# Strategy for Maintaining a Real-Time Reporting System

### Overview

To design a real-time reporting system for the given schema, the following key components are addressed:

- \*\*ETL Pipelines\*\*: Strategies for data ingestion, transformation, and loading.

- \*\*Query Optimization\*\*: Techniques to ensure high performance for real-time queries.

- \*\*Data Consistency\*\*: Approaches to maintain consistent and reliable metrics for dashboards.

---

### 1. ETL Pipeline Design

\*\*Goal:\*\* Efficiently process and prepare data for real-time reporting.

#### \*\*Technologies\*\*

- \*\*Apache Kafka\*\*: For real-time event streaming and data ingestion.

- \*\*Apache Flink / Spark Structured Streaming\*\*: For real-time transformation of incoming data.

- \*\*PostgreSQL\*\*: For the final storage and querying of processed data.

- \*\*Airflow\*\*: For orchestrating batch processes where needed.

#### \*\*ETL Steps\*\*

1. \*\*Ingestion\*\*:

- Use \*\*Kafka\*\* to stream events related to critical tables like `orders`, `product\_stocks`, `users`, and `notifications`.

- Create Kafka topics corresponding to key events (e.g., `order\_created`, `stock\_updated`, `user\_registered`).

2. \*\*Transformation\*\*:

- Use \*\*Apache Flink\*\* or \*\*Spark Streaming\*\* to process the streams in real-time.

- Examples of transformations:

- Enrich `orders` with user and product data from `users` and `products`.

- Calculate derived metrics like `total\_sales` and `average\_rating` on the fly.

3. \*\*Loading\*\*:

- Load aggregated data into materialized views in PostgreSQL, such as `group\_cart\_variation\_with\_price`.

- Update pre-aggregated tables for metrics like:

- Daily sales metrics.

- Product performance metrics.

#### \*\*Batch Processing for Non-Critical Data\*\*

- Use \*\*Airflow\*\* to process less frequent updates (e.g., weekly reports) by reading data from tables like `refund` or `vendor\_payments`.

---

### 2. Query Optimization

\*\*Goal:\*\* Ensure real-time dashboard queries are fast and scalable.

#### \*\*Techniques\*\*

1. \*\*Indexing\*\*:

- Create appropriate indexes on high-frequency query fields. For example:

- `orders (user\_id, created\_at)`

- `products (category\_id, price)`

- Use partial indexing for filtering commonly queried data (e.g., active users).

2. \*\*Materialized Views\*\*:

- Use materialized views like `group\_cart\_variation\_with\_price` to pre-aggregate commonly queried data.

- Schedule periodic refreshes or set up incremental updates using triggers.

3. \*\*Partitioning\*\*:

- Partition large tables like `orders` and `product\_stocks` by time (e.g., daily, monthly).

- This improves query performance for time-series analysis.

4. \*\*Caching\*\*:

- Use \*\*Redis\*\* or \*\*Memcached\*\* to cache frequent queries for dashboards.

- Example: Store the latest `top\_products` query results in Redis.

5. \*\*Query Optimization in PostgreSQL\*\*:

- Analyze query plans using `EXPLAIN` and optimize them by adjusting indexes and re-writing queries.

- Use Common Table Expressions (CTEs) for better readability and performance.

---

### 3. Data Consistency

\*\*Goal:\*\* Ensure data accuracy across the pipeline for reliable metrics.

#### \*\*Techniques\*\*

1. \*\*Transaction Handling\*\*:

- Use database transactions to ensure atomicity in updates, especially for critical tables like `orders` and `product\_stocks`.

2. \*\*Event Deduplication\*\*:

- Design Kafka consumers to handle duplicate events (e.g., using unique message keys).

3. \*\*Data Validation\*\*:

- Validate transformed data before loading into PostgreSQL using schema enforcement tools like \*\*Great Expectations\*\*.

4. \*\*CDC (Change Data Capture)\*\*:

- Use tools like \*\*Debezium\*\* to capture changes in PostgreSQL tables and reflect them in materialized views or caches.

5. \*\*Monitoring and Alerts\*\*:

- Implement monitoring using \*\*Prometheus\*\* and \*\*Grafana\*\* to track data pipeline health.

- Set up alerts for lagging Kafka consumers or delayed materialized view refreshes.

---

### Example Workflow

#### \*\*Real-Time Sales Dashboard\*\*

1. \*\*Data Sources\*\*:

- Tables: `orders`, `users`, `products`, `product\_stocks`

- Events: Order creation, stock updates

2. \*\*ETL Pipeline\*\*:

- Kafka ingests `order\_created` events.

- Flink enriches events with `users` and `products` data.

- Aggregated sales data is written to PostgreSQL materialized views.

3. \*\*Dashboard Query\*\*:

- Query pre-aggregated materialized views like:

```sql

SELECT product\_id, SUM(quantity) AS total\_sold, SUM(total\_price) AS revenue

FROM sales\_aggregates

WHERE created\_at >= NOW() - INTERVAL '1 DAY'

GROUP BY product\_id;

```

---

### Tools and Stack Summary

- \*\*Data Ingestion\*\*: Apache Kafka

- \*\*Real-Time Processing\*\*: Apache Flink or Spark Streaming

- \*\*Database\*\*: PostgreSQL

- \*\*Batch Orchestration\*\*: Apache Airflow

- \*\*Caching\*\*: Redis

- \*\*Monitoring\*\*: Prometheus + Grafana

---

By following the above strategy, you can build a robust real-time reporting system that handles high throughput, ensures data consistency, and provides fast query responses for dashboards.