Sprint 2

REQUIREMENTS

Story ID	Story description
US1	As an Analyst, I want to query portfolio returns by date range so that I can measure performance versus benchmarks. *(DB design completed in Sprint 1)*
US4	As an Analyst, I want to run a performance attribution report (contribution to return by asset/strategy) so that I can identify drivers of performance. *(DB design completed in Sprint 1)*
US3	As an Analyst, I want to compute rolling 12-month or custom rolling Sharpe for the portfolio so that I can assess risk-adjusted performance. *(DB design completed in Sprint 1)*
US6	As a PM, I want to simulate portfolio rebalancing based on target strategy allocations so that I can test different allocation scenarios. *(New for Sprint 2)*
US7	As an Analyst, I want to calculate portfolio Value at Risk (VaR) for a selected confidence interval so that I can evaluate downside risk. *(New for Sprint 2)*
US8	As an Admin, I want to automatically flag trades with inconsistent FX rates or missing data so that data quality is maintained. *(New for Sprint 2)*
US2	As a PM, I want to see current positions and exposures by sector/asset so that I can decide rebalances. *(DB design completed in Sprint 1)*
US5	As an Admin, I want data integrity checks on trades so that dirty data is flagged. *(DB design completed in Sprint 1)*

CONCEPTUAL DESIGN

Entity:Fund

Attributes:

- fund_id [PK]
- name
- currency
- inception_date
- benchmark_index

Entity:Portfolio

Attributes:

- portfolio_id [PK]
- fund_id [FK - Fund.fund_id]
- name
- start_date
- strategy

Entity:Investor

Attributes:

- investor_id [PK]
- name
- investor_type (institutional / retail)
- contact_info

Entity:Asset

Attributes:

- asset_id [PK]
- ticker
- name
- asset_type (Equity/Bond/ETF/FX/Derivative)
- currency
- sector

Entity:Price

Attributes:

- price_id [PK]
- asset_id [FK - Asset.asset_id]
- price_date
- close_price

- source

Entity:Trade

Attributes:

- trade_id [PK]
- portfolio_id [FK - Portfolio.portfolio_id]
- asset_id [FK - Asset.asset_id]
- trade_date
- side (BUY/SELL)
- quantity
- price
- currency
- trade_fx_rate
- fees

Entity:Position(derived/aggregated)

Attributes:

- position_id [PK]
- portfolio_id [FK]
- asset_id [FK]
- as_of_date
- quantity
- market_value (derived using Price)

Entity:DailyReturn

Attributes:

- return_id [PK]
- portfolio_id [FK]
- return_date
- daily_return_pct
- nav

Entity:StrategyAllocation

Attributes:

- allocation_id [PK]
- portfolio_id [FK]
- strategy_name
- target_pct

Relationships:

- Fund 1 - Portfolio (Fund has many Portfolios)
- Portfolio 1 - Trade (Portfolio records many trades)

- Asset 1 - Price (Asset has many daily prices)
- Portfolio 1 - DailyReturn (Portfolio has many daily return records)
- Portfolio 1 - Position (Portfolio has many positions)
- Portfolio 1 - StrategyAllocation (Portfolio has multiple allocation targets)

Participation:

- Portfolio has total participation with Fund.
- Trade has partial participation with Portfolio.
- StrategyAllocation has total participation with Portfolio.

LOGICAL DESIGN WITH HIGHEST NORMAL FORM IDENTIFICATION

Table:strategy_allocation*(New)*

Columns:

- allocation_id SERIAL PRIMARY KEY
- portfolio_id INT REFERENCES portfolio(portfolio_id) ON DELETE CASCADE
- strategy_name VARCHAR(100)
- target_pct NUMERIC(5,2)

Normalization:3NF

Indexes:

- Non-clustered index on (portfolio_id) for fast lookup of allocations per portfolio.

