Design and Implementation of a Relational Database for Futures & Options (F&O) Trading Data

1. Introduction

The objective of this project is to design and implement a scalable relational database to store and analyse high-volume Futures & Options (F&O) market data from Indian exchanges. The dataset used for this implementation is the *"NSE Future and Options Dataset (3M)"* obtained from Kaggle, containing over 2.5 million rows of end-of-day trading data.

The solution emphasizes proper database normalization, efficient query design, and performance optimization for analytical workloads such as open interest analysis, volatility calculation, and cross-exchange comparisons. The schema is designed to support future ingestion from multiple exchanges including NSE, BSE, and MCX.

---

2. Dataset Overview

Each record in the dataset represents a single trading day for a specific futures or options contract. The dataset includes the following key attributes:

- Instrument type (Future or Option)
- Symbol (e.g., NIFTY, BANKNIFTY)
- Expiry date
- Strike price and option type (CE/PE)
- Open, High, Low, Close, and Settlement prices
- Number of contracts traded and traded value
- Open interest and change in open interest
- Trading date (timestamp)

The dataset represents **end-of-day (EOD)** trading data and does not contain intraday timestamps.

---

3. Database Design Approach

3.1 Normalization Strategy

The raw dataset contains a high degree of redundancy, such as repeated exchange names, symbols, and expiry information across millions of rows. To eliminate redundancy and ensure data integrity, the database schema was normalized to **Third Normal Form (3NF)**.

A star schema was intentionally avoided because:

- The workload involves both ingestion and analytics
- Frequent updates favor normalized schemas
- Dimensional redundancy would significantly increase storage cost for large datasets

The normalized design ensures scalability and consistency while maintaining analytical flexibility.

---

4. Entity Design and Table Structures

4.1 Exchanges Table

The exchanges table stores metadata about the trading exchanges and enables cross-exchange analysis.

Key Columns:

- exchange_id (Primary Key)
- exchange_code (e.g., NSE, BSE, MCX)

This design allows additional exchanges to be added without modifying existing tables.

4.2 Instruments Table

The instruments table represents unique tradable instruments.

Key Columns:

- instrument_id (Primary Key)
- instrument (e.g., FUTIDX, OPTIDX)
- symbol (e.g., NIFTY, BANKNIFTY)

Separating instruments prevents repeated storage of symbol-level metadata in the high-volume trades table.

4.3 Expiries Table

The expiries table defines contract specifications for futures and options.

Key Columns:

- expiry_id (Primary Key)
- expiry_dt
- strike_pr
- option_typ (CE / PE / FUT)

This separation enables efficient handling of multiple strike prices and expiry dates reused across many trading sessions.

4.4 Trades Table

The trades table is the central fact table that stores daily trading metrics.

Key Columns:

- Foreign keys referencing exchanges, instruments, and expiries
- Open, High, Low, Close prices
- Settlement price
- Traded contracts and traded value
- Open interest and change in open interest
- Trading timestamp

This table is optimized for time-series analysis and large-scale aggregations.

---

5. Handling Real-World Data Challenges

5.1 Date and Timestamp Standardization

The raw dataset stores dates in non-ISO string formats such as 01-AUG-2019. These values were parsed using explicit date format conversion. Since the dataset represents end-of-day data, a standard market close time (15:30 IST) was appended to generate valid timestamps.

This approach ensures compatibility with time-series queries and window-based analytics.

---

6. Performance Optimization Techniques

6.1 Indexing Strategy

Indexes were created on the following columns:

- timestamp for time-range queries
- symbol for instrument-based filtering
- exchange_id for cross-exchange comparisons

These indexes significantly reduce query execution time for analytical workloads.

6.2 Query Optimization

- Window functions were used for rolling volatility calculations
- Aggregation queries were optimized to reduce unnecessary joins
- Filters on indexed columns were applied early to minimize scan volume

An EXPLAIN ANALYZE plan was used to validate query execution paths and index usage.

---

7. Scalability Considerations

The schema is designed to scale beyond 10 million rows with minimal changes:

- New exchanges can be added through the exchanges table
- The trades table can be partitioned by expiry date or exchange
- The design supports future ingestion of intraday or high-frequency trading data

---

8. Conclusion

This project demonstrates a production-ready approach to storing and analyzing large-scale financial market data. By applying normalization principles, handling real-world data inconsistencies, and optimizing analytical queries, the solution effectively supports advanced trading analytics across multiple exchanges.

The overall design closely aligns with real-world data engineering practices used in financial analytics and trading platforms.

---

9. Tools and Technologies Used

- DuckDB
- SQL
- Python
- dbdiagram.io
- Kaggle (NSE F&O Dataset)