Day 12: Cryptocurrency Exchange: Matching Engine & Risk Management

Domain: Finance / Crypto Exchange / Market Microstructure

Difficulty: Expert+ — time-series, matching/aggregation, window functions, ledger balancing, anti-fraud checks, position exposure, and reconciliation.

Goal: Analyze orders, trades, user balances, fees, and risk flags to detect inconsistencies, compute realized P&L, VWAP, and user exposure across symbols.

Scenario description

You operate a centralized crypto exchange that supports multiple spot trading markets (e.g., BTC/USD, ETH/USD). Users place limit and market orders which the matching engine executes into trades (possible partial fills). The exchange records deposits/withdrawals and maintains ledger entries per user (credits/debits). Analysts must:

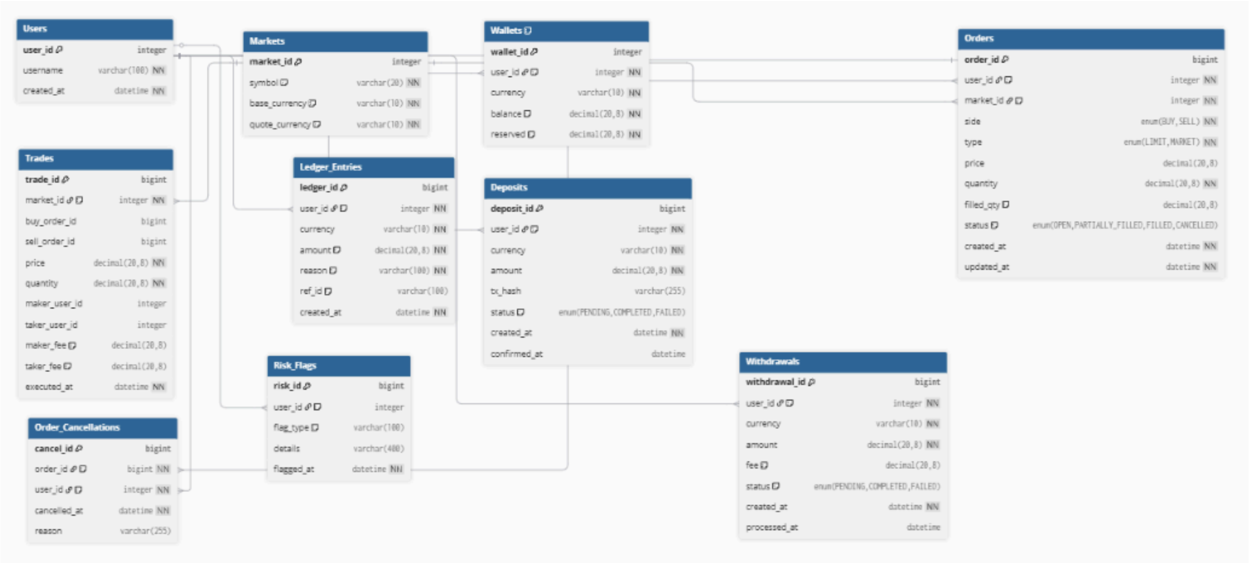
- Validate ledger integrity (no out-of-thin-air money)
- Compute VWAP and realized fees per market
- Identify users with risky behavior (e.g., wash trading, excessive cancellations, negative balances)
- Reconstruct order execution chains (partial fills)
- Reconcile trades vs ledgers

This dataset intentionally includes partial fills, cancelled orders, fees, maker/taker designations, and a few intentionally inconsistent ledger rows for detection.

