Scenario

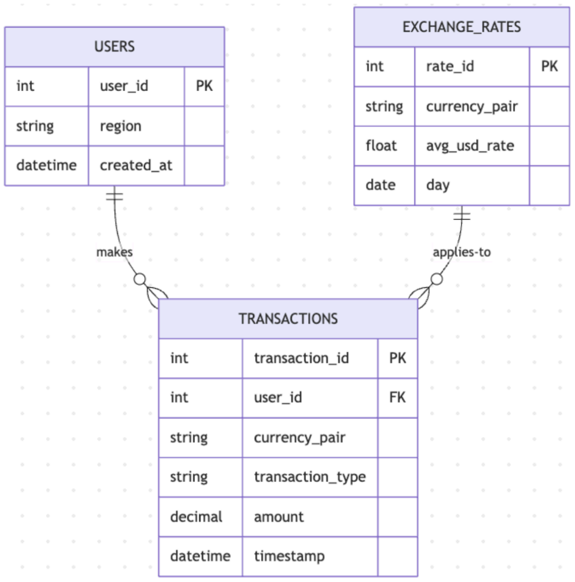
You're working on Coinbase's internal analytics pipeline. The pipeline aggregates total cryptocurrency trading volume across all supported coins using several internal data sources and presents it to executives in a daily dashboard. It uses complicated logic and sources contrasted with the Coinbase exchange API used in the previous question which is a raw feed from the Coinbase exchange.

Your Task

Yesterday, Brian Armstrong noticed a discrepancy: the internal dashboard showed $3 billion in trading volume on the Coinbase exchange, but the public Coinbase API showed $2.8 billion for the same time period. Your task is to diagnose the problem and propose solutions.

Data Model

The internal volume calculation uses the following database schema:



```
SELECT day, SUM(transactions.amount * exchange_rates.avg_usd_rate) AS volume
FROM users
JOIN transactions
  ON users.user_id = transactions.user_id
JOIN exchange_rates
  ON transactions.currency_pair = exchange_rates.currency_pair
  AND transactions.timestamp::DATE = exchange_rates.day
WHERE transactions.day BETWEEN CURRENT_DATE()-7 AND CURRENT_DATE()
GROUP BY 1
```

1. Root Cause Analysis

Question: What could cause the volume discrepancy?

First for some brief analysis, there is a $200 million discrepancy between what the internal dashboard shows and the public Coinbase API. Below, I will list out specific reasons why this discrepancy is prevalent.

1. **Missing Transaction Type Filter**

   The SQL query does not filter by transaction type. It sums ALL transactions, including deposits, withdrawals, transfers, and fees. Trading volume should only count actual buy and sell trades. If roughly 7 percent of the transaction amount comes from non-trading activities like deposits and withdrawals, this alone could explain the entire $ 200 million discrepancy. The public API likely only counts actual trades.

2. **Date Range Interpretation Difference**

   The WHERE clause uses BETWEEN CURRENT_DATE minus 7 AND CURRENT_DATE, which is inclusive on both ends. This means if the query runs on November 24th, it includes November 17th through November 24th, which is 8 full days, not 7 days. The public API may use exactly 7 days or exclude the current day. If November 24th had approximately 200 million dollars in trading volume, including that extra day, would explain the discrepancy.

3. **Column Name Mismatch Causing Timezone Issues**

   The WHERE clause filters transactions for a given day, but the schema shows only a timestamp column in the transactions table. The JOIN condition uses timestamp: DATE, but the WHERE uses day. If the day is computed separately or in a different timezone than the timestamp to date conversion, transactions near midnight could be assigned to other days in other parts of the query. This could cause transactions to be double-counted or cause the date range actually to span more than intended. We need to use UTC or a standard time zone.

4. **Multiple Exchange Rates Per Day Creating Duplicate Joins**

   The JOIN to exchange rates uses timestamp colon colon DATE equals exchange rates dot day. If the exchange rates table contains multiple rate entries for the same currency pair on the same day, each transaction would join to multiple rate records. For example, if rates are updated every hour, one transaction could join 24 different rate records, resulting in that transaction being counted 24 times. This would massively inflate the volume calculation.

