SQL Interview Cheat Sheet

DataLemur.com

SAMPLE DATA

***customers* Table:**

id	name	age	city	country	has_subscription
1	Adam	58	New York	USA	TRUE
2	Bella	NULL	Tijuana	Mexico	FALSE
3	Chetan	36	New Delhi	India	TRUE

***orders* Table:**

order_id	cus_id	date	cost	discount	status
101	1	2023-04-05	300.00	0.00	Delivered
102	2	2023-10-02	400.00	0.00	Shipped
103	2	2024-11-19	100.00	25.35	TBD
999	NULL	2027-06-16	1200	0.00	TBD

QUERYING TABLES WITH SELECT

Fetch all columns from the customers table:

```
SELECT *
FROM customer;
```

Fetch name and age columns for all customers:

```
SELECT name, age
FROM customers;
```

Sort Output Using ORDER BY

Sort customers by age in the default ASCending order:

```
SELECT *
FROM customers
ORDER BY age ASC;
```

Sort customers by age in DESCending order (high to low):

```
SELECT *
FROM customers
ORDER BY age DESC;
```

Find the discount percentage for all orders:

```
SELECT discount/cost * 100
FROM orders;
```

Round the discount percentage to 2 decimal places:

```
SELECT ROUND(discount/cost * 100, 2)
FROM orders;
```

Aliases

AS is used to rename columns:

```
SELECT cost * 0.04 AS sales_tax
FROM orders;
```

AS is also used to rename tables:

```
SELECT cus.name, cus.age
FROM customers AS cus;
```

FILTERING OUTPUT WITH WHERE

Comparison Operators

Fetch customers who are over the age of 35:

```
SELECT *
FROM customers
WHERE age > 35;
```

Fetch customers that live in either USA OR Canada AND also have a subscription:

```
SELECT *
FROM customers
WHERE (country = 'USA' OR country = 'Canada')
AND has_subscription = TRUE;
```

BETWEEN and IN

Fetch the status of orders that have a cost between 100-200:

```
SELECT status
FROM orders
WHERE TotalCost BETWEEN 100 AND 200;
```

Fetch customers that live in North America:

```
SELECT name
FROM customers
WHERE country IN ('USA', 'Canada', 'Mexico');
```

Filter Text With LIKE

Fetch customers who live in a city that starts with "New":

```
SELECT *
FROM customers
WHERE city LIKE 'New%';
```

Fetch customers whose city ends in 'e' or the 2nd letter is 'r' or the name contains the letter 'x' anywhere:

```
SELECT *
FROM customers
WHERE city LIKE '%e'
  OR city LIKE '_r%'
  OR city LIKE '%x%';
```

NOT and NULLs

Fetch customers that are not missing a value for age:

```
SELECT name
FROM customers
WHERE age IS NOT NULL;
```

Fetch customers that aren't USA minors (no diddy):

```
SELECT name
FROM customers
WHERE NOT (age < 18 AND country = 'USA')
```

COMBINING MULTIPLE TABLES WITH JOINS

INNER JOIN

JOIN (or explicitly INNER JOIN) returns rows that have matching values in both tables.

```
SELECT orders.order_id, orders.cus_id,
customers.id, customers.name
FROM orders
INNER JOIN customers
ON orders.cus_id = customers.id;
```

order_id	cus_id	id	name
101	1	1	Adam
102	2	2	Bella
103	2	2	Bella

LEFT JOIN

LEFT JOIN returns all rows from the left table with corresponding rows from the right table. If no matched row, NULLs are returned as values for the 2nd table.

```
SELECT customers.name, orders.date, orders.cost
FROM customers
LEFT JOIN orders
ON customers.id = orders.cus_id;
```

name	date	cost
Adam	2023-04-05	300.00
Bella	2023-10-02	400.00
Bella	2024-11-19	100.00
Chetan	NULL	NULL

CROSS JOIN

CROSS JOIN returns all possible combinations or rows from both tables. It doesn't have a join condition!

```
SELECT orders.order_id, customers.name
FROM orders
CROSS JOIN customers;
```

order_id	name
101	Adam
101	Bella
101	Chetan
102	Adam
102	Bella
102	Chetan

FULL JOIN

FULL JOIN (or explicitly FULL OUTER JOIN) returns all rows from both tables - if there's no matching row in the second table, NULLs are returned.

```
SELECT orders.order_id, orders.cost,
customers.id, customers.name
FROM orders
FULL OUTER JOIN customers
ON orders.cus_id = customers.id;
```

order_id	cost	id	name
101	300	1	Adam
102	400	2	Bella
103	100	2	Bella
999	1200	NULL	NULL
NULL	NULL	3	Chetan