Database schema (MySQL)

> Use MySQL 8+. Add suggested indexes after tables.

https://dbdiagram.io/d/69059d486735e11170be933c



Users		
user_id	integer	
username	varchar(100)	NN
created_at	datetime	NN

Markets		
market_id	integer	
symbol	varchar(20)	NN
base_currency	varchar(10)	NN
quote_currency	varchar(10)	NN

Wallets		
wallet_id	integer	
user_id	integer	NN
currency	varchar(10)	NN
balance	decimal(20,8)	NN
reserved	decimal(20,8)	NN

Orders		
order_id	bigint	
user_id	integer	NN
market_id	integer	NN
side	enum(BUY,SELL)	NN
type	enum(LIMIT,MARKET)	NN
price	decimal(20,8)	
quantity	decimal(20,8)	NN
filled_qty	decimal(20,8)	
status	enum(OPEN,PARTIALLY_FILLED,FILLED,CANCELLED)	
created_at	datetime	NN
updated_at	datetime	NN

Trades		
trade_id	bigint	
market_id	integer	NN
buy_order_id	bigint	
sell_order_id	bigint	
price	decimal(20,8)	NN
quantity	decimal(20,8)	NN
maker_user_id	integer	
taker_user_id	integer	
maker_fee	decimal(20,8)	
taker_fee	decimal(20,8)	
executed_at	datetime	NN

Ledger_Entries		
ledger_id	bigint	
user_id	integer	NN
currency	varchar(10)	NN
amount	decimal(20,8)	NN
reason	varchar(100)	NN
ref_id	varchar(100)	
created_at	datetime	NN

Deposits		
deposit_id	bigint	
user_id	integer	NN
currency	varchar(10)	NN
amount	decimal(20,8)	NN
tx_hash	varchar(255)	
status	enum(PENDING,COMPLETED,FAILED)	
created_at	datetime	NN
confirmed_at	datetime	

Risk_Flags		
risk_id	bigint	
user_id	integer	
flag_type	varchar(100)	
details	varchar(400)	
flagged_at	datetime	NN

Withdrawals		
withdrawal_id	bigint	
user_id	integer	NN
currency	varchar(10)	NN
amount	decimal(20,8)	NN
fee	decimal(20,8)	
status	enum(PENDING,COMPLETED,FAILED)	
created_at	datetime	NN
processed_at	datetime	

Order_Cancellations		
cancel_id	bigint	
order_id	bigint	NN
user_id	integer	NN
cancelled_at	datetime	NN
reason	varchar(255)	

1. Users

```
CREATE TABLE Users (
  user_id INT PRIMARY KEY,
  username VARCHAR(100) UNIQUE NOT NULL,
  created_at DATETIME NOT NULL
);
```

2. Markets

```
CREATE TABLE Markets (
  market_id INT PRIMARY KEY,
  symbol VARCHAR(20) UNIQUE NOT NULL, -- e.g., 'BTC_USD'
  base_currency VARCHAR(10) NOT NULL, -- e.g., BTC
  quote_currency VARCHAR(10) NOT NULL -- e.g., USD
);
```

3. Wallets

(one wallet per user per currency)

```
CREATE TABLE Wallets (
  wallet_id INT PRIMARY KEY,
  user_id INT NOT NULL,
  currency VARCHAR(10) NOT NULL,
  balance DECIMAL(20,8) NOT NULL DEFAULT 0,
  reserved DECIMAL(20,8) NOT NULL DEFAULT 0,
  FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
-- UNIQUE (user_id, currency) recommended in production
```

4. Orders

```
CREATE TABLE Orders (
  order_id BIGINT PRIMARY KEY,
  user_id INT NOT NULL,
  market_id INT NOT NULL,
  side ENUM('BUY','SELL') NOT NULL,
  type ENUM('LIMIT','MARKET') NOT NULL,
  price DECIMAL(20,8) NULL,      -- null for market
  quantity DECIMAL(20,8) NOT NULL,
  filled_qty DECIMAL(20,8) NOT NULL DEFAULT 0,
  status ENUM('OPEN','PARTIALLY_FILLED','FILLED','CANCELLED') DEFAULT 'OPEN',
  created_at DATETIME NOT NULL,
  updated_at DATETIME NOT NULL,
  FOREIGN KEY (user_id) REFERENCES Users(user_id),
  FOREIGN KEY (market_id) REFERENCES Markets(market_id)
);
```

