### Introduction to Data Vault Modeling Approach:

The Data Vault modeling approach is a methodology designed for creating data warehouses with a focus on flexibility, scalability, and auditability. It comprises several key concepts:

#### **1. Hub Tables:**

* Represent core business entities (customers, products).
* Store unique business keys.
* Typically, simple with one primary key column.

#### **2. Satellite Tables:**

* Capture detailed attributes and historical changes associated with hub entities.
* Linked to hub tables.
* Include metadata like load dates and source system identifiers.

#### **3. Link Tables:**

* Establish relationships between hub entities or between hubs and satellites.
* Represent connections or associations between different business entities.

#### **4. Surrogate Keys:**

* Artificially generated unique identifiers for records in hub tables.
* Simplify data integration and querying.

#### **5. Slowly Changing Dimensions (SCDs):**

* Techniques for managing changes to dimension attributes over time.
* Data Vault supports various SCD types like Type 1 (overwrite), Type 2 (add new row), and Type 3 (add new column).

### Tasks to Complete Part 2:

#### **1. Design SQL Scripts for Data Mart:**

* Define structure using Data Vault modeling approach.
* Create hub tables for customers, products, satellite tables for historical changes, and link tables for associations.
* Ensure data integrity with primary and foreign key constraints.

#### **2. Data Mart Population:**

* Populate with diverse sample data.
* Use realistic data generation techniques.

#### **3. Incremental Loading:**

* Support loading only new records during updates.
* Implement change detection mechanisms.
* Ensure efficiency and data consistency.

#### **4. Surrogate Keys:**

* Assign unique identifiers to hub tables.
* Simplify data integration and referencing.

#### **5. Slowly Changing Dimensions (SCDs):**

* Extend satellite tables to support various SCD types.
* Track historical changes to attributes.

#### **6. Partitioning and Compression:**

* Optimize storage with partitioning and compression techniques.
* Improve query performance and reduce storage requirements.

#### **7. Error Handling and Logging:**

* Handle data validation errors.
* Log comprehensive information for monitoring and troubleshooting.

#### **8. GitHub Integration:**

* Utilize GitHub for version control.
* Establish branching strategy, pull requests, and issue tracking for collaborative development.

### SQL Script Design for Data Mart:

#### **Hub Tables:**

**customers\_hub Table:**

-- Create customers\_hub table with hash key

CREATE TABLE customers\_hub (

    customer\_hash\_key CHAR(32) PRIMARY KEY DEFAULT MD5(customer\_id::TEXT),

    customer\_id SERIAL

);

Explanation:

* **customer\_hash\_key**: Primary key using MD5 hash of **customer\_id**.
* **customer\_id**: Unique identifier for customers (auto-incrementing serial).

**products\_hub Table:**

-- Create products\_hub table with hash key

CREATE TABLE products\_hub (

    product\_hash\_key CHAR(32) PRIMARY KEY DEFAULT MD5(product\_id::TEXT),

    product\_id SERIAL

);

Explanation:

* **product\_hash\_key**: Primary key using MD5 hash of **product\_id**.
* **product\_id**: Unique identifier for products (auto-incrementing serial).

#### **Satellite Tables:**

**customers\_satellite Table:**

-- Create customers\_satellite table with hash difference and related fields

CREATE TABLE customers\_satellite (

    customer\_hash\_key CHAR(32) REFERENCES customers\_hub(customer\_hash\_key),

    hash\_diff CHAR(32) DEFAULT MD5(customer\_name || customer\_email || new.customer\_address) GENERATED ALWAYS AS (MD5(customer\_name || customer\_email || customer\_address)) STORED,

    customer\_name VARCHAR(100),

    customer\_email VARCHAR(100),

    customer\_address VARCHAR(255),

    start\_date DATE,

Explanation:

* **customer\_hash\_key**: Foreign key referencing **customer\_hash\_key** in **customers\_hub**.
* **hash\_diff**: Hash difference for tracking changes in customer attributes.
* Other fields: Customer attributes, metadata, and tracking fields.

