From Key-Value to Relational: Bridging Semi-Structured and Tabular Worlds

2025-03-14

Contents

1 Key-Value Pair Handling in Relational World

Key-value pair handling is a challenge in modern data architecture, particularly when dealing with semi-structured data sources such as JSON, NoSQL databases, telemetry logs, and configuration files. We want to explore various approaches to managing and querying key-value data, analyzing their strengths, limitations, and optimal use cases.

While key-value pairs are more naturally suited to NoSQL and document-based databases due to their schema-flexibility and performance characteristics, there are scenarios where relational systems are mandated due to architectural, integration, or regulatory constraints. In many enterprises, upstream or downstream systems may depend on SQL-based tooling for data access, governance, or analytics. As a result, engineers must translate semi-structured or dynamic key-value datasets into a relationally compliant form. This introduces challenges related to normalization, indexing, query efficiency, and schema evolution—areas where relational databases are traditionally optimized for static and well-defined schemas.

There are primarily two recognized approaches to modeling key-value data within relational systems: the Entity-Attribute-Value (EAV) model and the JSON/VARIANT model. The EAV model decomposes data into a triplet form—entity, attribute, and value—allowing for the representation of sparse and heterogeneous attributes. It provides high normalization, making it suitable for metadata-heavy applications where attribute sets differ significantly between entities. However, this model often complicates query logic, requires complex pivoting for reporting, and may introduce performance overhead without proper indexing or clustering. Moreover, the EAV approach typically offloads type enforcement and integrity checks to the application layer or stored procedures, increasing the burden on data engineers and developers.

An alternative approach, supported by modern relational engines like Snowflake and PostgreSQL, is to store key-value pairs using semi-structured data types such as JSON or VARIANT within a single column. This strategy defers schema enforcement while retaining the ability to query deeply nested data using path expressions and cast values into appropriate types at runtime. It also enables flexible and agile ingestion pipelines without the need to continuously modify the underlying table schema. By leveraging features like materialized views, lateral flattening, and automatic clustering, systems can balance flexibility with query performance. However, this method comes with trade-offs: while flexible, JSON-based structures can lead to reduced transparency for governance teams, inconsistent schemas across records, and less efficient use of query optimizers unless handled carefully.

Ultimately, choosing a key-value modeling strategy in a relational system requires a deliberate evaluation of trade-offs across multiple dimensions—query complexity, schema volatility, governance, tooling compatibility, and operational performance. The EAV model is often favored in use cases requiring high data integrity and where metadata models evolve slowly but must remain auditable and queryable in normalized form. JSON/VARIANT storage, on the other hand, excels in environments that prioritize rapid evolution, minimal ingestion friction, and flexibility in application-layer logic. In practice, hybrid models are increasingly common—using EAV for core reference data and JSON for peripheral or vendor-defined metadata—allowing organizations to achieve the benefits of both normalization and agility. Below we discuss this in more details:

- **Entity-Attribute-Value (EAV) Model**
  - Transforms key-value pairs into a normalized relational structure
  - Uses three columns: entity, attribute, and value
  - Ideal for sparse data with many optional attributes
  - Provides strong data integrity and flexibility
  - Best suited for metadata-heavy applications
  - As an example consider a Patient Medical Records System that each patient's data is stored in a highly normalized format where every data point is decomposed into a triple: the patient's identifier (entity), a specific medical attribute, and its corresponding value. This

allows the system to accommodate a wide range of patient-specific data—such as blood pressure, heart rate, diagnosis codes, temperature, and allergies—without requiring a fixed schema or sparsely populated columns.

  - EAV model is particularly advantageous where the nature and number of attributes can vary greatly. By representing data in this flexible structure, the database can support extensibility and ensure data integrity while avoiding schema modifications for new or rare attributes. This is good for scenarios involving complex, semi-structured, or sparse datasets, and it aligns with the needs of metadata-heavy applications.

- **JSON/VARIANT Storage**
  - Stores key-value pairs directly as semi-structured data
  - Leverages native JSON support in modern databases
  - Offers maximum flexibility for schema evolution
  - Enables complex nested data structures
  - Perfect for rapid prototyping and dynamic data
  - Example: **JSON Storage** can be used in e-commerce platform where product specifications vary widely across categories, and sometime even unknown. Instead of predefining columns for every possible attribute, each product record includes a `specifications` column that stores a JSON object. For instance, a laptop might have `{ "RAM": "32 GB", "CPU": "Intel i9", "Storage": "512GB NVMe" }`, while a T-shirt might store `{ "Size": "L", "Color": "Black", "Material": "Cotton" }`.
  - This semi-structured format allows the database to flexibly accommodate heterogeneous and evolving data schemas without altering table definitions, making it ideal for systems where new product types and attributes are frequently introduced. JSON-based storage is especially useful for rapid development and data models that benefit from complex nesting or dynamic attribute sets.
  - To improve parsing, validation, and interoperability, a `schema` field can be embedded or referenced within the JSON document, similar to standards like JSON-LD. This enables downstream systems to understand the structure and semantics of the data, facilitating automation and integration in data pipelines and analytics workflows.

- **Hybrid Schema Design**
  - Combines structured columns with flexible storage
  - Uses fixed columns for common attributes
  - Stores variable attributes in JSON/VARIANT columns
  - Balances performance and flexibility
  - Ideal for applications with mixed data patterns

- **Dynamic Schema Management**
  - Implements metadata-driven schema evolution
  - Maintains schema registry and validation rules
  - Enables controlled schema changes

- Provides versioning and migration support
- Best for enterprise applications requiring strict governance
- Example: A large financial institution managing customer onboarding workflows across multiple jurisdictions. To support evolving regulatory requirements, each jurisdiction may introduce new compliance fields—such as tax residency, beneficial ownership, or risk score—that must be incorporated into the data model. By maintaining a centralized schema registry with versioning, validation rules, and migration logic, the system can validate incoming data against the current schema version, trigger automated migrations when the schema changes, and ensure backward compatibility for legacy data. This metadata-driven approach supports controlled schema evolution while preserving auditability and governance, which is essential in regulated industries such as banking, healthcare, and insurance.