5. Trades

(each trade is the result of matching — maker & taker sides are explicit)

```
CREATE TABLE Trades (
  trade_id BIGINT PRIMARY KEY,
  market_id INT NOT NULL,
  buy_order_id BIGINT,
  sell_order_id BIGINT,
  price DECIMAL(20,8) NOT NULL,
  quantity DECIMAL(20,8) NOT NULL,
  maker_user_id INT,
  taker_user_id INT,
  maker_fee DECIMAL(20,8) NOT NULL DEFAULT 0,
  taker_fee DECIMAL(20,8) NOT NULL DEFAULT 0,
  executed_at DATETIME NOT NULL,
  FOREIGN KEY (market_id) REFERENCES Markets(market_id)
);
```

6. Ledger_Entries

(atomic ledger movements; positive = credit to wallet, negative = debit)

```
CREATE TABLE Ledger_Entries (
  ledger_id BIGINT PRIMARY KEY,
  user_id INT NOT NULL,
  currency VARCHAR(10) NOT NULL,
  amount DECIMAL(20,8) NOT NULL, -- positive credit, negative debit
  reason VARCHAR(100) NOT NULL,   -- 'Deposit','Withdraw','Trade_Fill','Fee','Cancel_Refund', etc.
  ref_id VARCHAR(100),            -- e.g., trade_id or order_id
  created_at DATETIME NOT NULL,
  FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

7. Deposits

```
CREATE TABLE Deposits (
  deposit_id BIGINT PRIMARY KEY,
  user_id INT NOT NULL,
  currency VARCHAR(10) NOT NULL,
  amount DECIMAL(20,8) NOT NULL,
  tx_hash VARCHAR(255),
  status ENUM('PENDING','COMPLETED','FAILED') DEFAULT 'PENDING',
  created_at DATETIME NOT NULL,
  confirmed_at DATETIME NULL,
  FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

8. Withdrawals

```
CREATE TABLE Withdrawals (
  withdrawal_id BIGINT PRIMARY KEY,
  user_id INT NOT NULL,
  currency VARCHAR(10) NOT NULL,
  amount DECIMAL(20,8) NOT NULL,
  fee DECIMAL(20,8) DEFAULT 0,
  status ENUM('PENDING','COMPLETED','FAILED') DEFAULT 'PENDING',
  created_at DATETIME NOT NULL,
  processed_at DATETIME NULL,
  FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

9. Order_Cancellations

```
CREATE TABLE Order_Cancellations (
  cancel_id BIGINT PRIMARY KEY,
  order_id BIGINT NOT NULL,
  user_id INT NOT NULL,
  cancelled_at DATETIME NOT NULL,
  reason VARCHAR(255),
  FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
```

10. Risk_Flags

```
CREATE TABLE Risk_Flags (
  risk_id BIGINT PRIMARY KEY,
  user_id INT,
  flag_type VARCHAR(100), -- 'WashTrade','HighCancelRate','NegativeBalance','SuspiciousVolume'
  details VARCHAR(400),
  flagged_at DATETIME NOT NULL
);
```

Sample data (representative; includes edge cases)

```
-- Users
INSERT INTO Users VALUES
(1,'alice','2024-01-01 09:00:00'),
(2,'bob','2024-02-01 10:00:00'),
(3,'carol','2024-03-15 11:30:00');

-- Markets
INSERT INTO Markets VALUES
(1,'BTC_USD','BTC','USD'),
(2,'ETH_USD','ETH','USD');

-- Wallets (balances may intentionally include a negative for anomaly)
INSERT INTO Wallets VALUES
```