**products\_satellite Table:**

-- Create products\_satellite table with hash difference and related fields

CREATE TABLE products\_satellite (

    product\_hash\_key CHAR(32) REFERENCES products\_hub(product\_hash\_key),

    hash\_diff CHAR(32) DEFAULT MD5(product\_name || product\_category || product\_brand) GENERATED ALWAYS AS (MD5(product\_name || product\_category || new.product\_brand)) STORED,

    product\_name VARCHAR(100),

    product\_category VARCHAR(50),

    product\_brand VARCHAR(50),

    start\_date DATE,

    end\_date DATE,

    load\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    source VARCHAR(50)

);

Explanation:

* **product\_hash\_key**: Foreign key referencing **product\_hash\_key** in **products\_hub**.
* **hash\_diff**: Hash difference for tracking changes in product attributes.
* Other fields: Product attributes, metadata, and tracking fields.

**sales\_link Table:**

-- Create sales\_link table with hash keys referencing customers\_hub and products\_hub

CREATE TABLE sales\_link (

    transaction\_hash\_key CHAR(32) PRIMARY KEY DEFAULT MD5(CONCAT(customer\_id::TEXT, product\_id::TEXT)),

    customer\_hash\_key CHAR(32) REFERENCES customers\_hub(customer\_hash\_key),

    product\_hash\_key CHAR(32) REFERENCES products\_hub(product\_hash\_key),

    transaction\_date DATE,

    transaction\_amount NUMERIC(10, 2),

    load\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    source VARCHAR(50)

);

Explanation:

* **transaction\_hash\_key**: Primary key using MD5 hash of concatenated **customer\_id** and **product\_id**.
* Foreign key references to hub tables, transaction attributes, and metadata.

**sales\_transactions\_satellite Table:**

CREATE TABLE sales\_transactions\_satellite (

    transaction\_hash\_key CHAR(32) PRIMARY KEY DEFAULT MD5(CONCAT(customer\_id::TEXT, product\_id::TEXT)),

    start\_date DATE,

    end\_date DATE,

    load\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    source VARCHAR(50),

    hash\_diff CHAR(32) DEFAULT MD5(load\_date || source) GENERATED ALWAYS AS (MD5(load\_date || source)) STORED,

    CONSTRAINT fk\_sales\_link FOREIGN KEY (transaction\_hash\_key) REFERENCES sales\_link(transaction\_hash\_key)

);

Explanation:

* Additional metadata and contextual information related to sales transactions.
* Directly linked to sales\_link table for streamlined association.
* Hash\_diff field for data integrity validation and change tracking.

### Reasoning for Decisions:

* **Data Integrity:** Utilizing hash keys ensures data integrity by providing unique, stable identifiers that resist manipulation.
* **Efficiency:** Hash keys and hash difference fields enable efficient change detection and historical tracking, supporting incremental loading.
* **Referential Integrity:** Foreign key constraints maintain referential integrity between tables, ensuring valid associations.
* **Auditing and Tracking:** Metadata fields like load\_date and source enable comprehensive auditing and tracking of data loading activities.

These design decisions aim to optimize data modeling for the data mart, ensuring a balance between data integrity, efficiency, and usability considerations.

Note: recently, the standard practice is beginning to shift towards using hash keys instead of sequence values to assign surrogate keys, particularly within the data vault 2.0 approach to data warehousing. A hash key is the output from a hashing algorithm, where a specific input value is transformed into a distinct, unique string per input value.

### The conceptual schema for data vault:

### 

* 1. **Data Generation Process**:
  + First, mock customer data is generated using the **generate\_customer\_data** function, creating a list of lists where each inner list represents a customer record. This data is then written to a CSV file named **customer\_data.csv**.
  + Next, mock product data is generated using the **generate\_product\_data** function, creating a list of lists where each inner list represents a product record. This data is then written to a CSV file named **product\_data.csv**.
  + Then, mock sales data is generated using the **generate\_sales\_data** function. This data includes random customer and product IDs from the previously generated data. It creates a list of lists where each inner list represents a sales transaction record. This data is then written to a CSV file named **sales\_data.csv**.

