SQL Interview Cheat Sheet

Essential SQL Commands & Concepts

Interview Master

Sample Data: TokTuk Platform

Tables used: `Users`, `Videos`, `Interactions`.

Users Table:

user_id	username	email	join_date
1	alice	alice@toktuk.com	2023-01-01
2	bob	bob@toktuk.com	2023-01-02
3	charlie	charlie@toktuk.com	2023-01-03
4	david	david@toktuk.com	2023-01-04

Videos Table:

video_id	user_id	title	upload_date	views
1	1	Funny Cat	2023-02-01	1000
2	2	Dance Challenge	2023-02-02	2000
3	1	Cooking Tips	2023-02-03	500
4	3	Travel Vlog	2023-02-04	1500

Interactions Table:

interaction_id	user_id	video_id	interaction_type	timestamp	comment_text
1	2	1	like	2023-02-01 10:00:00	NULL
2	3	1	comment	2023-02-01 10:05:00	So funny!
3	1	2	like	2023-02-02 11:00:00	NULL
4	5	2	like	2023-02-02 11:05:00	NULL

Querying Tables with SELECT

Interview Master

Fetch all columns from the Users table:

```
SELECT * FROM Users;
```

Fetch username and email for all users:

```
SELECT username, email FROM Users;
```

Calculate estimated ad revenue (e.g., $0.01 per view):

```
SELECT title, views, views * 0.01 AS estimated_revenue FROM Videos;
```

Round the estimated revenue to 2 decimal places:

```
SELECT title, ROUND(views * 0.01, 2) AS rounded_revenue FROM Videos;
```

Sort Output Using ORDER BY:

```
-- Sort users by join_date in ASCending order (default): SELECT * FROM Users ORDER BY join_date; -- Sort videos by views in DESCending order (high to low): SELECT * FROM Videos ORDER BY views DESC;
```

Aliases:

```
-- AS is used to rename columns: SELECT username AS toktuk_user, email AS user_contact FROM Users; -- It is also used to rename tables (useful in JOINs): SELECT u.username, v.title FROM Users AS u, Videos AS v WHERE u.user_id = v.user_id LIMIT 1; -- Example join condition
```

ASC: Used for ascending order (default).
DESC: Used for descending order.
LIMIT: Restricts the number of rows returned.

Subqueries (Nested Queries)

A query nested inside another SQL query.

Scalar Subquery (returns a single value):

Find videos with views above the average number of views.

```
SELECT title, views FROM Videos WHERE views > (SELECT AVG(views) FROM Videos);
```

title	views
Dance Challenge	2000
Travel Vlog	1500

Multi-row/Multi-column Subqueries (used with IN, EXISTS, JOINs):

Find videos uploaded by users who joined before '2023-01-03'.

```
SELECT title, upload_date FROM Videos WHERE user_id IN ( SELECT user_id FROM Users WHERE join_date < '2023-01-03' );
```

title	upload_date
Funny Cat	2023-02-01
Cooking Tips	2023-02-03
Dance Challenge	2023-02-02

Subqueries can also be used in `SELECT` (correlated), `FROM` (derived table), and `JOIN` clauses.

Filtering Output with WHERE

Comparison Operators:

Find videos with more than 1000 views:

```
SELECT * FROM Videos WHERE views > 1000;
```

Fetch users who joined on or after '2023-01-03':

```
SELECT * FROM Users WHERE join_date >= '2023-01-03';
```

Fetch interactions that are 'comments':

```
SELECT * FROM Interactions WHERE interaction_type = 'comment';
```

BETWEEN and IN:

```
-- Fetch videos with views between 500 and 1000 (inclusive): SELECT title, views FROM Videos WHERE views BETWEEN 500 AND 1000; -- Fetch users with specific user_ids: SELECT username, email FROM Users WHERE user_id IN (1, 3, 5);
```

Filter Text with LIKE:

Fetch users whose username starts with "a":

```
SELECT * FROM Users WHERE username LIKE 'ak';
```

Fetch videos with "Cat" or "Dog" in the title:

```
SELECT * FROM Videos WHERE title LIKE '%Cat%' OR title LIKE '%Dog%';
```

NOT and NULL:

Fetch interactions that have a comment (comment_text is not NULL):

```
SELECT * FROM Interactions WHERE comment_text IS NOT NULL;
```

Fetch users whose user_id is NOT 1:

```
SELECT * FROM Users WHERE NOT user_id = 1;
```

LIKE Wildcards:
`%` - Represents zero or more characters
`_` - Represents exactly one character

Set Operations

Combine results of two or more SELECT statements. Queries must have the same number of columns and compatible data types.