```
(1001,1,'USD',100000.00,0.00000000),
(1002,1,'BTC',5.00000000,0.00000000),
(1003,2,'USD',50000.00,0.00000000),
(1004,2,'BTC',0.10000000,0.00000000),
(1005,3,'USD',10000.00,0.00000000),
(1006,3,'BTC',0.00000000,0.00000000);

-- Orders (partial fills, market order, cancelled)
INSERT INTO Orders VALUES
(9001,1,1,'BUY','LIMIT',30000.00,1.5,1.0,'PARTIALLY_FILLED','2024-10-01 09:10:00','2024-10-01 09:15:00'),
(9002,2,1,'SELL','LIMIT',30000.00,1.0,1.0,'FILLED','2024-10-01 09:11:00','2024-10-01 09:11:30'),
(9003,1,2,'SELL','MARKET',NULL,2.0,2.0,'FILLED','2024-10-01 09:30:00','2024-10-01 09:30:10'),
(9004,3,1,'BUY','LIMIT',31000.00,0.5,0.0,'CANCELLED','2024-10-01 10:00:00','2024-10-01 10:05:00');

-- Trades (partial fill trade linking 9001 buy & 9002 sell)
INSERT INTO Trades VALUES
(8001,1,9001,9002,30000.00,1.0,1,2,0.0005,0.001, '2024-10-01 09:11:00'),
(8002,2,NULL,9003,1800.00,2.0,NULL,1,0.0000,0.001, '2024-10-01 09:30:05'); -- market fill, buy order absent (taker buys from market)

-- Ledger entries (credits/debits). Intentionally include inconsistent ledger row to flag later
INSERT INTO Ledger_Entries VALUES
(7001,1,'BTC',1.0,'Trade_Fill','8001','2024-10-01 09:11:01'),  -- buyer receives BTC
(7002,1,'USD', -30000.00,'Trade_Fill','8001','2024-10-01 09:11:01'), -- buyer pays USD
(7003,2,'BTC', -1.0,'Trade_Fill','8001','2024-10-01 09:11:01'), -- seller gives BTC
(7004,2,'USD',30000.00,'Trade_Fill','8001','2024-10-01 09:11:01'),  -- seller receives USD
(7005,1,'USD', -3600.00,'Fee','8002','2024-10-01 09:30:06'), -- fee incorrectly large (intentional anomaly)
```

```
(7006,1,'BTC',2.0,'Trade_Fill','8002','2024-10-01 09:30:06'), -- alice sold 2 ETH? inconsistent currency - anomaly
(7007,1,'USD',1800.00,'Trade_Fill','8002','2024-10-01 09:30:06'); -- buyer paid for market order

-- Deposits & withdrawals
INSERT INTO Deposits VALUES
(6001,1,'USD',50000.00,'tx123','COMPLETED','2024-09-01 10:00:00','2024-09-01 12:00:00');

INSERT INTO Withdrawals VALUES
(6002,2,'USD',1000.00,5.00,'COMPLETED','2024-09-05 11:00:00','2024-09-05 11:30:00');

-- Order cancellations
INSERT INTO Order_Cancellations VALUES
(5001,9004,3,'2024-10-01 10:05:00','User requested cancel');

-- Risk flags (seeded)
INSERT INTO Risk_Flags VALUES
(4001,3,'HighCancelRate','5 cancels in 1 hour','2024-10-01 11:00:00');
```

> Notes on sample data:
>
> - Order 9001 (alice buy) partially filled by trade 8001 (1.0 filled of 1.5).
> - Trade 8002 is a market trade with possible ledger anomalies intentionally included (currency mismatch, large fee).
> - Wallets include normal and edge balances for detection.

ERD (textual)

```
Users (1) ——< Wallets (M)
Users (1) ——< Orders (M) ——< Trades (M) >— Orders (M)
```