## Load Script Documentation

The script begins by importing necessary modules and functions:

* **logging**: Python's built-in logging module for logging information during script execution.
* **csv**: Module for reading data from CSV files.
* **load\_config**, **connect\_to\_database**, **close\_connection**: Custom functions from the **database\_utils** module for database operations.
* **hashlib**: Module for generating hash keys.
* **datetime**: Module for working with dates and times.
* **Faker**: Module for generating fake data.

### Faker Initialization

Initializes a Faker generator to generate fake data when necessary.

### Logging Configuration

Configures logging to write logs to a file named **data\_vault.log** with the format including the timestamp, log level, and message.

### Hash Key Generation Functions

* **generate\_hash\_key**: Function to generate a unique hash key for each record using the MD5 hashing algorithm.
* **generate\_concat\_hash**: Function to concatenate fields and generate a hash difference, useful for tracking changes to the data over time.

### Function: insert\_data\_from\_csv

This function is responsible for inserting data from a CSV file into the appropriate tables in the database based on the specified **table\_name**.

#### **Parameters:**

* **csv\_file**: Path to the CSV file containing the data to be inserted.
* **table\_name**: Name of the table in the database where the data should be inserted.
* **conn**: Database connection object.

#### **Process:**

1. The function begins by attempting to establish a cursor for executing SQL queries on the database.
2. It then opens the specified CSV file in read mode and creates a CSV reader object to iterate over the rows.
3. For each row in the CSV file:
   * It checks the **table\_name** to determine which table the data should be inserted into.
   * For **products\_hub** table:
     + Generates a unique **product\_hash\_key** using the **generate\_hash\_key** function based on the **product\_id**.
     + Constructs an SQL **INSERT** query to insert the data into the **products\_hub** table.
     + Retrieves additional product details (name, category, brand) from the row.
     + Sets the **start\_date** to the current date/time and **end\_date** to **None** to indicate current validity.
     + Calculates the **hash\_diff** based on the concatenated product details.
     + Updates the end date of existing records in the **products\_satellite** table, if necessary.
     + Inserts the new record into the **products\_satellite** table.
   * For **customers\_hub** table:
     + Generates a unique **customer\_hash\_key** using the **generate\_hash\_key** function based on the **customer\_id**.
     + Constructs an SQL **INSERT** query to insert the data into the **customers\_hub** table.
     + Retrieves additional customer details (name, email, address) from the row.
     + Sets the **start\_date** to the current date/time and **end\_date** to **None**.
     + Calculates the **hash\_diff** based on the concatenated customer details.
     + Updates the end date of existing records in the **customers\_satellite** table, if necessary.
     + Inserts the new record into the **customers\_satellite** table.
   * For **sales\_link** table:
     + Generates a unique **transaction\_hash\_key** using the **generate\_hash\_key** function based on the **transaction\_id**.
     + Retrieves **customer\_hash\_key** and **product\_hash\_key** by querying the respective tables.
     + Constructs an SQL **INSERT** query to insert the data into the **sales\_link** table.
     + Sets the **start\_date** to the current date/time and **end\_date** to **None**.
     + Calculates the **hash\_diff** based on the transaction date and source.
     + Inserts the new record into the **sales\_transactions\_satellite** table.
4. Commits the transaction to persist the changes in the database and closes the cursor.
5. Logs a success message indicating that the data insertion was successful.

#### **Error Handling:**

* If any exception occurs during the execution of the function, it logs an error message and prints the error.
* The **try-except** block ensures that the database connection is closed properly even if an error occurs during the execution of the function.

**Main Execution Block:**

* Loads the configuration from the **config.yaml** file, establishes a connection to the database, and inserts data into the **products\_hub**, **customers\_hub**, and **sales\_link** tables using the **insert\_data\_from\_csv** function.
* Closes the database connection after data insertion.
* Handles any exceptions that occur during the execution and logs/print the error messages.