UNION / UNION ALL:

UNION combines results and removes duplicates. UNION ALL includes all duplicates.

```
-- Get a list of user IDs who either uploaded a video OR made a comment SELECT user_id FROM Videos UNION SELECT user_id FROM Interactions WHERE comment_text IS NOT NULL;
```

INTERSECT:

Returns rows that are common to both queries.

```
-- User IDs that have uploaded videos AND also made comments SELECT user_id FROM Videos INTERSECT SELECT user_id FROM Interactions WHERE comment_text IS NOT NULL;
```

EXCEPT (MINUS in Oracle):

Returns rows from the first query that are not in the second query.

```
-- User IDs that have uploaded videos BUT have NOT made any comments SELECT user_id FROM Videos EXCEPT SELECT user_id FROM Interactions WHERE comment_text IS NOT NULL;
```

Common SQL Concepts: Q&A

Interview Master

Q: What's the difference between WHERE and HAVING?
A: WHERE filters rows before any groupings are made. HAVING filters groups after the GROUP BY clause has been applied.

Q: What's the difference between Primary Key and Foreign Key?
A: A Primary Key uniquely identifies each record in a table and cannot contain NULL values. A Foreign Key is a field (or collection of fields) in one table that refers to the Primary Key in another table, creating a link between them. Foreign Keys can be NULL if the relationship is optional.

Q: What is the difference between DELETE and TRUNCATE?
A: DELETE is a DML command that removes rows one by one and logs these deletions, so it can be rolled back. TRUNCATE is a DDL command that removes all rows from a table by deallocating the data pages; it's much faster and typically cannot be rolled back easily.

Q: Can a Foreign Key be NULL?
A: Yes, a Foreign Key can be NULL. This is often used to represent an optional relationship or a record that doesn't have a corresponding entry in the referenced table.

Q: What's the difference between INNER JOIN and LEFT JOIN?
A: INNER JOIN returns only the rows where there is a match in both tables. LEFT JOIN (or LEFT OUTER JOIN) returns all rows from the left table, and the matched rows from the right table; if there is no match, NULL is returned for columns from the right table.

Q: What is an index and why is it useful?
A: An index is a special lookup table that the database search engine can use to speed up data retrieval. It helps queries find data faster by creating pointers to data in a table, similar to a book's index. However, indexes can slow down data modification operations (INSERT, UPDATE, DELETE).

Q: What's the difference between UNION and UNION ALL?
A: UNION combines the result sets of two or more SELECT statements and removes duplicate rows. UNION ALL also combines result sets but includes all rows, including duplicates. UNION ALL is generally faster if duplicate removal is not necessary.

Q: What is Normalization?
A: Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It typically involves dividing larger tables into smaller, more manageable ones and defining relationships between them. Common forms include 1NF (First Normal Form), 2NF (Second Normal Form), and 3NF (Third Normal Form).

Q: What is a View in SQL?
A: A view is a virtual table based on the result-set of a stored SQL statement. A view contains rows and columns, just like a real table, with fields from one or more real tables. Views can simplify complex queries, restrict data access, or present data in a different format without storing data separately.

Q: What is the difference between ROW_NUMBER(), RANK(), and DENSE_RANK()?
A: All are window functions. ROW_NUMBER() assigns a unique sequential integer to each row within its partition. RANK() assigns a rank to each row within its partition, with gaps in the ranking sequence if there are ties (e.g., 1, 1, 3). DENSE_RANK() also assigns a rank, but without gaps in the ranking sequence for ties (e.g., 1, 1, 2).