The following sections explore each approach in detail, providing implementation examples, performance considerations, and best practices for choosing the right strategy for your use case.

2 Sample Implementation for Entity-Attribute-Value (EAV) Model

Below Snowflake SQL script implements a secure and flexible EAV data model. It begins by defining a metadata tag called `data_sensitivity` used to classify data for privacy and compliance. It then creates two supporting lookup tables: one for tenant-level access control (`tenant_access_control`) that maps user roles to tenant IDs, and one for identifying sensitive attributes (`sensitive_attributes`) which will later be used in dynamic data masking policies. These lookup tables are seeded with sample data to demonstrate role mappings and attribute sensitivity.

We also create the core EAV schema consisting of three main tables: `attributes`, `attribute_values`, and `attribute_value_history`. The `attributes` table stores metadata about dynamic attributes such as name, type, version, and optional schema definitions. The `attribute_values` table captures the actual values associated with entities, supporting multi-tenancy through a `tenant_id` column and using the `VARIANT` type for flexible storage. The `attribute_value_history` table maintains an audit trail of changes to attribute values, including timestamps and user identifiers.

To improve usability and query efficiency, a view called `vw_attribute_values` is created, joining attribute values with their metadata for easier analytical access. The tables are clustered by commonly queried fields to optimize performance. Governance controls are enforced through a row-level access policy (`eav_row_policy`) which restricts access based on the tenant and the user's role, and a masking policy (`mask_sensitive_values`) that redacts data for sensitive attributes unless the user has a privileged role. This masking policy uses a subquery against the `sensitive_attributes` table, ensuring that sensitive data is only exposed to authorized users.

Lastly, we include tagging commands to annotate sensitive tables for data cataloging purposes, and we create a `SECURE VIEW` called `vw_safe_attributes` that excludes complex data types like JSON or arrays to provide safer data access for analysts. It concludes by defining an optional external function, `log_audit_event`, which enables integration with an external API (such as a security monitoring system) to log audit events. This allows for observability and compliance auditing when data is accessed or modified, aligning the model with enterprise-grade data governance practices.

```
-- ======================================
-- Tag Definitions (required before usage)
-- ======================================

CREATE OR REPLACE TAG data_sensitivity STRING COMMENT = 'Data classification level';

-- ======================================
```

```sql
-- Supporting Lookup Tables
-- ========================================

-- Role-based tenant isolation
CREATE OR REPLACE TABLE tenant_access_control (
    tenant_id NUMBER,
    role_name STRING
);

-- Sample data
INSERT INTO tenant_access_control (tenant_id, role_name) VALUES
(101, 'ANALYST_TENANT_101'),
(101, 'DATA_ENGINEER'),
(102, 'ANALYST_TENANT_102'),
(102, 'COMPLIANCE_OFFICER');

-- Attributes considered sensitive (used in masking policy)
CREATE OR REPLACE TABLE sensitive_attributes (
    attribute_id NUMBER
);

-- Sample data (attribute IDs must match IDs in `attributes` table)
INSERT INTO sensitive_attributes VALUES (1), (2), (3);

-- ========================================
-- Core Schema for EAV Model
-- ========================================

CREATE OR REPLACE TABLE attributes (
    attribute_id NUMBER AUTOINCREMENT PRIMARY KEY,
    name STRING NOT NULL,
    data_type STRING NOT NULL CHECK (data_type IN ('STRING', 'NUMBER', 'BOOLEAN', 'DATE', 'OBJECT', 'ARRAY')),
    attribute_group STRING,
    is_required BOOLEAN DEFAULT FALSE,
    version INTEGER DEFAULT 1,
    schema_definition OBJECT,
    created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP()
);

CREATE OR REPLACE TABLE attribute_values (
    value_id NUMBER AUTOINCREMENT PRIMARY KEY,
    entity_id NUMBER NOT NULL,
```

```
    attribute_id NUMBER NOT NULL,
    value VARIANT,
    tenant_id NUMBER NOT NULL,
    created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    FOREIGN KEY (attribute_id) REFERENCES attributes(attribute_id)
);

CREATE OR REPLACE TABLE attribute_value_history (
    history_id NUMBER AUTOINCREMENT PRIMARY KEY,
    value_id NUMBER NOT NULL,
    entity_id NUMBER NOT NULL,
    attribute_id NUMBER NOT NULL,
    old_value VARIANT,
    new_value VARIANT,
    changed_by STRING,
    changed_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    FOREIGN KEY (attribute_id) REFERENCES attributes(attribute_id)
);

-- ========================================
-- View for Analytical Querying
-- ========================================

CREATE OR REPLACE VIEW vw_attribute_values AS
SELECT
    av.entity_id,
    a.name AS attribute_name,
    a.attribute_group,
    a.data_type,
    a.version,
    av.value,
    av.created_at,
    av.updated_at
FROM attribute_values av
JOIN attributes a ON av.attribute_id = a.attribute_id;

-- ========================================
-- Performance Optimizations
-- ========================================

ALTER TABLE attribute_values
    CLUSTER BY (entity_id, attribute_id);
```

```
ALTER TABLE attribute_value_history
    CLUSTER BY (entity_id, changed_at);

-- ========================================
-- Governance Hooks and Policies
-- ========================================

-- Row access policy for tenant isolation
CREATE OR REPLACE ROW ACCESS POLICY eav_row_policy
AS (tenant_id_column NUMBER)
RETURNS BOOLEAN ->
    EXISTS (
        SELECT 1 FROM tenant_access_control
        WHERE tenant_id = tenant_id_column
        AND CURRENT_ROLE() = role_name
    );

ALTER TABLE attribute_values
    ADD ROW ACCESS POLICY eav_row_policy
    ON (tenant_id);

-- Masking policy using lookup table (Snowflake compliant)
CREATE OR REPLACE MASKING POLICY mask_sensitive_values
AS (val VARIANT, attr_id NUMBER)
RETURNS VARIANT ->
    CASE
        WHEN EXISTS (
            SELECT 1 FROM sensitive_attributes
            WHERE attribute_id = attr_id
        )
        AND CURRENT_ROLE() NOT IN ('DATA_ENGINEER', 'COMPLIANCE_OFFICER')
        THEN '***REDACTED***'
        ELSE val
    END;

ALTER TABLE attribute_values
    MODIFY COLUMN value
    SET MASKING POLICY mask_sensitive_values
    USING (value, attribute_id);

-- ========================================
-- Metadata Tagging (for Cataloging/Classification)
```

```
-- ========================================

ALTER TABLE attribute_values
    SET TAG data_sensitivity = 'PII';

ALTER TABLE attribute_value_history
    SET TAG data_sensitivity = 'PII_AUDIT';

-- ========================================
-- Secure View for Controlled Access
-- ========================================

CREATE OR REPLACE SECURE VIEW public.vw_safe_attributes AS
SELECT
    av.entity_id,
    a.name AS attribute_name,
    av.value
FROM attribute_values av
JOIN attributes a ON av.attribute_id = a.attribute_id
WHERE a.data_type NOT IN ('OBJECT', 'ARRAY');

-- ========================================
-- Optional Logging/Auditing via External Functions
-- ========================================

-- API Integration must be created by account admin
-- CREATE OR REPLACE API INTEGRATION audit_log_integration
--   API_PROVIDER = aws_api_gateway
--   ENABLED = TRUE
--   API_AWS_ROLE_ARN = 'arn:aws:iam::<account>:role/snowflake_audit_logger'
--   API_ALLOWED_PREFIXES = ('https://<your-gateway>.execute-api.us-east-1.amazonaws.com/prod/')
--   COMMENT = 'Audit integration for attribute-level logging';

CREATE OR REPLACE EXTERNAL FUNCTION log_audit_event(
    event_type STRING,
    table_name STRING,
    entity_id NUMBER,
    attribute_id NUMBER,
    user_name STRING,
    timestamp TIMESTAMP_NTZ
)
RETURNS STRING
API_INTEGRATION = audit_log_integration
```

```
HEADERS = ( 'x-api-key' = '<your_api_key>' )
URL = 'https://<your-gateway>.execute-api.us-east-1.amazonaws.com/prod/log_event';

-- Example usage:
-- SELECT log_audit_event('ATTRIBUTE_UPDATED', 'attribute_values', 123, 456, CURRENT_USER(), CURRENT_TIMESTAMP());
```

