OVERVIEW OF ETL PIPELINE

1. Data Extraction and Loading into MongoDB

Process:

1. **Extract JSON Data**:
    - JSON files such as `clients.json`, `suppliers.json`, `sonar_runs.json`, and `sonar_results.json` are extracted.
    - A transformation process is applied to convert special fields (`$oid`, `$date`) into MongoDB-compatible formats (e.g., ObjectIds and Date objects).
2. **Transform JSON Data**:
    - The function `transform_json_data()` recursively processes the JSON files, converting the fields like `$oid` (ObjectId) and `$date` (Date) to MongoDB-compatible formats.
3. **Load into MongoDB**:
    - Each collection (`clients`, `suppliers`, `sonar_runs`, `sonar_results`) is inserted into the respective MongoDB collection after transformation.

2. Data Extraction from MongoDB and Transformation

Process:

1. **Data Extraction**:
    - The collections are queried from MongoDB using the `extract_data_and_infer_schema()` function, which also normalizes the data to remove special MongoDB-specific fields like `$oid` and `$date`.
2. **Normalization**:
    - MongoDB documents often have complex structures. The function `normalize_mongo_data()` recursively flattens the data, converting embedded objects to simple key-value pairs and converting ObjectId and Date objects to strings and timestamps, respectively.
3. **Schema Inference**:
    - Based on the extracted data, the function `infer_schema()` automatically infers the PostgreSQL schema by examining the type of each field.

3. Schema Creation and Foreign Key Detection

Process:

1. **Schema Creation**:
    - Using the inferred schema, the pipeline dynamically creates tables in PostgreSQL using SQLAlchemy. The function `create_table()` defines columns for each table, ensuring that the `_id` field is the primary key.
2. **Foreign Key Detection**:
    - Foreign keys are automatically detected based on field names using the `detect_foreign_keys()` function. Fields ending with `_id` are automatically linked to the respective parent table, creating foreign key relationships.
3. **Handling Many-to-Many Relationships**:
    - The `sonar_runs` collection has a field `supplier_ids`, which represents a many-to-many relationship between `sonar_runs` and `suppliers`. Instead of storing this data in `sonar_runs`, a separate association table `sonar_runs_suppliers` is created to handle this relationship.

4. Loading Data into PostgreSQL

Process:

1. **Data Loading**:
    - Data from MongoDB collections is normalized, and the corresponding PostgreSQL tables are populated. Fields that involve many-to-many relationships (e.g., `supplier_ids`) are handled separately by inserting data into the association table.
2. **Index Creation**:
    - To optimize query performance, indexes are added to key columns (e.g., `_id`, `client_id`, `part_id`, `sonar_run_id`). This ensures that the most frequent queries can be executed efficiently.

5. SQL File Execution (ALTER and Views)

Process:

1. **SQL Scripts for Table Alterations**:
    - After the initial data load, additional SQL commands are executed to perform necessary adjustments to the schema, such as adding foreign key constraints and removing unnecessary fields (e.g., `supplier_ids`).
    - Missing suppliers (those not found in `suppliers` table) are temporarily added.
2. **View Creation**:
    - Several views are created to support analytical queries. These views aggregate data by part, supplier, country, and track price development over time.

6. Error Handling and Retry Logic

Process:

1. **Retry Mechanism**:
    - The pipeline uses a retry mechanism for resilience, ensuring that transient failures do not cause the pipeline to fail permanently. If a function fails (e.g., due to a database timeout), it will retry up to 3 times before raising a critical error.

7. Final Optimizations

1. Indexing:
    - Indexes are added to the most commonly queried fields to ensure fast data retrieval.
2. Data Normalization and Transformation:
    - All embedded data structures and complex fields from MongoDB are flattened and converted to a format suitable for PostgreSQL.
3. Foreign Key Constraints:
    - Foreign key constraints ensure data integrity and relational consistency between the various tables, especially with the many-to-many relationship between `sonar_runs` and `suppliers`.
4. View Creation for Analytical Queries:
    - Views are created to facilitate common analytical queries like:
        - Results per part and shop.
        - Results per country.
        - Price development per part over time.

8. ERD Diagram of Relational Database