SQL Queries for Data Science: Theory & Practice

1. Data Extraction and Filtering

Concept:

Filtering and selecting relevant data using `SELECT`, `WHERE`, `LIMIT`, and `ORDER BY`.

Query Example:

```
SELECT player_id, last_deposit, total_withdraw
FROM wallet
WHERE total_withdraw > 1000
ORDER BY last_deposit DESC
LIMIT 10;
```

Explanation:

- **`SELECT`** retrieves specific columns (`player_id`, `last_deposit`, `total_withdraw`).
- **`WHERE`** filters players who withdrew more than 1000.
- **`ORDER BY`** sorts by last deposit in descending order.
- **`LIMIT`** restricts results to the top 10 players.

2. Aggregation and Summary Statistics

Concept:

Using `GROUP BY`, `COUNT()`, `AVG()`, `SUM()`, `MAX()`, `MIN()` to analyze trends.

Query Example:

```
SELECT currency, COUNT(player_id) AS total_players,
      AVG(total_withdraw) AS avg_withdrawal,
      MAX(last_deposit) AS last_max_deposit
FROM wallet
GROUP BY currency;
```

Explanation:

- **`GROUP BY`** groups data by `currency`.
- **`COUNT(player_id)`** counts the number of players in each currency group.
- **`AVG(total_withdraw)`** calculates the average withdrawal.
- **`MAX(last_deposit)`** finds the most recent deposit date.

3. Joins for Data Integration

Concept:

Combining multiple tables using `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`.

Query Example:

```
SELECT p.player_id, p.name, w.total_withdraw, w.last_deposit
FROM players p
INNER JOIN wallet w ON p.player_id = w.wallet_account_id
WHERE w.total_withdraw > 500;
```

Explanation:

- **`INNER JOIN`** merges data where `player_id` matches in both `players` and `wallet`.
- **`WHERE`** filters players who withdrew more than 500.
- Used to analyze withdrawal behavior of different players.

4. Window Functions for Advanced Analysis

Concept:

Performing calculations across partitions of data using `RANK()`, `ROW_NUMBER()`, `LAG()`, `LEAD()`.

Query Example:

```
SELECT player_id, total_withdraw,
      RANK() OVER (ORDER BY total_withdraw DESC) AS withdrawal_rank
FROM wallet;
```

Explanation:

- **`RANK()`** assigns a ranking to players based on total withdrawal.
- **`OVER(ORDER BY total_withdraw DESC)`** ranks the highest withdrawal first.

5. Case When for Conditional Logic

Concept:

Applying conditional logic in queries.

Query Example:

```
SELECT player_id, total_withdraw,
      CASE
         WHEN total_withdraw >= 1000 THEN 'High Roller'
         WHEN total_withdraw BETWEEN 500 AND 999 THEN 'Regular'
         ELSE 'Casual'
      END AS player_category
FROM wallet;
```

Explanation:

- **`CASE`** classifies players based on their withdrawal amount.
- **Output**: Players are categorized into *High Roller*, *Regular*, and *Casual*.

6. CTE (Common Table Expressions) for Readability

Concept:

Using `WITH` statements to break down complex queries.

Query Example:

```
WITH top_players AS (
    SELECT player_id, SUM(total_withdraw) AS total_withdraw
    FROM wallet
    GROUP BY player_id
    HAVING SUM(total_withdraw) > 2000
)
SELECT p.name, tp.total_withdraw
FROM top_players tp
JOIN players p ON tp.player_id = p.player_id;
```

Explanation:

- **CTE (`WITH top_players`)** simplifies complex queries.
- **Finds players with withdrawals > 2000**.
- **Joins with `players` table** to get names.

7. Subqueries for Complex Filtering

Concept:

Using subqueries to filter data based on another query.

Query Example:

```
SELECT player_id, total_withdraw
FROM wallet
WHERE total_withdraw > (
    SELECT AVG(total_withdraw) FROM wallet
);
```

Explanation:

- **Subquery (`SELECT AVG(total_withdraw)`)** calculates the average withdrawal.
- **Main query** returns players with above-average withdrawals.

8. Handling Missing Data (NULL Handling)

Concept:

Using `COALESCE()` and `IS NULL` to handle missing values.

Query Example:

```
SELECT player_id, COALESCE(last_deposit, 'No Deposit') AS last_deposit_status
FROM wallet;
```

Explanation:

- **`COALESCE()`** replaces NULL values with `"No Deposit"`.

9. Time-Based Analysis

Concept:

Using `DATE()`, `EXTRACT()`, and `INTERVAL` for time-series analysis.

Query Example:

```
SELECT COUNT(player_id) AS new_players
FROM players
WHERE created_at >= CURRENT_DATE - INTERVAL '30 days';
```

Explanation:

- **Finds players who joined in the last 30 days**.

10. Performance Optimization with Indexing

Concept:

Using `INDEX` to speed up queries.

Query:

```
CREATE INDEX idx_player_id ON wallet(player_id);
```

Explanation:

- **Speeds up searches on `player_id` in `wallet` table.**

Conclusion

SQL is essential for **data extraction, analysis, and performance optimization** in Data Science. Mastering these queries will help you **efficiently handle large datasets** and **gain valuable insights**.

Let me know if you want additional topics like **recursive queries, JSON handling, or NoSQL integration with SQL!** 🚀

Get the SQL project + Revision Sheet: Click here

Happy Learning!