```
Trades (1) ——< Ledger_Entries (M)
Users (1) ——< Ledger_Entries (M)
Markets (1) ——< Orders (M)
Users (1) ——< Deposits (M)
Users (1) ——< Withdrawals (M)
Users (1) ——< Risk_Flags (M)
```

5 SQL Questions (increasing difficulty)

1. **Easy:** Q1 — For each market, compute 1-minute VWAP and total trade volume for `2024-10-01 09:10:00` to `2024-10-01 09:12:00`.

   Output: `market_id, minute_bucket, vwap_price, total_volume`.

2. **Medium:** Q2 — Find users who have a negative USD wallet balance or negative balance in any currency (i.e., `balance + reserved < 0`) and list `user_id, username, currency, balance, reserved, net_balance`.

3. **Hard:** Q3 — Reconstruct the execution chain for order `9001`: show all trades that filled it, with `trade_id, executed_at, quantity, counterparty_user_id`, and the remaining unfilled qty for the order. (Handle partial fills.)

4. **Difficult:** Q4 — Identify potential wash trades in the last 24 hours: trades where `maker_user_id = taker_user_id` OR trades where the same two user_ids trade both directions within 1 minute and the trade amounts and prices are identical. Output candidate `trade_id` pairs and involved user_ids.

5. **Expert:** Q5 — Ledger reconciliation: For each trade in a given time window (`2024-10-01 09:00:00` — `2024-10-01 10:00:00`), validate that corresponding ledger entries exist and sum to zero per trade across currencies (i.e., sum of amounts across all ledger entries for `ref_id = trade_id` should equal 0 when converted to a common base — but without FX pairs assume currency-level balancing: for each trade, total base currency change + total quote currency change + fees should net properly). Produce a report showing `trade_id, expected_base_delta, expected_quote_delta, ledger_base_sum, ledger_quote_sum, discrepancy_flag` for `BTC_USD` (market_id=1) trades only. Explain assumptions.

Solutions (MySQL queries + explanations, tips)

> All queries are MySQL 8+ and assume tables/indexes as defined. Add indexes for production: Trades(executed_at, market_id), Orders(order_id, user_id), Ledger_Entries(ref_id, currency), Wallets(user_id, currency), Trades(maker_user_id,taker_user_id).

Q1 — 1-minute VWAP & total volume (09:10–09:12)

```
-- Q1: 1-minute VWAP & volume for each market in the 3-minute window
SELECT
  t.market_id,
  DATE_FORMAT(t.executed_at, '%Y-%m-%d %H:%i:00') AS minute_bucket,
  ROUND(SUM(t.price * t.quantity) / NULLIF(SUM(t.quantity),0), 8) AS vwap_price,
  SUM(t.quantity) AS total_volume
FROM Trades t
WHERE t.executed_at >= '2024-10-01 09:10:00'
  AND t.executed_at <  '2024-10-01 09:13:00'
GROUP BY t.market_id, minute_bucket
ORDER BY t.market_id, minute_bucket;
```

Explanation:

- VWAP = sum(price * quantity) / sum(quantity) per minute bucket.
- `DATE_FORMAT(..., '%Y-%m-%d %H:%i:00')` buckets by minute.
- `NULLIF` avoids division by zero.

Edge cases: minutes with zero volume will be excluded; to include them, LEFT JOIN against a minute-series table.

Indexes: `Trades(executed_at, market_id)`.

Q2 — Users with negative net balance

```
-- Q2: Users with negative net balance in any currency
SELECT
  w.user_id,
  u.username,
  w.currency,
  w.balance,
  w.reserved,
  (w.balance + w.reserved) AS net_balance
FROM Wallets w
JOIN Users u ON w.user_id = u.user_id
WHERE (w.balance + w.reserved) < 0
ORDER BY net_balance ASC;
```

Explanation:

- `reserved` often holds funds locked for open orders; negative net implies oversubscription or failure to reserve/rollback.
- Use this list for immediate risk checks or forced liquidation.

Index: `Wallets(user_id, currency)`.

Q3 — Reconstruct execution chain for order 9001