3 Sample Implementation for JSON/VARIANT Storage Model

Below we quickly discuss how to use Snowflake's native support for semi-structured data through the `VARIANT` type, enabling flexible storage of JSON-like documents without predefined schema. A table named `json_storage` is created with two timestamp columns (`created_at` and `updated_at`) and a central `data` column of type `VARIANT`, which holds arbitrary key-value pairs or nested JSON objects. This design is ideal for scenarios where incoming data structures may differ between rows, such as product catalogs, IoT telemetry, or event logs.

To support analysis and consumption by downstream tools, a view named `vw_json_storage` is created to expose structured fields from the `data` column using dot notation and explicit type casting. The view extracts fields such as `color`, `size`, `price`, and `category` as scalar values, making them directly accessible for SQL-based analytics or business intelligence tools. By doing so, it combines the flexibility of schema-less storage with the readability of traditional relational outputs.

A sample query is provided that demonstrates how to access deeply nested JSON fields using path expressions. In this case, dimensions like `height` and `width` are accessed from the nested object `data:specifications.dimensions`. The query also includes filtering logic on both top-level and nested keys using type casting, ensuring that numeric comparisons (e.g., prices greater than 100) behave correctly even when sourced from untyped JSON input. This approach supports rich analytical queries without requiring schema flattening or upfront transformations.

Overall, this model is effective for ingesting and exploring heterogeneous or evolving datasets where schema enforcement is either impractical or premature. It supports rapid prototyping, agile ingestion pipelines, and heterogeneous application domains while preserving Snowflake's performance optimizations. Developers and data engineers can dynamically parse fields on-demand, and views help standardize access patterns for data consumers without rigid schemas.

```
CREATE OR REPLACE TABLE json_storage (
    id NUMBER AUTOINCREMENT,
    data VARIANT,
    created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP
);

-- Create a secure view with type casting
CREATE OR REPLACE VIEW vw_json_storage AS
SELECT
    id,
    data:color::STRING as color,
    data:size::NUMBER as size,
    data:price::FLOAT as price,
    data:category::STRING as category,
    created_at,
```

```
    updated_at
FROM json_storage;

-- Query example with proper type casting
SELECT
    id,
    data:color::STRING as color,
    data:size::NUMBER as size,
    data:price::FLOAT as price,
    data:specifications.dimensions.height::FLOAT as height,
    data:specifications.dimensions.width::FLOAT as width
FROM json_storage
WHERE data:category::STRING = 'electronics'
  AND data:price::FLOAT > 100;
```

4 Practical Hybrid Example: Structuring Question and Answer Pairs in Relational World

Below we discuss structuring a series of Question and Answer pairs into relational world. The QA pairs demonstrates the practical application of various key-value storage approaches we discussed, showcasing the flexibility and efficiency of hybrid and dynamic schema design management. The system's implementation includes data validation, performance optimization, and data migration capabilities, ensuring a reliable and scalable data management model.