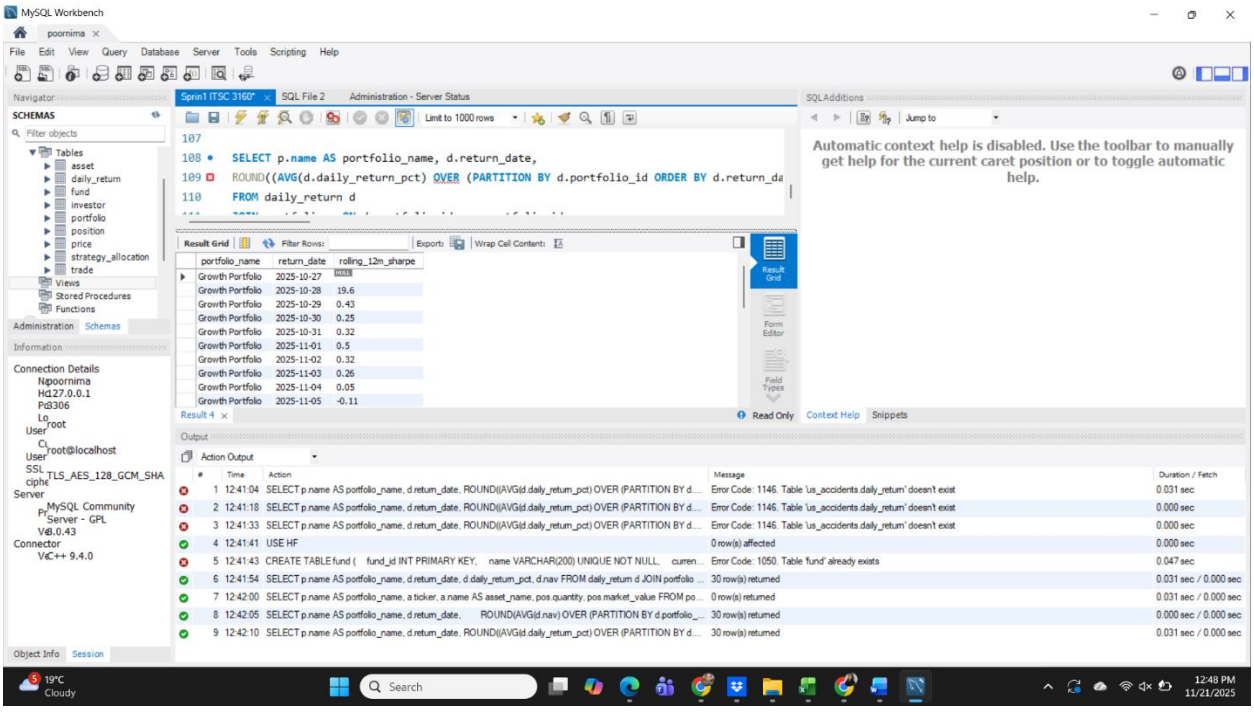
Table: trade *(updated for data integrity)*

- Add CHECK constraint for valid FX rates: CHECK(trade_fx_rate > 0)
- Add trigger for missing trade price/quantity