```
-- Q3: Trades that filled order 9001 and remaining unfilled qty
WITH order_trades AS (
  SELECT
    t.trade_id,
    t.executed_at,
    t.quantity,
    CASE WHEN t.buy_order_id = 9001 THEN t.taker_user_id WHEN t.sell_order_id = 9001 THEN t.maker_user_id ELSE NULL END AS counterparty_user_id,
    CASE WHEN t.buy_order_id = 9001 THEN 'BUY_SIDE' WHEN t.sell_order_id = 9001 THEN 'SELL_SIDE' ELSE 'OTHER' END AS side_of_order
  FROM Trades t
  WHERE t.buy_order_id = 9001 OR t.sell_order_id = 9001
  ORDER BY t.executed_at
)
SELECT
  ot.trade_id,
  ot.executed_at,
  ot.quantity,
  ot.counterparty_user_id,
  ot.side_of_order,
  o.quantity AS order_quantity,
  o.filled_qty AS order_filled_qty,
  (o.quantity - o.filled_qty) AS remaining_qty
FROM order_trades ot
CROSS JOIN (
  SELECT quantity, filled_qty FROM Orders WHERE order_id = 9001
) o
ORDER BY ot.executed_at;
```

Explanation:

- Select trades where `buy_order_id` or `sell_order_id` equals the order; determine the counterparty depending on which side the order was on.
- The order row shows `filled_qty` (current state) and remaining quantity = `quantity - filled_qty`.
- For partial fills, multiple trades will be returned; sum(trade.quantity) should equal `filled_qty` (sanity check).

Sanity check (ensure sums match):

```
SELECT
  o.order_id,
  o.quantity,
  o.filled_qty AS order_filled_qty,
  COALESCE(SUM(t.quantity),0) AS trades_sum_qty,
  (o.filled_qty - COALESCE(SUM(t.quantity),0)) AS mismatch
```

```
FROM Orders o
LEFT JOIN Trades t ON t.buy_order_id = o.order_id OR t.sell_order_id = o.order_id
WHERE o.order_id = 9001
GROUP BY o.order_id, o.quantity, o.filled_qty;
```

If mismatch != 0, investigate ledger/trade insertion failures.

Index: `Trades(buy_order_id)`, `Trades(sell_order_id)`.

Q4 — Detect potential wash trades