Combining Multiple Tables with JOINS

INNER JOIN:

Returns rows that have a matching value in both tables. E.g., users and their videos.

```
SELECT u.username, v.title, v.views FROM Users u INNER JOIN Videos v ON u.user_id = v.user_id;
```

username	title	views
alice	Funny Cat	1000
bob	Dance Challenge	2000
alice	Cooking Tips	500
charlie	Travel Vlog	1500

LEFT JOIN:

Returns all rows from the left table (Users) and matched rows from the right table (Videos). E.g., all users and any videos they have (or NULL if none).

```
SELECT u.username, v.title FROM Users u LEFT JOIN Videos v ON u.user_id = v.user_id;
```

username	title
alice	Funny Cat
alice	Cooking Tips
bob	Dance Challenge
charlie	Travel Vlog
david	NULL

RIGHT JOIN:

Returns all rows from the right table (Videos) and matched rows from the left (Users). (Less common than LEFT JOIN).

```
SELECT v.title, u.username FROM Users u RIGHT JOIN Videos v ON u.user_id = v.user_id;
```

FULL OUTER JOIN:

Returns all records when there is a match in either Users or Videos table.

```
SELECT u.username, v.title FROM Users u FULL OUTER JOIN Videos v ON u.user_id = v.user_id;
```

CROSS JOIN:

Cartesian product. Every user combined with every video. (Use with caution!)

```
-- First 2 users X first 2 videos SELECT U.username, V.title FROM ( SELECT user_id, username FROM Users ORDER BY user_id LIMIT 2 ) U CROSS JOIN ( SELECT video_id, title FROM Videos ORDER BY video_id LIMIT 2 ) V;
```

SELF JOIN:

Join a table to itself. E.g., find pairs of users who joined on the same date.

```
SELECT U1.username AS User1, U2.username AS User2, U1.join_date FROM Users U1 JOIN Users U2 ON U1.join_date = u2.join_date AND U1.user_id < U2.user_id;
```

CASE Statements

Allows for conditional logic within SQL queries.

```
-- Categorize videos based on views SELECT title, views, CASE WHEN views >= 1500 THEN 'Viral Hit' WHEN views >= 500 THEN 'Popular' ELSE 'Regular' END AS video_category FROM Videos;
```

title	views	video_category
Funny Cat	1000	Popular
Dance Challenge	2000	Viral Hit
Cooking Tips	500	Regular
Travel Vlog	1500	Viral Hit

Aggregation and Grouping

`GROUP BY`: Groups rows with the same values into a summary row.

```
-- Count videos and sum views per user SELECT u.username, COUNT(v.video_id) as count_videos, SUM(v.views) as total_views, ROUND(AVG(v.views),0) as avg_views_per_video, MAX(v.views) as max_views_single_video FROM Users u LEFT JOIN Videos v ON u.user_id = v.user_id GROUP BY u.user_id, u.username;
```

username	count_videos	total_views	avg_views_per_video	max_views_single_video
alice	2	1500	750	1000
bob	1	2000	2000	2000
charlie	1	1500	1500	1500
david	0	NULL	NULL	NULL

Common Aggregate Functions:

`COUNT(*)`	`MIN(col)`
`COUNT(col)`	`MAX(col)`
`SUM(col)`	`STRING_AGG()`
`AVG(col)`	`GROUP_CONCAT()`

HAVING Clause:

HAVING filters groups (after GROUP BY).

```
-- Users with >0 videos and total views > 1000 SELECT u.username, COUNT(v.video_id) as num_videos, SUM(v.views) as total_views FROM Users u JOIN Videos v ON u.user_id = v.user_id GROUP BY u.user_id, u.username HAVING COUNT(v.video_id) > 0 AND SUM(v.views) > 1000;
```

CTEs (Common Table Expressions)

Interview Master

Define temporary, named result sets. Improves readability.

