  AVG(close) AS avg_close,
  ROUND(AVG(close), 2) AS rounded_avg_close
FROM stock_prices
GROUP BY ticker;
```

Syntax: `ROUND(number, decimal_places)`

CEIL() and FLOOR() - Round Up/Down

```sql
```

```sql
-- CEIL rounds UP, FLOOR rounds DOWN
SELECT
  date,
  high,
  CEIL(high) AS resistance_level,    -- Rounds up: 123.4 → 124
  low,
  FLOOR(low) AS support_level        -- Rounds down: 123.9 → 123
FROM stock_prices
WHERE ticker = 'META';
```

POWER() - Exponentiation

sql

```sql
-- Calculate squared values
SELECT
  date,
  close,
  ROUND(POWER(close, 2), 2) AS squared_close
FROM stock_prices;

-- Shorthand in PostgreSQL:
SELECT close ^ 2 AS squared_close FROM stock_prices;
```

MOD() or % - Modulus/Remainder

sql

```sql
-- Find stocks with prices divisible by 5
SELECT
  ticker,
  close,
  MOD(close, 5) AS price_remainder_mod,
  close % 5 AS price_remainder_modulo
FROM stock_prices
WHERE ticker = 'GOOG';
```

Interview Question - CVS CEIL Practice

sql

```sql
-- Find per unit cost for Merck drugs, rounded up
SELECT
  drug,
  CEIL(total_sales / units_sold) AS unit_cost
FROM pharmacy_sales
WHERE manufacturer = 'Merck'
ORDER BY unit_cost;
```

➗ SQL Division - The Tricky Parts!

Integer Division Problem

Query	SQL Output	Excel Output
`SELECT 10/4`	2	2.5
`SELECT 10/2`	5	5
`SELECT 10/6`	1	1.67
`SELECT 10.0/4`	2.5	2.5

⚠️ **Problem**: Integer division in SQL **discards the remainder**!

Solution 1: CAST()

sql

```sql
-- Convert to DECIMAL or FLOAT
SELECT
  CAST(10 AS DECIMAL) / 4,        -- Result: 2.5
  10 / CAST(6 AS DECIMAL),        -- Result: 1.67
  CAST(10 AS FLOAT) / 4;          -- Result: 2.5
```

Solution 2: Multiply by 1.0

sql

```sql
-- Simple trick to get decimals
SELECT
  10 / 6,             -- Result: 1 (integer division)
  10 * 1.0 / 6,       -- Result: 1.67 (decimal!)
  10 / (6 * 1.0);     -- Result: 1.67 (decimal!)
```

Solution 3: :: Notation

sql

```
-- Explicit type casting
SELECT
  10::DECIMAL / 4,     -- Result: 2.5
  10::FLOAT / 6,       -- Result: 1.67
  10 / 4::DECIMAL;     -- Result: 2.5
```

Calculating Percentages

Basic Formula: `(part / total) * 100`

Without Rounding:

```sql
SELECT
  sale_id,
  actual_sales,
  target_sales,
  (actual_sales / target_sales) * 100 AS sales_percentage
FROM sales;
```

With Rounding (Recommended):

```sql
SELECT
  sale_id,
  ROUND((actual_sales / target_sales) * 100, 2) AS sales_percentage_rounded
FROM sales;
```

Example Output:

sale_id	actual_sales	target_sales	sales_percentage
1	500.00	1000.00	50.00
2	700.00	900.00	77.78
5	1000.00	1000.00	100.00

💡 **Display Formats**: Both 0.50 and 50.0 are correct for percentages - choose based on context!

Google Interview Question - ROAS

```sql
```

```
-- Calculate Return on Ad Spend (ROAS) = revenue / spend
SELECT
  advertiser_id,
  ROUND(SUM(revenue) / SUM(spend), 2) AS roas
FROM ad_campaigns
GROUP BY advertiser_id
ORDER BY advertiser_id;
```

❓ NULL - Handling Missing Values

What is NULL?

- **NULL** = absence of a value (not empty string, not zero!)
- Represents missing or unknown information
- Common in real-world data: survey responses, incomplete records, pending data

IS NULL and IS NOT NULL

❌ WRONG WAY:

```sql
-- This DOESN'T WORK!
SELECT * FROM goodreads
WHERE book_title = NULL;  -- Returns nothing!
```

✅ CORRECT WAY:

```sql
-- Find records with NULL values
SELECT * FROM goodreads
WHERE book_title IS NULL;

-- Find records WITHOUT NULL values
SELECT * FROM goodreads
WHERE book_title IS NOT NULL;
```

COALESCE() - First Non-NULL Value

Syntax: `COALESCE(value1, value2, value3, ...)`