```
-- Q4: Candidate wash trades (same maker & taker OR immediate cross trades between two users)
WITH recent_trades AS (
  SELECT trade_id, maker_user_id, taker_user_id, market_id, price, quantity, executed_at
  FROM Trades
  WHERE executed_at >= NOW() - INTERVAL 1 DAY
),
-- same-user trades (maker == taker)
same_user AS (
  SELECT trade_id, maker_user_id AS user_id, 'SELF_TRADE' AS reason, executed_at
  FROM recent_trades
  WHERE maker_user_id IS NOT NULL AND maker_user_id = taker_user_id
),
-- reciprocal trade pairs: user A trades with B then within 1 minute B trades with A at same price & qty
recip AS (
  SELECT
    t1.trade_id AS trade_a,
    t2.trade_id AS trade_b,
    t1.maker_user_id AS maker_a,
    t1.taker_user_id AS taker_a,
    t2.maker_user_id AS maker_b,
    t2.taker_user_id AS taker_b,
    t1.price, t1.quantity,
    TIMESTAMPDIFF(SECOND, t1.executed_at, t2.executed_at) AS delta_second
s
  FROM recent_trades t1
  JOIN recent_trades t2
    ON t1.market_id = t2.market_id
   AND t1.price = t2.price
   AND t1.quantity = t2.quantity
   AND t1.maker_user_id = t2.taker_user_id
   AND t1.taker_user_id = t2.maker_user_id
   AND t2.executed_at >= t1.executed_at
   AND TIMESTAMPDIFF(SECOND, t1.executed_at, t2.executed_at) <= 60
)
SELECT 'SELF' AS type, s.trade_id, s.user_id, s.reason, s.executed_at
FROM same_user s
UNION ALL
SELECT 'RECIPROCAL' AS type, CONCAT(r.trade_a,'|',r.trade_b) AS trade_pair,
CONCAT(r.maker_a,'↔',r.taker_a) AS user_pair, CONCAT('delta_s=',r.delta_se
conds) AS reason, NULL
FROM recip r
ORDER BY type;
```

Explanation:

- `same_user` picks trades where maker==taker (clear wash).
- `recip` finds reciprocal trades between same two users within 60 seconds with identical price/qty — suspicious.
- Returns pairs of trade_ids for investigation.

Caveats: legitimate OTC or market-maker strategies can look similar; follow-up with KYC/regulatory rules needed.

Index: `Trades(executed_at, maker_user_id, taker_user_id, price, quantity)` .

Q5 — Ledger reconciliation for BTC_USD trades (market_id=1) in 09:00–10:00

Goal & assumptions:

- For a BTC_USD trade, the base currency is BTC and quote is USD.
- Expected ledger effects per trade (simplified):
  - Buyer: +BTC (base) * quantity; -USD (quote) * price*quantity; -taker_fee (in USD) if taker
  - Seller: -BTC * quantity; +USD * price*quantity; -maker_fee (in USD) if maker
- Ledger entries for each trade should reflect these movements with `ref_id = trade_id`.
- We compare expected deltas to actual ledger sums grouped by currency. If sums mismatch, mark discrepancy.

```
-- Q5: Ledger reconciliation for BTC_USD (market_id = 1) between 09:00 and 10:00
WITH trades_window AS (
  SELECT t.trade_id, t.price, t.quantity, t.maker_user_id, t.taker_user_id, t.maker_fee, t.taker_fee, t.executed_at
  FROM Trades t
  WHERE t.market_id = 1
    AND t.executed_at >= '2024-10-01 09:00:00'
    AND t.executed_at <  '2024-10-01 10:00:00'
),
expected AS (
  -- compute expected base (BTC) and quote (USD) deltas per trade (positive = credit to user)
  SELECT
    tw.trade_id,
    -- base currency (BTC) expected deltas per user role (buyer +, seller -)
    tw.quantity AS expected_base_buyer,
    -tw.quantity AS expected_base_seller,
    -- quote currency (USD) expected deltas
    - (tw.price * tw.quantity) AS expected_quote_buyer,
    (tw.price * tw.quantity) AS expected_quote_seller,
    -- fees assumed in USD; maker_fee applies to maker_user_id, taker_fee to taker_user_id
    tw.maker_fee AS maker_fee_amt,
    tw.taker_fee AS taker_fee_amt,
    tw.maker_user_id,
    tw.taker_user_id
  FROM trades_window tw
),
-- aggregate ledger entries for each trade by currency
ledger_agg AS (
  SELECT
    le.ref_id AS trade_id,
    le.currency,
    SUM(le.amount) AS ledger_sum_amount
  FROM Ledger_Entries le
  WHERE le.ref_id IS NOT NULL
    AND le.ref_id IN (SELECT CAST(trade_id AS CHAR) FROM trades_window) -- ref_id stored as string
  GROUP BY le.ref_id, le.currency
)
SELECT
  e.trade_id,
  e.maker_user_id,
  e.taker_user_id,
  -- expected deltas per currency (BTC & USD)
  e.expected_base_buyer AS expected_btc_buyer,
  e.expected_base_seller AS expected_btc_seller,
  e.expected_quote_buyer AS expected_usd_buyer,
  e.expected_quote_seller AS expected_usd_seller,
  -- ledger sums
  COALESCE( (SELECT ledger_sum_amount FROM ledger_agg la WHERE la.trade_id = e.trade_id AND la.currency = 'BTC'), 0) AS ledger_btc_sum,
  COALESCE( (SELECT ledger_sum_amount FROM ledger_agg la WHERE la.trade_id = e.trade_id AND la.currency = 'USD'), 0) AS ledger_usd_sum,
  -- basic discrepancy logic: ledger sums should equal (expected_btc_buyer + expected_btc_seller) = 0 for BTC across trade
  CASE
    WHEN COALESCE( (SELECT ledger_sum_amount FROM ledger_agg la WHERE la.trade_id = e.trade_id AND la.currency = 'BTC'), 0)
        <> (e.expected_base_buyer + e.expected_base_seller) THEN 1
    WHEN COALESCE( (SELECT ledger_sum_amount FROM ledger_agg la WHERE la.trade_id = e.trade_id AND la.currency = 'USD'), 0)
        <> (e.expected_quote_buyer + e.expected_quote_seller - e.maker_fee_amt - e.taker_fee_amt) THEN 1
    ELSE 0
  END AS discrepancy_flag
FROM expected e
ORDER BY e.trade_id;
```

Explanation & assumptions:

- `expected_base_buyer + expected_base_seller` should sum to 0 (buyer +Q, seller -Q) — net zero across trade in base currency. Similarly, USD changes plus fees should net zero (seller +price*qty, buyer -price*qty, minus fees).
- Ledger entries are aggregated by `ref_id` (trade_id stored as string). We compare ledger sums for 'BTC' and 'USD' to expected deltas.
- Fees assumed to be in USD and recorded as negative ledger entries for the paying user in USD; the exchange fee income ledger entry is not modeled here — if present, it would offset the fee debits.

What discrepancy_flag = 1 means: transaction-level ledger mismatch — requires urgent reconciliation (example in sample data trade 8002 has inconsistent ledger rows and will be flagged).

Index: `Ledger_Entries(ref_id, currency)`.

Additional checks & maintenance queries

- **Check: trades vs order filled_qty consistency**