Basic CTE Example:

```
-- Users who uploaded more than 0 videos WITH UserVideoCounts AS ( SELECT user_id, COUNT(video_id) as num_videos FROM Videos GROUP BY user_id ) SELECT u.username, uvc.num_videos FROM Users u JOIN UserVideoCounts uvc ON u.user_id = uvc.user_id WHERE uvc.num_videos > 0;
```

username	num_videos
alice	2
bob	1
charlie	1

Multiple CTEs:

```
WITH EarlyUsers AS ( SELECT user_id, username FROM Users WHERE join_date < '2023-01-03' ), PopularVideos AS ( SELECT video_id, user_id, title, views FROM Videos WHERE views >= 1500 ) -- Popular videos by early users SELECT eu.username, pv.title, pv.views FROM EarlyUsers eu JOIN PopularVideos pv ON eu.user_id = pv.user_id;
```

username	title	views
bob	Dance Challenge	2000

Window Functions

Compute values across a set of table rows related to the current row.

AGGREGATE	RANKING	VALUE
`SUM() OVER()`	`ROW_NUMBER()`	`LAG() OVER()`
`COUNT() OVER()`	`RANK()`	`LEAD() OVER()`
`AVG() OVER()`	`DENSE_RANK()`	`FIRST_VALUE()`
`ROWS BETWEEN`		

PARTITION BY:

Divides rows into partitions; window function applied to each independently.

```
-- Rank videos by views within each user SELECT v.title, u.username, v.views, RANK() OVER (PARTITION BY u.user_id ORDER BY v.views DESC) as rank_within_user FROM Videos v JOIN Users u ON v.user_id = u.user_id;
```

title	username	views	rank_within_user
Funny Cat	alice	1000	1
Cooking Tips	alice	500	2
Dance Challenge	bob	2000	1
Travel Vlog	charlie	1500	1

ORDER BY (in Window Functions):

Specifies order of rows within each partition for the window function.

```
-- Overall rank of videos by views SELECT title, views, ROW_NUMBER() OVER (ORDER BY views DESC) as overall_rank FROM Videos;
```

title	views	overall_rank
Dance Challenge	2000	1
Travel Vlog	1500	2
Funny Cat	1000	3
Cooking Tips	500	4

LAG/LEAD Example:

Access data from a previous (LAG) or next (LEAD) row.

```
-- For each user, show current video views and views of their previously uploaded video SELECT u.username, v.title, v.upload_date, v.views, LAG(v.views, 1, 0) OVER (PARTITION BY v.user_id ORDER BY v.upload_date) as prev_video_views FROM Videos v JOIN Users u ON v.user_id = u.user_id;
```

username	title	upload_date	views	prev_video_views
alice	Funny Cat	2023-02-01	1000	0
alice	Cooking Tips	2023-02-03	500	1000
bob	Dance Challenge	2023-02-02	2000	0
charlie	Travel Vlog	2023-02-04	1500	0

Other Useful SQL Functions

Data Type Conversion & Math:

```
-- CAST: Convert data type SELECT CAST(views AS REAL) / 2 FROM Videos LIMIT 1; -- ROUND: Rounds number SELECT ROUND(views / 3.0, 2) FROM Videos LIMIT 1;
```

```
-- COALESCE: First non-NULL SELECT COALESCE(comment_text, 'N/A') FROM Interactions LIMIT 3; -- CONCAT (|| in SQLite/Postgres) SELECT username || ' (' || email || ')' FROM Users LIMIT 1;
```

Date & Time Functions (Syntax varies):

```
-- EXTRACT part from date SELECT EXTRACT(YEAR FROM DATE(join_date)) FROM Users LIMIT 1; -- Current date/time SELECT CURRENT_DATE, CURRENT_TIMESTAMP; -- Add/subtract interval (example for SQLite/Postgres) SELECT DATE(join_date, '+7 days') FROM Users LIMIT 1;
```

String Functions:

```
SELECT UPPER(username), LOWER(title) FROM Users u, Videos v WHERE u.user_id=v.user_id LIMIT 1; SELECT LENGTH(title) FROM Videos LIMIT 1; SELECT SUBSTR(email, 1, 5) FROM Users LIMIT 1; SELECT REPLACE(interaction_type, 'like', 'heart') FROM Interactions LIMIT 1;
```

Pro Tip: Date/time and some string functions (concatenation) vary significantly across SQL databases (MySQL, PostgreSQL, SQL Server, Oracle). Always check specific RDBMS docs!

Interview Master