The QA pairs management system exemplifies the hybrid schema design approach by combining structured columns for core data with flexible storage for metadata. The primary table structure uses fixed columns for essential information such as question and answer text, while leveraging Snowflake's VARIANT data type for metadata and ARRAY for tags. This design provides a balance between performance and flexibility, allowing the system to store common attributes efficiently while accommodating variable or evolving data points without schema modifications. The implementation demonstrates how modern databases can bridge the gap between structured and semi-structured data models, offering the best of both worlds.

The system's function definitions showcase data manipulation capabilities that leverage Snowflake's advanced SQL features. The CRUD operations are implemented through stored procedures that handle data validation, error checking, and proper timestamp management. Search functionality is enhanced through specialized functions that perform keyword matching across both structured and semi-structured data, demonstrating how to efficiently query hybrid schemas. Date range queries utilize Snowflake's powerful date functions to filter records based on temporal criteria, while statistical analysis functions aggregate data across various dimensions to provide insights into the QA pair collection.

Performance optimization is a key focus of the implementation, with several techniques employed to ensure efficient data access. The system uses clustering keys on timestamp columns to improve query performance for time-based operations, which is particularly important for retrieving recent QA pairs. Materialized views are created for common query patterns, pre-computing and storing results to reduce runtime overhead. The implementation also includes search-optimized views that extract commonly accessed attributes from the VARIANT columns, providing a more efficient interface for applications that frequently access specific metadata fields.

Data validation and integrity are maintained through a approach that combines schema-level constraints with application-level validation. The system enforces data quality through NOT NULL constraints on essential fields, while the VARIANT and ARRAY types provide flexibility for optional or variable data. The implementation includes functions for validating data against predefined schemas, ensuring that metadata conforms to expected

formats and types. This multi-layered validation approach ensures data integrity while maintaining the flexibility needed for a dynamic QA pair management system.

The system's architecture demonstrates effective separation of concerns through a modular design that separates database objects, functions, and demo scripts. The create_objects.sql file establishes the foundational database structure, while cleanup.sql provides a mechanism for resetting the environment. Sample data population is handled through dedicated scripts that showcase various data patterns and use cases. This modular approach facilitates maintenance, testing, and evolution of the system over time, allowing developers to modify specific components without affecting the entire architecture.

Advanced features such as time travel and zero-copy cloning are leveraged to enhance the system's capabilities. Time travel enables point-in-time recovery of QA pairs, providing a safety net for data modifications and supporting audit requirements. The implementation includes procedures for recovering specific records from historical snapshots, demonstrating how to leverage Snowflake's time travel capabilities for data management. Zero-copy cloning is used to create development and testing environments, allowing teams to work with production-like data without duplicating storage or compromising data security.

The demo files provide practical examples of how to interact with the system, showcasing various query patterns and use cases. These examples demonstrate how to retrieve all QA pairs, search for specific content, filter by date ranges, generate statistics, update existing records, and safely delete entries. The execution log captures the successful implementation of these operations, providing a real-world demonstration of the system in action. This documentation and example set ensures that developers can quickly understand and utilize the system's capabilities, facilitating adoption and extension of the key-value storage approaches demonstrated in this implementation.

4.1 Database Schema Implementation

Our Question-Answer pair management example implements a streamlined schema design focused on efficient data storage and retrieval. Below we discuss the detailed SQL scripts used. We start by defining a simple system for managing a collection of question-and-answer (QA) pairs, useful in contexts such as FAQ systems, chatbot datasets, or searchable knowledge bases. The core table, `daily_qa_pairs`, holds an `id` for each record along with the `question`, `answer`, and two timestamp fields: `created_at` and `updated_at`. Although the `id` column is declared as a standard `NUMBER`, it is designed to be populated using a separately defined sequence to ensure auto-increment behavior.

To manage the generation of unique identifiers, a sequence called `daily_qa_pairs_seq` is created. This sequence starts at 1 and increments by 1 with each call to `NEXTVAL`, allowing users or application logic to insert new rows into the QA table with consistent, unique IDs. This approach is commonly used in systems like Snowflake, where sequences offer a more flexible and controlled alternative to in-line auto-increment fields. By decoupling the ID generation logic from the table schema, developers maintain more precise control over concurrency and transaction consistency during batch inserts.