```
SELECT o.order_id, o.quantity, o.filled_qty, COALESCE(SUM(t.quantity),0) AS trades_sum
FROM Orders o
LEFT JOIN Trades t ON t.buy_order_id = o.order_id OR t.sell_order_id = o.order_id
GROUP BY o.order_id
HAVING ABS(o.filled_qty - COALESCE(SUM(t.quantity),0)) > 0.00000001;
```

- **Compute realized fees per user in a time window**

```
SELECT le.user_id, le.currency, SUM(-le.amount) AS total_fees_paid
FROM Ledger_Entries le
WHERE le.reason = 'Fee' AND le.created_at BETWEEN '2024-10-01 00:00:00'
AND '2024-10-02 00:00:00'
GROUP BY le.user_id, le.currency;
```

Optimization & production tips

1. **Indexes:** trades on `(market_id, executed_at)`, `(maker_user_id, taker_user_id)`, ledger entries on `(ref_id, currency)`, wallets on `(user_id, currency)`, orders on `(user_id, status)`.
2. **Atomicity:** matching engine must produce trades and ledger entries within a single transaction or via reliable ordered logs (WAL) to avoid orphan trades/ledgers.
3. **Partitioning:** partition `Trades` and `Ledger_Entries` by date for fast time-range scans.
4. **Denormalized summaries:** maintain per-market per-minute aggregates (VWAP, volume) as materialized tables for dashboards.
5. **Monitoring & Alerts:** pipeline to scan `discrepancy_flag` results and flag risk teams; run hourly reconciliations.

6. **Audit trail:** include unique identifiers (msg_seq, match_seq) in trades and ledgers to trace causality.
7. **Data types:** use DECIMAL with sufficient precision; consider integer atomic units (satoshis for BTC) to avoid floating point rounding.
8. **Simulate at scale:** use message queues and idempotent handlers to process matching and ledger posting.