FrontRunningRfqFX

✅ What is Front-Running in RFQ FX?

In an RFQ (Request for Quote) workflow for FX:

- A client sends a quote request to a dealer or liquidity provider.
- The dealer sees the client's intent (e.g., buy EUR/USD in large size).
- **Front-running** occurs if the dealer trades ahead of the client's order (e.g., buys EUR/USD before quoting to the client), using that information for profit.

This is considered **market abuse** and is prohibited under regulations like **MiFID II** and **Dodd-Frank**.

✅ Alert Rules

Alert rules are designed to **detect suspicious patterns** in trading behavior. Examples:

- **Pre-Quote Trading Alert**: If a dealer executes a trade in the same instrument within X seconds before responding to an RFQ.
- **Price Movement Alert**: If the dealer's quote deviates significantly from market mid-price after their own trade.
- **Volume Spike Alert**: If there's an unusual increase in dealer's own trading volume around RFQ events.

✅ Benchmark Rules

Benchmark rules compare dealer quotes against **independent market benchmarks**:

- **Quote vs Benchmark Spread**: If the dealer's quote is far from the benchmark mid-price (e.g., > 10 pips deviation).
- **Latency Benchmark**: If the dealer delays RFQ response unusually long (possibly to trade first).
- **Execution Quality Benchmark**: Compare client execution price vs market price at RFQ time.

✅ How They Work Together

- **Alert rules** flag potential misconduct in real-time.
- **Benchmark rules** provide objective reference points to validate fairness.
- Combined, they help compliance teams detect and investigate **front-running risk**.

Context

You are on the **FX compliance** team monitoring RFQ activity for potential **front-running**. A client sends an RFQ; a dealer should **quote fairly and promptly without trading ahead** based on the client's intent.

<u>Requirements for the Developer:</u>

Design and implement a PySpark-based compliance monitoring solution for RFQ FX workflows with the following requirements:

1. **Input Data Sources**
   - RFQ dataset containing: rfq_id, client, dealer, pair, rfq_time, side, size, market_mid, dealer_quote, latency_ms, client_exec_price.
   - Dealer trades dataset containing: dealer, pair, trade_time, side, size, price, source.
   - Optional benchmark feed for independent mid prices.

2. **Configuration Management**
   - All alert thresholds, risk weights, and file paths must be externalized in a **YAML configuration file**.
   - YAML should include:
     - pip_deviation_threshold
     - latency_threshold_ms
     - slippage_threshold_pips
     - pre_trade_lookback_seconds
     - Risk weights for each alert type
     - Input/output paths and toggles for CSV, Parquet, and JSON outputs.

3. **Alert Rules**
   - **Pre-trade alert**: Flag RFQs where the same dealer traded the same pair within X seconds before RFQ response.
   - **Benchmark deviation alert**: Flag RFQs where dealer quote deviates from market mid by more than pip_deviation_threshold.
   - **Latency alert**: Flag RFQs where response latency exceeds latency_threshold_ms.
   - **Execution quality alert**: Flag RFQs where slippage vs market mid exceeds slippage_threshold_pips.

4. **Risk Scoring**
   - Compute a composite risk_score using weights from YAML:

   ```
   risk_score = (pre_trade_alert * weight_pre_trade)
         + (benchmark_alert * weight_benchmark)
         + (latency_alert * weight_latency)
         + (execution_quality_alert * weight_execution_quality)
   ```

   Include a risk_reasons array listing which alerts triggered.

5. **Outputs**
   - Write enriched RFQ data with alerts and risk score to **CSV** and/or **Parquet** (controlled by YAML).
   - Additionally, **write alerts into JSON format**:
     - **Per-RFQ JSON**: Nested structure with rfq, metrics, alerts, and risk.
     - **Summary JSON**: Aggregate counts of each alert type and high-risk RFQs.

6. **Other Requirements**
   - Ensure timestamps are in UTC and ISO 8601 format.
   - Support easy threshold tuning by editing YAML without code changes.
   - Optimize for large datasets (partitioning, coalesce for single-file outputs where needed).

Sample Data:

1) rfq_fx_dataset.csv

```
rfq_id,client,dealer,pair,rfq_time,side,size,market_mid,dealer_quote,latency_ms,client_exec_price
RFQ0001,CLT003,DLR01,EURUSD,2025-12-18 09:00:15,BUY,5000000,1.08502,1.08515,120,1.08515
RFQ0002,CLT012,DLR02,USDJPY,2025-12-18 09:00:42,SELL,8000000,145.008,144.999,900,144.999
RFQ0003,CLT007,DLR03,GBPUSD,2025-12-18 09:01:10,BUY,3000000,1.26498,1.26514,1500,1.26514
RFQ0004,CLT019,DLR01,AUDUSD,2025-12-18 09:01:36,SELL,10000000,0.66510,0.66498,80,0.66498
RFQ0005,CLT001,DLR02,EURUSD,2025-12-18 09:02:05,BUY,12000000,1.08520,1.08530,1100,1.08530
```

2) dealer_trades.csv

```
dealer,pair,trade_time,side,size,price,source,rfq_id
DLR03,GBPUSD,2025-12-18 09:01:08,BUY,1800000,1.26499,DealerOwn,RFQ0003
DLR02,EURUSD,2025-12-18 09:02:04,BUY,6000000,1.08522,DealerOwn,RFQ0005
DLR01,USDJPY,2025-12-18 09:00:40,SELL,4000000,145.006,DealerOwn,RFQ0002
```

3) (Optional) market_benchmark.csv

```
pair,timestamp,mid
EURUSD,2025-12-18 09:00:15,1.08502
USDJPY,2025-12-18 09:00:42,145.008
GBPUSD,2025-12-18 09:01:10,1.26498
AUDUSD,2025-12-18 09:01:36,0.66510
EURUSD,2025-12-18 09:02:05,1.08520
```

4) (Optional) alert_config.yaml

```
# Alert thresholds & weights
thresholds:
  pip_deviation_threshold: 12       # Quote vs benchmark deviation in pips
  latency_threshold_ms: 1000        # Response latency threshold
  slippage_threshold_pips: 10       # Execution slippage threshold
  pre_trade_lookback_seconds: 5     # Window to check dealer pre-trade

risk_weights:
```

```yaml
  pre_trade: 3
  benchmark: 2
  latency: 1
  execution_quality: 2

# Paths for inputs/outputs (use DBFS/S3/ADLS/local)
paths:
  rfq_input: /tmp/rfq_fx_dataset.csv
  trades_input: /tmp/dealer_trades.csv
  benchmark_input: null       # optional; set a path if you have independent mid feed
  rfq_output: /tmp/rfq_alerts_out
  trades_output: /tmp/dealer_trades_out

write:
  csv: true
  parquet: false

# Optional runtime settings
runtime:
  timezone: UTC
```