Practice real **Meta** Full Join SQL Interview Question:
datalemur.com/questions/updated-status

RIGHT JOIN

RIGHT JOIN returns all rows from the right table with corresponding rows from the left table.
If there's no matching row, NULLs are returned as values from the left table.

```
SELECT orders.order_id, customers.name
FROM orders
RIGHT JOIN customers
ON orders.cus_id = customers.id;
```

SELF JOIN

SELF JOIN is used to join a table with itself to compare rows within the same table. It's typically used with an alias to differentiate the table instances.

```
SELECT A.cus_id, A.order_id AS ord_1, A.date AS
date_1, B.order_id AS ord_2, B.date AS date_2
FROM orders A
JOIN orders B ON A.cus_id = B.cus_id
AND A.order_id != B.order_id;
```

cus_id	ord_1	date_1	ord_2	date_2
2	102	2023-10-02	103	2024-11-19
2	103	2024-11-19	102	2023-10-02

Practice real **Amazon** Self-Join SQL Interview Question:
datalemur.com/questions/amazon-shopping-spree

AGGREGATION AND GROUPING

orders		
order_id	cus_id	cost
105	4	300.00
106	2	150.00
107	1	200.00
108	2	150.00
109	1	75.50
110	4	100.00
111	1	100.75

GROUP BY groups together rows with the same value in specified columns, and then computes summaries (aggregates) for each group of values.

```
SELECT cus_id,
SUM(cost) as sum_cost,
COUNT(order_id) as count_id,
MAX(cost) as max_cost,
ROUND(AVG(cost),2) as avg_cost
FROM orders
GROUP BY cus_id
ORDER BY cus_id;
```

orders				
cus_id	sum_cost	count_id	max_cost	avg_cost
1	376.25	3	200.00	125.42
2	300.00	2	150.00	150.00
4	400.00	2	300.00	200.00

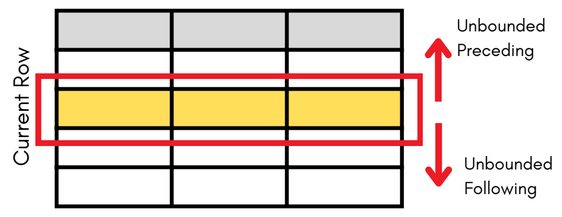
WINDOW FUNCTIONS

AGGREGATE
- AVG()
- MAX()
- MIN()
- SUM()
- COUNT()

RANKING
- ROW_NUMBER()
- RANK()
- DESNE_RANK()
- PERCENT_RANK()
- NTILE()

VALUE
- LAG()
- LEAD()
- FIRST_VALUE()
- LAST_VALUE()
- NTH_VALUE()

Compute their result based on a sliding window frame, a set of rows that somehow relate to the current row.



PARTITION BY

Divides rows into multiple groups, called **partitions,** to which the window function is applied.

```
SELECT order_id, cus_id, cost,
SUM(cost) OVER (PARTITION BY cus_id) AS sum_cost
FROM orders;
```

order_id	cus_id	cost	sum_cost
107	1	200.00	376.25
109	1	75.50	376.25
111	1	100.75	376.25
106	2	150.00	300.00
108	2	150.00	300.00
105	4	300.00	400.00
110	4	100.00	400.00

Default Partition: with no PARTITION BY clause, the entire result set is the partition.

