Technical Document for ETL Pipeline

Overview

This document provides an overview of the ETL pipeline setup, including the structure and purpose of auditing and metadata tables, the staging process, the use of stored procedures, and data flow from ingestion to storage.

---

1. Tables for Auditing and Metadata

1.1 Source Audit Table

Schema:

```
CREATE TABLE `source_audit` (
  `source_id` int DEFAULT NULL,
  `source_name` longtext,
  `last_refresh_dt` longtext,
  `last_exec_dt` longtext,
  `tot_exec_time` double DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Purpose:

- Tracks the latest state of data sources.
- Stores information about the last refresh and execution dates.
- Monitors total execution time for incremental loads.

Columns:

Column	Data Type	Description
`source_id`	int	Identifier for the data source.
`source_name`	longtext	Name of the data source.
`last_refresh_dt`	longtext	Timestamp of the last successful refresh.

`last_exec_dt`	longtext	Timestamp of the last execution.
`tot_exec_time`	double	Total time (in seconds) taken for the execution.

1.2 Task Audit Table

Schema:

```
CREATE TABLE `task_audit` (
  `task_id` int NOT NULL AUTO_INCREMENT,
  `task_name` varchar(20) DEFAULT NULL,
  `last_exec_dt` datetime DEFAULT NULL,
  `flg_success` tinyint DEFAULT NULL,
  `exec_time` int DEFAULT NULL,
  PRIMARY KEY (`task_id`)
) ENGINE=InnoDB AUTO_INCREMENT=145 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

Purpose:

- Stores execution details of tasks in the ETL pipeline.
- Tracks execution timestamps, success flags, and execution duration.

Columns:

Column	Data Type	Description
`task_id`	int	Unique identifier for each task execution.
`task_name`	varchar(20)	Name of the task being tracked.
`last_exec_dt`	datetime	Timestamp of the task's last execution.
`flg_success`	tinyint	Flag indicating task success (1: Success, 0: Fail).
`exec_time`	int	Time (in seconds) taken for task completion.

1.3 Source Info Table

Schema:

```
CREATE TABLE `source_info` (
  `id` int NOT NULL AUTO_INCREMENT,
  `source_name` varchar(20) DEFAULT NULL,
  `last_refresh_dt` date DEFAULT NULL,
  `time_zone` varchar(49) DEFAULT NULL,
  `info` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=141 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

Purpose:

- Stores metadata about data sources.
- Tracks additional information such as the time zone and descriptive details.

Columns:

Column	Data Type	Description
`id`	int	Unique identifier for each record.
`source_name`	varchar(20)	Name of the data source.
`last_refresh_dt`	date	Date of the last refresh for the data source.
`time_zone`	varchar(49)	Time zone information for the source.
`info`	varchar(255)	Additional metadata or descriptive information.

---

2. Staging and Main Table Workflow

2.1 Staging Table

Example:

Staging table for Tesla stock data:

```
CREATE TABLE `st_stock_tsla_info` (
  `trade_dt` date NOT NULL,
  `stock_info` json NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Purpose:

- Temporary storage for raw or intermediate data fetched from external APIs.
- Enables validation and cleaning before loading into the main table.

2.2 Main Table

Example:

Main table for Tesla stock data:

```
CREATE TABLE `stock_tsla_info` (
  `trade_dt` date NOT NULL,
  `stock_info` json NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Purpose:

- Stores cleaned and transformed data for long-term use.
- Ensures data integrity and consistency for analytics and reporting.

---

3. Stored Procedure for Data Loading

Definition:

Stored procedure to load data from the staging table to the main table:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `dummy`.`sp_stock_tsla`()
BEGIN
    DELETE FROM stock_tsla_info
```

```
    WHERE trade_dt IN (
        SELECT trade_dt
        FROM st_stock_tsla_info
    );

    INSERT INTO stock_tsla_info
    SELECT trade_dt, stock_info
    FROM st_stock_tsla_info;

    COMMIT;
END;
```

Workflow:

1. **Delete Existing Records:**
    - Removes rows from the main table (`stock_tsla_info`) that match the trade dates in the staging table (`st_stock_tsla_info`).
2. **Insert New Records:**
    - Inserts data from the staging table into the main table.
3. **Commit Transaction:**
    - Ensures atomicity and consistency of the operation.

---

4. End-to-End Pipeline Flow

Step 1: Data Ingestion

- Fetch data from external APIs (e.g., stock data for Tesla).
- Load raw data into staging tables (e.g., `st_stock_tsla_info`).

Step 2: Data Transformation

- Use stored procedures to move data from staging to main tables.
- Perform deduplication and validation during the transfer.

Step 3: Auditing and Metadata Updates

- Update `source_audit` with the latest execution and refresh timestamps.
- Record task execution details in `task_audit`.
- Maintain metadata consistency in `source_info`.

---

5. Maintenance and Monitoring

- **Error Handling:**
  - Use triggers or logs to capture failures in stored procedures.
  - Record errors in the `task_audit` table.
- **Monitoring:**
  - Create dashboards to track data refresh times and task execution metrics.
  - Use database queries or third-party tools for performance tuning.

---

6. Best Practices

- **Atomicity:** Use transactions in stored procedures to ensure consistent updates.
- **Indexing:** Optimize frequently queried fields (e.g., `trade_dt` in main tables).
- **Scalability:** Partition tables or offload older data to improve query performance.
- **Documentation:** Maintain up-to-date technical documentation for easy onboarding.

This document provides a comprehensive understanding of the ETL pipeline's structure and workflow. For any additional clarifications, feel free to reach out.