The script also defines a user-defined SQL function, `search_qa_pairs`, which enables full-text search capabilities over both the question and answer fields. It accepts a single input parameter, `keyword`, and performs a case-insensitive search using the `LOWER(...)` and `LIKE` operators. The function returns a result set of matching QA records ordered by `created_at` in descending order, ensuring that the most recent relevant entries are prioritized. This abstraction simplifies querying for analysts and application developers, encapsulating search logic in a reusable and maintainable interface.

```
-- Main table for QA pairs
CREATE OR REPLACE TABLE daily_qa_pairs (
  id NUMBER, -- Auto-incrementing ID
  question TEXT, -- The question text
```

```
  answer TEXT, -- The answer text
  created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(), -- Creation timestamp
  updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP() -- Last update timestamp
);

-- Create sequence for auto-incrementing IDs
CREATE OR REPLACE SEQUENCE daily_qa_pairs_seq START = 1 INCREMENT = 1;

-- Example of search function
CREATE OR REPLACE FUNCTION search_qa_pairs(keyword TEXT)
RETURNS TABLE (
  id NUMBER,
  question TEXT,
  answer TEXT,
  created_at TIMESTAMP_NTZ,
  updated_at TIMESTAMP_NTZ
  )
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
  'SELECT id, question, answer, created_at, updated_at
  FROM daily_qa_pairs
  WHERE
  LOWER(question) LIKE ''%'' || LOWER(keyword) || ''%'' OR
  LOWER(answer) LIKE ''%'' || LOWER(keyword) || ''%''
  ORDER BY created_at DESC';
```

4.2 Database Setup and Cleanup

Here is how we begin with database setup files that create and manage the necessary objects:

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

-- Create sequence for auto-incrementing IDs
CREATE OR REPLACE SEQUENCE daily_qa_pairs_seq START = 1 INCREMENT = 1;

-- Create the table with timestamp management
CREATE OR REPLACE TABLE daily_qa_pairs (
    id NUMBER,                    -- Auto-incrementing ID
    question TEXT,                -- The question text
    answer TEXT,                  -- The answer text
    created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),  -- Creation timestamp
```

```
    updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP()   -- Last update timestamp
);

SELECT 'Database objects created successfully' as status;
```

The cleanup script ensures proper removal of all objects:

```
-- Use the correct context
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

-- Drop all functions and procedures
DROP FUNCTION IF EXISTS get_all_qa_pairs();
DROP PROCEDURE IF EXISTS get_all_qa_pairs();
DROP FUNCTION IF EXISTS get_qa_pair_by_id(NUMBER);
DROP PROCEDURE IF EXISTS get_qa_pair_by_id(NUMBER);
DROP FUNCTION IF EXISTS search_qa_pairs(TEXT);
DROP PROCEDURE IF EXISTS search_qa_pairs(TEXT);
DROP FUNCTION IF EXISTS get_qa_pairs_by_date_range(TIMESTAMP_NTZ, TIMESTAMP_NTZ);
DROP PROCEDURE IF EXISTS get_qa_pairs_by_date_range(TIMESTAMP_NTZ, TIMESTAMP_NTZ);
DROP FUNCTION IF EXISTS get_recent_qa_pairs(NUMBER);
DROP PROCEDURE IF EXISTS get_recent_qa_pairs(NUMBER);
DROP FUNCTION IF EXISTS get_qa_pairs_count();
DROP PROCEDURE IF EXISTS get_qa_pairs_count();
DROP FUNCTION IF EXISTS get_qa_pairs_stats();
DROP PROCEDURE IF EXISTS get_qa_pairs_stats();
DROP PROCEDURE IF EXISTS batch_insert_qa_pairs_proc(VARIANT);

-- Drop all stored procedures
DROP PROCEDURE IF EXISTS add_qa_pair(TEXT, TEXT);
DROP PROCEDURE IF EXISTS update_qa_pair(NUMBER, TEXT, TEXT);
DROP PROCEDURE IF EXISTS delete_qa_pair(NUMBER);
DROP PROCEDURE IF EXISTS add_qa_pairs_batch(VARIANT);

-- Drop the sequence
DROP SEQUENCE IF EXISTS daily_qa_pairs_seq;

-- Drop the main table
DROP TABLE IF EXISTS daily_qa_pairs;

-- Drop optional views (if any were created later)
-- DROP VIEW IF EXISTS qa_pairs_view;

-- Final status confirmation
```

```
SELECT 'Cleanup completed successfully' AS status;
```

4.3 Sample Data Population

We include sample data to demonstrate our functionalities:

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

-- Insert sample QA pairs with auto-incrementing IDs
INSERT INTO daily_qa_pairs (id, question, answer) VALUES
(daily_qa_pairs_seq.NEXTVAL, 'What is Snowflake?',
 'Snowflake is a cloud-native data platform that offers data warehouse, data lake, data engineering, and data sharing capabilities.'),
(daily_qa_pairs_seq.NEXTVAL, 'How does Snowflake handle data storage?',
 'Snowflake uses a unique architecture that separates storage and compute, allowing for independent scaling and cost optimization.'),
(daily_qa_pairs_seq.NEXTVAL, 'What are Snowflake warehouses?',
 'Snowflake warehouses are compute resources that process queries and DML operations. They can be started, stopped, and scaled independently.'),
(daily_qa_pairs_seq.NEXTVAL, 'What is data sharing in Snowflake?',
 'Snowflake enables secure data sharing between accounts without copying or transferring data, using a unique approach called Secure Data Sharing.'),
(daily_qa_pairs_seq.NEXTVAL, 'How does Snowflake handle data security?',
 'Snowflake provides comprehensive security features including encryption at rest and in transit, role-based access control, and network security policie

SELECT 'Sample data inserted successfully' as status;
```

4.4 Function Definitions and UDFs

The core functionality is implemented through a set of functions and procedures defined in `daily_qa_pairs_functions.sql`. This file defines all the core functions and procedures that implement the system's functionality:

- CRUD operations (add, get, update, delete)
- Search and filtering capabilities
- Date range queries
- Statistical analysis
- Pagination support

```
-- Use the KV_STORE database
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

-- Create a procedure to add a new QA pair
```

```
CREATE OR REPLACE PROCEDURE add_qa_pair(question TEXT, answer TEXT)
RETURNS NUMBER
LANGUAGE SQL
AS
$$
DECLARE
    new_id NUMBER;
BEGIN
    INSERT INTO daily_qa_pairs (id, question, answer)
    VALUES (daily_qa_pairs_seq.NEXTVAL, :question, :answer);
    SELECT daily_qa_pairs_seq.CURRVAL INTO :new_id;
    RETURN :new_id;
END;
$$;

-- Create a function to get all QA pairs
CREATE OR REPLACE FUNCTION get_all_qa_pairs()
RETURNS TABLE (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
ORDER BY created_at DESC';

-- Create a function to get a specific QA pair by ID
CREATE OR REPLACE FUNCTION get_qa_pair_by_id(qa_id NUMBER)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
```

```
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
WHERE id = qa_id';

-- Create a procedure to update a QA pair
CREATE OR REPLACE PROCEDURE update_qa_pair(
    qa_id NUMBER,
    new_question TEXT,
    new_answer TEXT
)
RETURNS BOOLEAN
LANGUAGE SQL
AS
$$
BEGIN
    UPDATE daily_qa_pairs
    SET
        question = :new_question,
        answer = :new_answer,
        updated_at = CURRENT_TIMESTAMP()
    WHERE id = :qa_id;
    RETURN TRUE;
END;
$$;

-- Create a procedure to delete a QA pair
CREATE OR REPLACE PROCEDURE delete_qa_pair(qa_id NUMBER)
RETURNS BOOLEAN
LANGUAGE SQL
AS
$$
BEGIN
    DELETE FROM daily_qa_pairs
    WHERE id = :qa_id;
    RETURN TRUE;
END;
$$;

-- Create a function to search QA pairs by keyword
CREATE OR REPLACE FUNCTION search_qa_pairs(keyword TEXT)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
```

```
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
WHERE
    LOWER(question) LIKE ''%'' || LOWER(keyword) || ''%'' OR
    LOWER(answer) LIKE ''%'' || LOWER(keyword) || ''%''
ORDER BY created_at DESC';

-- Create a function to get QA pairs by date range
CREATE OR REPLACE FUNCTION get_qa_pairs_by_date_range(
    start_date TIMESTAMP_NTZ,
    end_date TIMESTAMP_NTZ
)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
WHERE created_at BETWEEN start_date AND end_date
ORDER BY created_at DESC';

-- Create a function to get the most recent QA pairs
CREATE OR REPLACE FUNCTION get_recent_qa_pairs(limit_count NUMBER)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
```

```
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM (
    SELECT id, question, answer, created_at, updated_at,
           ROW_NUMBER() OVER (ORDER BY created_at DESC) as rn
    FROM daily_qa_pairs
)
WHERE rn <= limit_count';

-- Create a function to get QA pairs count
CREATE OR REPLACE FUNCTION get_qa_pairs_count()
RETURNS NUMBER
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT COUNT(*) FROM daily_qa_pairs';

-- Create a function to get QA pairs statistics
CREATE OR REPLACE FUNCTION get_qa_pairs_stats()
RETURNS TABLE (
    total_pairs NUMBER,
    oldest_pair_date TIMESTAMP_NTZ,
    newest_pair_date TIMESTAMP_NTZ,
    average_answer_length NUMBER
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT
    COUNT(*) as total_pairs,
    MIN(created_at) as oldest_pair_date,
    MAX(created_at) as newest_pair_date,
    AVG(LENGTH(answer)) as average_answer_length
FROM daily_qa_pairs';
```

The system includes several demo files that showcase different functionalities:

- Basic CRUD operations (create, read, update, delete)
- Search functionality with keyword matching
- Date-based filtering and range queries

- Statistical analysis and reporting
- Pagination and result limiting

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

SELECT 'All QA Pairs:' as message;
SELECT * FROM TABLE(get_all_qa_pairs());
```

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

SELECT 'QA Pairs containing "security":' as message;
SELECT * FROM TABLE(search_qa_pairs('security'));
```

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

SELECT '3 Most Recent QA Pairs:' as message;
SELECT * FROM TABLE(get_recent_qa_pairs(3));
```

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

SELECT 'QA Pairs Statistics:' as message;
SELECT * FROM TABLE(get_qa_pairs_stats());
```

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

SELECT 'Updating QA pair with ID 1:' as message;
CALL update_qa_pair(1, 'What is Snowflake?',
  'Snowflake is a cloud-native data platform that offers data warehouse, data lake, data engineering, data sharing, and application development capabilit

SELECT 'Updated QA pair:' as message;
SELECT * FROM TABLE(get_qa_pair_by_id(1));
```

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

SELECT 'QA Pairs created in the last hour:' as message;
SELECT * FROM TABLE(get_qa_pairs_by_date_range(
  CONVERT_TIMEZONE('UTC', DATEADD(hours, -1, CURRENT_TIMESTAMP()))::TIMESTAMP_NTZ,
  CONVERT_TIMEZONE('UTC', CURRENT_TIMESTAMP())::TIMESTAMP_NTZ
));
```

```
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

SELECT 'Deleting QA pair with ID 5:' as message;
CALL delete_qa_pair(5);

SELECT 'Remaining QA pairs:' as message;
SELECT * FROM TABLE(get_all_qa_pairs());
```

4.5 Batch Insert Demonstration

This Snowflake SQL and embedded Python script sets up a key-value storage system using a sequence, a table, and stored procedures in SQL and another in Python—for mass inserting our question-answer pairs. The script begins by establishing context with `USE DATABASE KV_STORE;` and `USE SCHEMA PUBLIC;`, ensuring operations occur in the intended logical namespace. It then cleans up any previously existing versions of the relevant procedures, functions, sequence, and table to ensure a fresh start. A `SEQUENCE` named `daily_qa_pairs_seq` is created to generate unique primary key values, followed by the creation of the `daily_qa_pairs` table, which stores QA pairs along with timestamps for creation and update events.

