05. SQL Basics for Blockchain Analytics

To unlock insights from onchain data, you need one essential tool: **SQL**.

It's the language that lets you ask questions like:

- "How many users swapped on Uniswap last week?"
- "Which wallets minted the most NFTs yesterday?"
- "What's the average gas fee per transaction over time?"
- "How much revenue did this protocol generate in the last 30 days?"

SQL (Structured Query Language) is the backbone of every great onchain dashboard—and you don't need to be a software engineer to learn it.

This guide will help you speak fluent SQL in the blockchain context.

SQL on Dune: A Quick Refresher

Dune uses **PostgreSQL-style syntax** with some extensions for time and math functions.

Queries are typically structured like this:

```
SELECT column_1, column_2
FROM table_name
WHERE conditions
GROUP BY column_1
ORDER BY column_1
```

Let's look at each part in the context of onchain data.

SELECT: Choose What You Want

Use `SELECT` to define the columns you want to display.

Example:

```
SELECT
  block_time,
  from_address,
  to_address,
  value
FROM ethereum.transactions
```

You can also use functions inside `SELECT`, such as:

- `COUNT(*)` – total rows
- `SUM(value)` – total value transferred
- `AVG(gas_price)` – average gas price

WHERE: Filter the Data

You don't want all of history—just what matters.

```
WHERE block_time > now() - interval '7 days'
```

Other common conditions:

```
WHERE value > 0
WHERE from = LOWER('0xabc123...')
WHERE contract_address = '0xUniswapV3Pool'
```

✅ Use `LOWER()` to normalize addresses when joining or filtering.

GROUP BY: Aggregate the Results

Use `GROUP BY` when you want to summarize data.

Example: count swaps per day:

```
SELECT
  date_trunc('day', block_time) AS day,
  COUNT(*) AS swap_count
FROM uniswap_v3.uniswap_v3_swaps
WHERE block_time > now() - interval '30 days'
GROUP BY 1
ORDER BY 1
```

💡 `date_trunc()` is your best friend for time series.

JOIN: Combine Tables

Join lets you merge data from different tables—for example, adding USD prices:

```
SELECT
  t.block_time,
  t.token_address,
  t.amount / 1e18 AS token_amount,
  p.price,
  (t.amount / 1e18) * p.price AS amount_usd
FROM erc20.token_transfers t
LEFT JOIN prices.usd p
  ON t.token_address = p.contract_address
  AND date_trunc('minute', t.block_time) = p.minute
WHERE t.block_time > now() - interval '7 days'
```

🧠 Always match on both `token_address` and `minute` for price joins.

CASE: Create Custom Categories

Conditional logic with `CASE` lets you label rows:

```
SELECT
  CASE
    WHEN amount > 1000 THEN 'whale'
    WHEN amount > 100 THEN 'mid-tier'
    ELSE 'small'
  END AS size_category,
  COUNT(*) AS tx_count
FROM erc20.token_transfers
GROUP BY 1
```

Common Functions in Blockchain Analytics

- `COUNT(*)` : number of rows (events, transactions)
- `SUM(column)` : total tokens transferred or gas used
- `AVG(column)` : average gas fee, average swap size
- `MAX/MIN` : largest NFT sale, smallest transfer
- `RANK()/ROW_NUMBER() OVER (PARTITION BY ...)` : leaderboard logic
- `date_trunc('day', timestamp)` : daily grouping

Template Queries to Get You Started

Daily transaction count:

```
SELECT
  date_trunc('day', block_time) AS day,
  COUNT(*) AS txs
FROM ethereum.transactions
WHERE block_time > now() - interval '30 days'
GROUP BY 1
ORDER BY 1
```

Top NFT minters (past 7 days):

```
SELECT
  minter,
  COUNT(*) AS mints
FROM nft_ethereum.mints
WHERE block_time > now() - interval '7 days'
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10
```

Uniswap revenue in USD (last 30 days):

```
SELECT
  date_trunc('day', block_time) AS day,
  SUM(fee_amount_usd) AS daily_revenue
FROM uniswap_v3.fees
WHERE block_time > now() - interval '30 days'
GROUP BY 1
ORDER BY 1
```

Mistakes to Avoid

🚫 Querying full tables without a time filter → always use `WHERE block_time >`

🚫 Joining price data without `date_trunc('minute')` → you'll miss matches

🚫 Comparing uppercase and lowercase addresses → use `LOWER()` consistently

🚫 Using `SELECT *` in big tables → only select the fields you need

Why SQL Matters in Web3

SQL is the **native language of data querying**—and onchain data is just a new domain.

If you can write clear, thoughtful queries, you can:

- Understand protocol health
- Monitor DAO treasuries
- Detect suspicious activity
- Track product usage
- Inform investment decisions

You're not just writing queries—you're building visibility in a transparent economy.

Next: 06. Useful Queries — From Token Transfers to Whale Watching