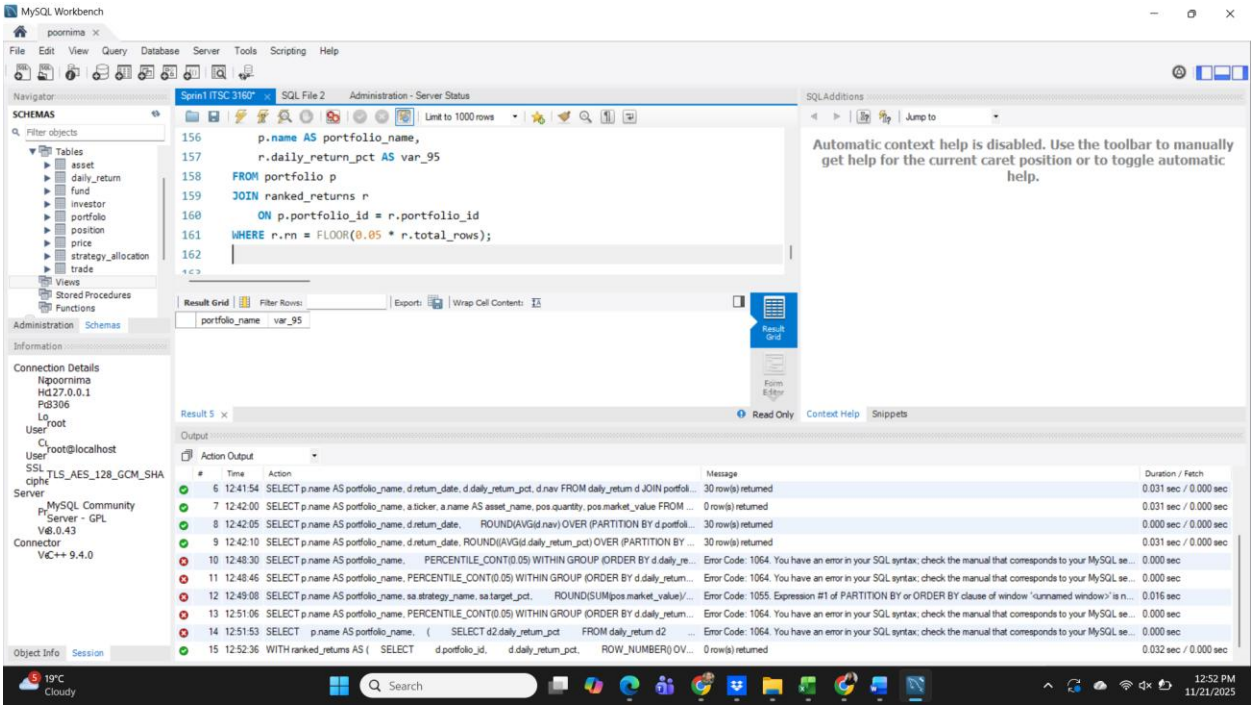
Other tables remain as in Sprint 1, with derived calculations for VaR and rolling Sharpe now handled through SQL or stored functions.

SQL QUERIES

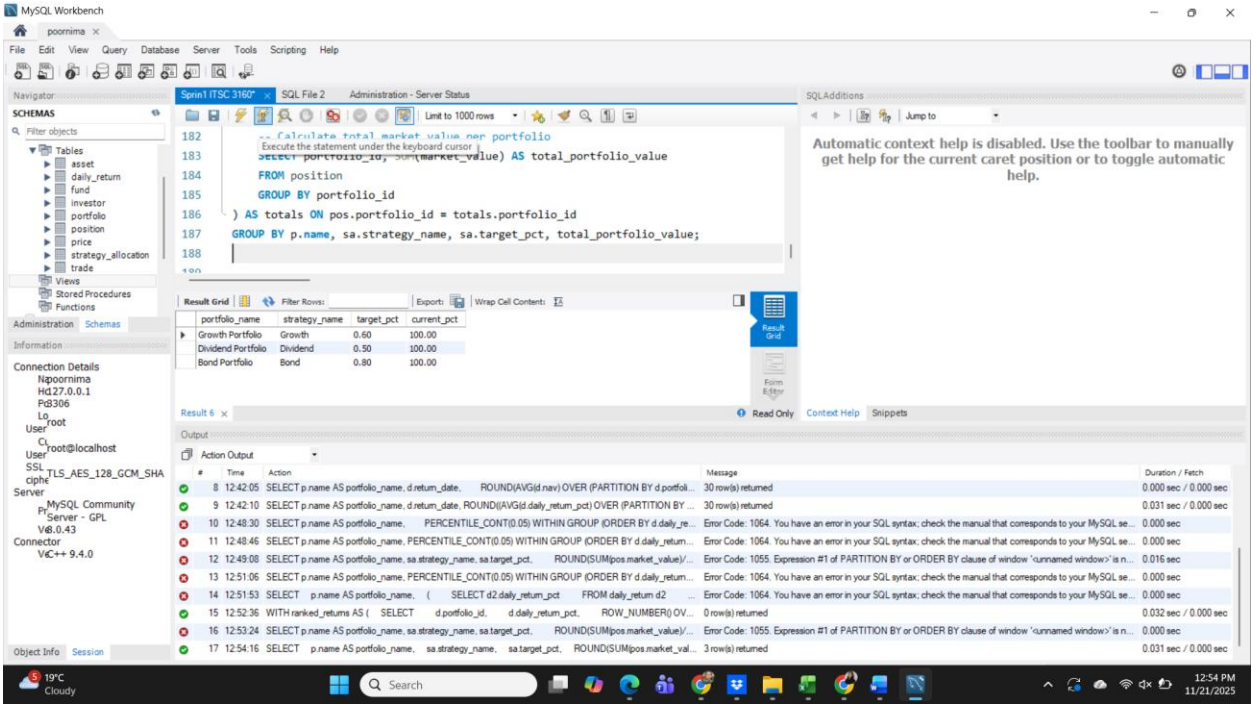
1. SELECT p.name AS portfolio_name, d.return_date,
ROUND((AVG(d.daily_return_pct) OVER (PARTITION BY d.portfolio_id ORDER BY d.return_date ROWS BETWEEN 252 PRECEDING AND CURRENT ROW)/NULLIF(STDDEV(d.daily_return_pct) OVER (PARTITION BY d.portfolio_id ORDER BY d.return_date ROWS BETWEEN 252 PRECEDING AND CURRENT ROW),0)),2) AS rolling_12m_sharpe
FROM daily_return d
JOIN portfolio p ON d.portfolio_id = p.portfolio_id
ORDER BY p.portfolio_id, d.return_date;