The `add_qa_pair` stored procedure, written in SQL, encapsulates the logic for inserting a single QA pair. It uses the sequence to generate a unique ID and inserts the provided question and answer into the `daily_qa_pairs` table. This encapsulation supports modular use and promotes reusability. Following this, a second procedure, `batch_insert_qa_pairs_sp`, is defined using Snowflake's Python UDF infrastructure with Snowpark. This Python-based procedure accepts a JSON array of question-answer dictionaries, performs validation, and for each valid pair, invokes the SQL-based `add_qa_pair` procedure via parameterized queries, aggregating the returned IDs of inserted rows.

Finally, the script executes the batch insert procedure with a sample JSON array containing two QA pairs. These are parsed, inserted into the table via the Python procedure (which in turn delegates to the SQL procedure), and the resulting inserted records are retrieved and displayed with a `SELECT` statement ordered by ID. This multi-tiered approach separates concerns—sequence management, single insert logic, and batch ingestion—leveraging Snowflake's extensibility through SQL and Python interoperability for data engineering workflows.

```
-- Use the correct context
USE DATABASE KV_STORE;
USE SCHEMA PUBLIC;

-- Clean up any existing objects
DROP PROCEDURE IF EXISTS add_qa_pair(TEXT, TEXT);
DROP PROCEDURE IF EXISTS batch_insert_qa_pairs_sp(VARIANT);
DROP FUNCTION IF EXISTS batch_insert_qa_pairs_udf(VARIANT);
DROP SEQUENCE IF EXISTS daily_qa_pairs_seq;
DROP TABLE IF EXISTS daily_qa_pairs;

-- Create the sequence
CREATE OR REPLACE SEQUENCE daily_qa_pairs_seq START = 1 INCREMENT = 1;

-- Create the table
CREATE OR REPLACE TABLE daily_qa_pairs (
```

```
    id NUMBER PRIMARY KEY,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP()
);

-- Create the add_qa_pair procedure
CREATE OR REPLACE PROCEDURE add_qa_pair(question TEXT, answer TEXT)
RETURNS NUMBER
LANGUAGE SQL
AS
$$
DECLARE
    new_id NUMBER;
BEGIN
    SELECT daily_qa_pairs_seq.NEXTVAL INTO :new_id;

    INSERT INTO daily_qa_pairs (id, question, answer)
    VALUES (:new_id, :question, :answer);

    RETURN :new_id;
END;
$$;

-- Create a stored procedure (not UDF) for batch insert using Python
CREATE OR REPLACE PROCEDURE batch_insert_qa_pairs_sp(json_data VARIANT)
RETURNS ARRAY
LANGUAGE PYTHON
RUNTIME_VERSION = '3.8'
PACKAGES = ('snowflake-snowpark-python')
HANDLER = 'run'
AS
$$
def run(session, json_data):
    inserted_ids = []

    # json_data is already a list of dictionaries
    for item in json_data:
        question = item.get('QUESTION') or item.get('question')
        answer = item.get('ANSWER') or item.get('answer')
        if not question or not answer:
            continue
```

```
        # Call add_qa_pair using parameterized query
        result = session.sql("CALL add_qa_pair(?, ?)", [question, answer]).collect()
        inserted_id = result[0][0]
        inserted_ids.append(inserted_id)

    return inserted_ids
$$;

-- Run the batch insert via stored procedure
CALL batch_insert_qa_pairs_sp(PARSE_JSON('[
    {
        "question": "What is Snowflake Time Travel?",
        "answer": "Time Travel is a feature that allows you to access historical data at any point within a defined period."
    },
    {
        "question": "How does Snowflake handle data loading?",
        "answer": "Snowflake supports multiple data loading methods including bulk loading, continuous loading, and Snowpipe."
    }
]'));

-- Show inserted pairs
SELECT * FROM daily_qa_pairs ORDER BY id;
```

4.6 Execution Log

The following log shows the actual execution log of our QA pairs management system. The log demonstrates the complete lifecycle of the system, from initial setup to final cleanup:

- **System Initialization**: Successful database and schema selection, establishing the correct context for all operations
- **Cleanup Process**: Proper removal of any existing objects to ensure a clean environment
- **Database Object Creation**: Creation of the sequence for ID generation and the main table with proper structure
- **Function and Procedure Creation**: Successful implementation of all core functions and procedures
- **Sample Data Population**: Insertion of initial QA pairs to demonstrate functionality
- **Batch Insert Demonstration**: Implementation and execution of the batch insert procedure
- **Functionality Demonstration**: Execution of all core operations:
  - Retrieving all QA pairs