```sql
```

```sql
-- Replace NULL ratings with 0
SELECT
  book_title,
  COALESCE(book_rating, 0) AS coalesced_rating
FROM goodreads;
```

How it works: Returns the first non-NULL value from the list

Example:

- If `book_rating = 4.5` → Returns `4.5`
- If `book_rating = NULL` → Returns `0`

IFNULL() - Simple NULL Replacement

Syntax: `IFNULL(expression, value_if_null)`

sql

```sql
-- Replace NULL ratings with 0
SELECT
  book_title,
  IFNULL(book_rating, 0) AS rated_books
FROM goodreads;
```

COALESCE() vs IFNULL()

Function	Arguments	Use Case
`COALESCE()`	Multiple (2+)	More flexible, returns first non-NULL
`IFNULL()`	Exactly 2	Simpler, concise for basic cases

Example:

sql

```sql
-- COALESCE with multiple fallbacks
COALESCE(arg1, arg2, arg3)  -- Returns first non-NULL

-- IFNULL with one fallback
IFNULL(arg1, arg2)  -- If arg1 is NULL, return arg2
```

Tesla Interview Question - Unfinished Parts

sql

```
-- Find car parts that started but aren't finished
SELECT * FROM parts_assembly
WHERE finish_date IS NULL;
```

💡 **Fun Fact**: In SQL sorting, NULL is considered the smallest value - NULL rows appear at the top when sorting ascending!

🎯 Interview Tips & Common Patterns

Pattern 1: Profit Calculation

```sql
-- Revenue - Cost = Profit
SELECT product, (revenue - cost) AS profit
FROM sales
ORDER BY profit DESC;
```

Pattern 2: Finding Odd/Even Numbers

```sql
-- Odd numbers (remainder = 1)
WHERE id % 2 = 1

-- Even numbers (remainder = 0)
WHERE id % 2 = 0
```

Pattern 3: Percentage Calculations

```sql
-- Always multiply by 1.0 to avoid integer division!
SELECT ROUND((part * 1.0 / total) * 100, 2) AS percentage
FROM table_name;
```

Pattern 4: Handling NULL in Calculations

```sql
-- Replace NULL before calculating
SELECT
  product,
  COALESCE(sales, 0) + COALESCE(bonus, 0) AS total
FROM revenue;
```

Pattern 5: Absolute Difference

```sql
-- Use ABS for absolute values
SELECT ABS(value1 - value2) AS difference
FROM comparisons;
```

📝 Quick Cheat Sheet

```sql
-- DISTINCT
SELECT DISTINCT column FROM table;
SELECT COUNT(DISTINCT column) FROM table;

-- ARITHMETIC
column1 + column2    -- Addition
column1 - column2    -- Subtraction
column1 * column2    -- Multiplication
column1 / column2    -- Division (watch for integers!)
column1 % column2    -- Modulus/Remainder
column1 ^ 2          -- Power/Exponentiation

-- MATH FUNCTIONS
ABS(number)              -- Absolute value
ROUND(number, decimals)    -- Round to decimals
CEIL(number)             -- Round up
FLOOR(number)            -- Round down
POWER(number, power)       -- Exponentiation
MOD(number, divisor)       -- Modulus

-- DIVISION FIXES
CAST(column AS DECIMAL)    -- Convert type
column * 1.0               -- Force decimal
column::DECIMAL            -- PostgreSQL casting

-- NULL HANDLING
WHERE column IS NULL         -- Find NULLs
WHERE column IS NOT NULL     -- Exclude NULLs
COALESCE(col1, col2, 0)      -- First non-NULL
IFNULL(column, default)      -- Replace NULL
```

⚠️ Common Mistakes to Avoid

1. **Don't use `= NULL`** → Use `IS NULL`
2. **Don't forget integer division** → Multiply by 1.0 or CAST
3. **Don't put DISTINCT outside COUNT()** → `COUNT(DISTINCT column)`
4. **Don't forget order of operations** → Use parentheses
5. **Don't assume NULL = 0 or empty string** → They're different!

🎓 Practice Problem Categories

✅ **DISTINCT**: Count unique products, users, categories
✅ **Arithmetic**: Calculate profit, revenue, percentages
✅ **Math Functions**: Round prices, find absolute differences
✅ **Division**: ROAS, percentages, unit costs
✅ **NULL Handling**: Find incomplete records, clean data

Remember: Practice is key! Work through interview questions from CVS, Google, JPMorgan, and Tesla to solidify these concepts.