2. WITH ranked_returns AS (
    SELECT
        d.portfolio_id,
        d.daily_return_pct,
        ROW_NUMBER() OVER (PARTITION BY d.portfolio_id ORDER BY d.daily_return_pct) AS rn,
        COUNT(*) OVER (PARTITION BY d.portfolio_id) AS total_rows
    FROM daily_return d
)
SELECT
    p.name AS portfolio_name,
    r.daily_return_pct AS var_95
FROM portfolio p
JOIN ranked_returns r
    ON p.portfolio_id = r.portfolio_id
WHERE r.rn = FLOOR(0.05 * r.total_rows);



3. SELECT
    p.name AS portfolio_name,
    sa.strategy_name,
    sa.target_pct,
    ROUND(SUM(pos.market_value) / total_portfolio_value * 100, 2) AS current_pct
FROM position pos
JOIN portfolio p ON pos.portfolio_id = p.portfolio_id
JOIN strategy_allocation sa ON p.portfolio_id = sa.portfolio_id
JOIN (
    -- Calculate total market value per portfolio
    SELECT portfolio_id, SUM(market_value) AS total_portfolio_value
    FROM position
    GROUP BY portfolio_id
) AS totals ON pos.portfolio_id = totals.portfolio_id
GROUP BY p.name, sa.strategy_name, sa.target_pct, total_portfolio_value;



VIEWS AND STORED PROGRAMS

View: vw_portfolio_exposure

- Goal: Show current positions and exposures by asset type/sector for PM/Analyst review.

Stored Procedure: sp_rebalance_simulation

- Parameters: IN portfolio_id INT, IN target_date DATE
- Goal: Simulate portfolio rebalancing to match target allocations, adjusting positions and producing a summary report.

Stored Function: fn_portfolio_var

- Parameters: IN portfolio_id INT, IN confidence_level DECIMAL(4,2)
- Goal: Calculate historical VaR for a portfolio at the specified confidence level; returns numeric value.

Trigger: trg_trade_integrity *(AFTER INSERT/UPDATE on trade)*

- Goal: Automatically validate trade data (price, quantity, FX rate) and flag anomalies in a log table.