RANK Window Function Example

The RANK() function is used to assign ranks to rows based on values in the specified column.

```
SELECT order_id, cost,
RANK() OVER (
 ORDER BY cost DESC
)
AS order_rank
FROM orders;
```

order_id	cost	order_rank
105	300.00	1
107	200.00	2
106	150.00	3
108	150.00	3

Practice real **Google** RANK SQL Interview Question:
datalemur.com/questions/odd-even-measurements

ORDER BY

Specifies the order of rows in each partition to which the window function is applied.

```
SELECT order_id, cus_id, cost,
SUM(cost) OVER (PARTITION BY cus_Id ORDER BY cost ASC)
AS sum_cost
FROM orders;
```

order_id	cus_id	cost	sum_cost
109	1	75.50	376.25
111	1	100.75	376.25
107	1	200.00	376.25
106	2	150.00	300.00
108	2	150.00	300.00
110	4	100.00	400.00
105	4	300.00	400.00

Default ORDER BY: with no ORDER BY clause, the order of rows within each partition is arbitrary

LAG Year-over-Year Example

The LAG() function is used to access data from a previous row in the same result set without needing a self-join. Often used for Year-over-Year or Month-over-Month calculations.

```
SELECT YEAR(date) AS year,
SUM(cost) AS cur_sales,
LAG(SUM(cost)) OVER (
  ORDER BY YEAR(date)
)
AS last_year_sales
FROM orders
GROUP BY YEAR(date)
ORDER BY YEAR(date);
```

year	cur_sales	last_year_sales
2023	700.00	NULL
2024	300.00	700.00
2025	450.00	300.00

SUBQUERIES

A subquery is a query that is nested inside another query, or inside another subquery.

Single Value Subqueries

The simplest subquery returns exactly one column and exactly one row. It can be used with comparison operators (=, <, >, <=, or >=)

This query finds all orders that cost more than the average order.

```
SELECT order_id
FROM orders
WHERE cost > (
  SELECT AVG(cost)
  FROM orders
);
```

Multiple Value Subqueries

A subquery can also return multiple columns or multiple rows. Such subqueries can be used with operators:
IN, EXISTS, ANY, or ALL

This query finds all orders made by USA customers:

```
SELECT order_id
FROM orders
WHERE cus_id = ANY (
  SELECT id as cus_id
  FROM customers
  WHERE country = USA);
```

CASE STATEMENTS

CASE goes through a list of conditions and returns a value when the first condition is met.
If no conditions are true it returns the value in the **ELSE** clause.

```
SELECT order_id, cus_id, cost,
CASE
 WHEN cost > 175 THEN 'luxury'
 WHEN cost > 100 THEN 'mid-tier'
 ELSE 'budget' END AS product_type
FROM orders;
```

order_id	cus_id	cost	product_type
107	1	200.00	luxury
109	1	75.50	budget
111	1	100.75	mid-tier
106	2	150.00	mid-tier
108	2	150.00	mid-tier
105	4	300.00	luxury

CTEs

Common Table Expressions (CTEs) are temporary result sets that you can reference within a query. CTEs improve query readability and are often used to simplify complex queries by breaking them into smaller, manageable parts.

This query calculates the total sales per customer and then filters customers with total sales above 350.

```
WITH sum_sales AS (
  SELECT cus_id, SUM(cost) AS tot_sales
  FROM orders
  GROUP BY cus_id)
SELECT cus_id, tot_sales
FROM sum_sales
WHERE tot_sales > 350;
```

cus_id	tot_sales
4	400.00

SET OPERATIONS

Set operations are used to combine the results of two or more queries/tables into a single result.
UNION combines the results and removes duplicates.
UNION ALL doesn't remove duplicate rows

actors		
id	name	country
1	Ryan Gosling	Canada
2	Drake	Canada
3	Emma Stone	USA

singers		
id	name	country
1	Drake	Canada
2	Badshah	India
3	Justin Bieber	Canada

```
SELECT name
FROM actors
WHERE country = 'Canada'
UNION ALL
SELECT name
FROM singers
WHERE country = 'Canada';
```

id	name	country
1	Ryan Gosling	Canada
2	Drake	Canada
1	Drake	Canada
3	Justin Bieber	Canada

Practice real **Amazon** UNION SQL Interview Question:
datalemur.com/questions/prime-warehouse-storage

OTHER SQL COMMANDS

LENGTH(): Returns the length of the provided string.
CAST(): Converts an expression into the specified data type.
NOW(): Returns the current date and time.
CEILING(): Rounds up to the nearest integer.
FLOOR(): Rounds down to the nearest integer.
TRIM(): Removes spaces from both or one side of a string.
CONCAT(): Combines multiple strings.
COALESCE(): Returns the first non-NULL value.

Practice 200+ FAANG SQL Interview Questions for FREE: DataLemur.com/questions
Learn SQL with 30 Interactive SQL Lessons for FREE: DataLemur.com/sql-tutorial