5. **Inclusion of Off-Exchange or Over-the-Counter (OTC) Transactions**

   The internal pipeline may aggregate data from multiple sources, including on-exchange trades, over-the-counter trades, internal transfers, and other products like margin trading or futures. The public API likely only reports spot market trades that happen on the central Coinbase exchange. If the internal system includes OTC desk volume or other off-exchange activity totaling several hundred million dollars, this would create a discrepancy, since those transactions would not appear in the public API data.

2. Investigation Plan

Question: How would you systematically diagnose this issue?

Step 1: Verify the Schema and Column Existence

First, check if the day column actually exists in the transactions table. I would query the database schema or run a describe command on the transactions table to see all available columns. If the day column does not exist, this confirms there is a mismatch between the SQL query and the actual schema. This immediately tells me that the query is either broken or relying on some computed column that might not be working as expected.

Step 2: Check Transaction Type Distribution

Next, I would query the transactions table to identify distinct transaction types and calculate the total volume of each kind. I would group by transaction type and sum the amounts to see how much volume comes from trades, deposits, withdrawals, and other types. If I see that deposits and withdrawals account for roughly 200 million dollars or 7 percent of the total volume, this suggests that including non-trading transactions is the root cause. I would compare the total volume of only buy and sell transactions against the 2.8 billion from the public API.

Step 3: Validate the Date Range and Check for Duplicate Exchange Rates

I would check which dates are being included in the current query by grouping by date to see the volume for each day. I would count the number of distinct days to verify whether it is 7 or 8 days. At the same time, I would query the exchange rates table to check if there are multiple rate entries for the same currency pair on the same day. If I find 8 days of data or multiple rates per day, either could explain significant volume inflation.

Step 4: Check for Duplicate Transactions

I would count the total number of transaction records versus the number of distinct transaction IDs to see if there are any duplicates. If the counts differ, this means some transactions are recorded multiple times in the table. I would calculate what percentage of records are duplicates and estimate the volume impact. Even a small duplication rate like 1 or 2 percent could account for tens of millions of dollars in inflated volume.

Step 5: Run a Corrected Version of the Query

After identifying the likely issues from the previous steps, I would write a corrected version of the SQL query that fixes the problems I found. The corrected query would use timestamp colon colon DATE consistently, filter to only trade transaction types like buy and sell, use a proper 7-day date range that excludes the current day, and ensure each transaction joins to only one exchange rate record. I would run this corrected query and compare the result to the 2.8 billion from the public API.

Step 6: Compare Results by Currency Pair and Investigate Remaining Differences

To narrow down where any remaining discrepancy is coming from, I would break down the volume calculation by currency pair, comparing both the original and corrected queries against the public API. If there is still a gap after my corrections, I would investigate whether the internal system is pulling from additional data sources that the public API does not include, such as OTC trades, or whether test users and internal accounts are being appropriately filtered.

Step 7: Document the Findings in a Reconciliation Table

Finally, I would create a summary table listing each issue I found, the estimated volume impact of each, and whether fixing it would close the gap. For example, removing non-trade transactions might reduce volume by 150 million, fixing the date range might reduce it by 50 million, and so on. I would add up all the corrections and show that the sum of fixes brings the internal calculation in line with the public API number. This reconciliation would prove exactly which combination of issues caused the original 200-million-dollar discrepancy.

3. Prevention Strategy

Question: What data quality checks or monitoring would you add to catch this automatically in the future?

Data Quality Checks at Ingestion

Add a unique constraint on transaction_id in the transactions table to prevent duplicate records. Implement a check constraint on transaction_type to ensure only valid values like buy, sell, deposit, and withdrawal are allowed. Add a unique constraint on the combination of currency_pair and day in the exchange_rates table to ensure only one exchange rate exists per currency pair per day. These constraints would prevent bad data from entering the system in the first place.

Daily Reconciliation Job

Create an automated job that runs every morning at 2 AM to compare the previous day's internal volume calculation against the public Coinbase API volume. The job would calculate the percentage difference between the two sources and send an alert to the data engineering team if the discrepancy exceeds 2 percent. This alert would include both volume numbers and the exact percentage difference. The reconciliation results would be logged in a monitoring table so the team can track trends over time and quickly identify when discrepancies began.

Real-Time Monitoring Dashboard

Build a dashboard that displays key data quality metrics updated every 15 minutes. The dashboard would show the count of duplicate transactions detected, any missing exchange rate records, the breakdown of transaction types as percentages, and the current discrepancy between internal and public API volumes. The dashboard would also display alerts for any anomalies, such as sudden volume spikes that are more than 3 standard deviations above the 7-day average. This gives the team visibility into data quality issues as they happen.

Automated Alerting Rules

Set up three tiers of alerts. A critical P1 alert page is to notify the on-call engineer immediately if the volume on the internal versus public API differs by more than 5 percent. A P2 alert sends a Slack message to the data quality channel within one hour if duplicate transactions or missing exchange rates are detected. A P3 alert sends an email to the data team if daily volume shows a statistical anomaly. Each alert would include enough context to start investigating without needing to dig through logs. Many error-monitoring/logging tools can be integrated to enable AI-powered fixes.

SQL Query Improvements and Testing

Fix the current query to use timestamp colon colon DATE consistently throughout, add a filter for transaction_type IN buy and sell to exclude non-trading transactions, and change the date range to exclude the current day, ensuring exactly 7 days are captured. Create a materialized view that pre-aggregates daily trading volume using this corrected logic and refresh it once per day. Implement a change management process requiring any modifications to the volume calculation query to be peer reviewed and validated against 30 days of historical public API data before deployment.

Monthly API Validation

Run an automated monthly job on the first of each month that validates the internal calculation logic still matches the public API methodology. This job would compare volumes for the previous month, broken down by currency pair and by day. If systematic differences emerge in how specific pairs are calculated or if the correlation between internal and public numbers drops below 0.98, the job would flag this for investigation. This catches drift between the two systems before it becomes a larger problem.

Extra - My personal implementation using Databricks Tools:

In terms of experience, I've used Databricks, so here's how I would do it in a full Databricks flow using Databricks' technologies:

Databricks Implementation for Volume Monitoring Workflow

Step 1: Delta Live Tables - Data Ingestion with Quality Checks

Ingest transactions in real time with built-in expectations that fail on duplicate transaction IDs, null values in critical fields, and invalid transaction types. Write clean data to the bronze layer with schema evolution and change data capture enabled.

Step 2: Delta Live Tables - Silver Layer Transformation

Transform bronze transactions to silver layer by filtering to trade types only, joining to exchange rates with one-to-one validation, and aggregating daily volume by currency pair. Materialized view refreshes hourly with full audit trail.

Step 3: Databricks Workflows - Daily Reconciliation Job

Runs at 2 AM (or any agreed-upon time) daily with three tasks: query internal volume from the silver layer, call the Coinbase public API for the same period, compare results, and alert if the discrepancy exceeds 2 percent via email and Slack. Use integrations to send messages.

Step 4: Databricks AI and BI - Anomaly Detection

The AutoML forecasting model predicts expected daily volume and flags anomalies when actual values deviate by more than 3 standard deviations. Model trains weekly with MLflow tracking versions and metrics.

Step 5: Databricks SQL - Real-Time Monitoring Dashboard

The dashboard displays a comparison of internal versus public API volume, the current discrepancy percentage against the red alert threshold, transaction type distribution, and data quality metrics. Auto-refreshes every 15 minutes.

Step 6: Unity Catalog - Governance and Certified Views

Row-level security automatically filters out test users. The certified gold layer view implements the correct calculation logic with documented business rules. Column-level lineage tracks source fields to the final volume. Access controls restrict analysts to certified views only.

Step 7: Lakehouse Monitoring - Continuous Data Quality

Monitors duplicate counts, null rates, schema violations, and join cardinality issues. Generates automated alerts when quality metrics fall below thresholds and logs all violations to the alerts table.

Step 8: MLflow - Model Registry and Deployment

Stores anomaly detection model with version control, A/B testing capabilities, and an automated retraining pipeline. Tracks prediction accuracy and sends alerts when model performance degrades below baseline.