  - Searching for specific content
  - Getting recent items with pagination
  - Generating statistics
  - Updating existing records
  - Filtering by date range
  - Deleting records
- **Error Handling**: Proper handling of edge cases, such as searching for non-existent terms
- **Status Reporting**: Clear success indicators for each operation

```
==== QA Pairs Management System Demo ====

==== Cleaning up existing objects ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
DROP FUNCTION IF EXISTS get_all_qa_pairs();
+--------------------------------------+
| status                               |
|--------------------------------------|
| GET_ALL_QA_PAIRS successfully dropped. |
+--------------------------------------+
DROP PROCEDURE IF EXISTS get_all_qa_pairs();
+-------------------------------------------------------------------------+
| status                                                                  |
|-------------------------------------------------------------------------|
| Drop statement executed successfully (GET_ALL_QA_PAIRS already dropped). |
+-------------------------------------------------------------------------+
```

```
DROP FUNCTION IF EXISTS get_qa_pair_by_id(NUMBER);
+----------------------------------------+
| status                                 |
|----------------------------------------|
| GET_QA_PAIR_BY_ID successfully dropped. |
+----------------------------------------+
DROP PROCEDURE IF EXISTS get_qa_pair_by_id(NUMBER);
+-------------------------------------------------------------------------------+
| status                                                                        |
|-------------------------------------------------------------------------------|
| Drop statement executed successfully (GET_QA_PAIR_BY_ID already dropped). |
+-------------------------------------------------------------------------------+
DROP FUNCTION IF EXISTS search_qa_pairs(TEXT);
+--------------------------------------+
| status                               |
|--------------------------------------|
| SEARCH_QA_PAIRS successfully dropped. |
+--------------------------------------+
DROP PROCEDURE IF EXISTS search_qa_pairs(TEXT);
+-------------------------------------------------------------------------+
| status                                                                  |
|-------------------------------------------------------------------------|
| Drop statement executed successfully (SEARCH_QA_PAIRS already dropped). |
+-------------------------------------------------------------------------+
DROP FUNCTION IF EXISTS get_qa_pairs_by_date_range(TIMESTAMP_NTZ, TIMESTAMP_NTZ);
+-------------------------------------------------+
| status                                          |
|-------------------------------------------------|
| GET_QA_PAIRS_BY_DATE_RANGE successfully dropped. |
+-------------------------------------------------+
DROP PROCEDURE IF EXISTS get_qa_pairs_by_date_range(TIMESTAMP_NTZ, TIMESTAMP_NTZ);
+-----------------------------------------------------------------------------------------+
| status                                                                                  |
|-----------------------------------------------------------------------------------------|
| Drop statement executed successfully (GET_QA_PAIRS_BY_DATE_RANGE already dropped). |
+-----------------------------------------------------------------------------------------+
DROP FUNCTION IF EXISTS get_recent_qa_pairs(NUMBER);
+-----------------------------------------+
| status                                  |
|-----------------------------------------|
```

```
| GET_RECENT_QA_PAIRS successfully dropped. |
+-------------------------------------------+
DROP PROCEDURE IF EXISTS get_recent_qa_pairs(NUMBER);
+-------------------------------------------------------------------------------+
| status                                                                        |
|-------------------------------------------------------------------------------|
| Drop statement executed successfully (GET_RECENT_QA_PAIRS already dropped). |
+-------------------------------------------------------------------------------+
DROP FUNCTION IF EXISTS get_qa_pairs_count();
+---------------------------------------+
| status                                |
|---------------------------------------|
| GET_QA_PAIRS_COUNT successfully dropped. |
+---------------------------------------+
DROP PROCEDURE IF EXISTS get_qa_pairs_count();
+-------------------------------------------------------------------------------+
| status                                                                        |
|-------------------------------------------------------------------------------|
| Drop statement executed successfully (GET_QA_PAIRS_COUNT already dropped). |
+-------------------------------------------------------------------------------+
DROP FUNCTION IF EXISTS get_qa_pairs_stats();
+---------------------------------------+
| status                                |
|---------------------------------------|
| GET_QA_PAIRS_STATS successfully dropped. |
+---------------------------------------+
DROP PROCEDURE IF EXISTS get_qa_pairs_stats();
+-------------------------------------------------------------------------------+
| status                                                                        |
|-------------------------------------------------------------------------------|
| Drop statement executed successfully (GET_QA_PAIRS_STATS already dropped). |
+-------------------------------------------------------------------------------+
DROP PROCEDURE IF EXISTS batch_insert_qa_pairs_proc(VARIANT);
+-------------------------------------------------+
| status                                          |
|-------------------------------------------------|
| BATCH_INSERT_QA_PAIRS_PROC successfully dropped. |
+-------------------------------------------------+
DROP PROCEDURE IF EXISTS add_qa_pair(TEXT, TEXT);
+----------------------------------+
```

```
| status                           |
|----------------------------------|
| ADD_QA_PAIR successfully dropped. |
+----------------------------------+
DROP PROCEDURE IF EXISTS update_qa_pair(NUMBER, TEXT, TEXT);
+-------------------------------------+
| status                              |
|-------------------------------------|
| UPDATE_QA_PAIR successfully dropped. |
+-------------------------------------+
DROP PROCEDURE IF EXISTS delete_qa_pair(NUMBER);
+-------------------------------------+
| status                              |
|-------------------------------------|
| DELETE_QA_PAIR successfully dropped. |
+-------------------------------------+
DROP PROCEDURE IF EXISTS add_qa_pairs_batch(VARIANT);
+-------------------------------------------------------------------------------+
| status                                                                        |
|-------------------------------------------------------------------------------|
| Drop statement executed successfully (ADD_QA_PAIRS_BATCH already dropped). |
+-------------------------------------------------------------------------------+
DROP SEQUENCE IF EXISTS daily_qa_pairs_seq;
+----------------------------------------+
| status                                 |
|----------------------------------------|
| DAILY_QA_PAIRS_SEQ successfully dropped. |
+----------------------------------------+
DROP TABLE IF EXISTS daily_qa_pairs;
+-------------------------------------+
| status                              |
|-------------------------------------|
| DAILY_QA_PAIRS successfully dropped. |
+-------------------------------------+
SELECT 'Cleanup completed successfully' AS status;
+--------------------------------+
| STATUS                         |
|--------------------------------|
| Cleanup completed successfully |
+--------------------------------+
```

```
\checkmark Cleanup completed successfully

==== Step 1: Setting up the QA Pairs System ====


==== Executing Creating database objects ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
CREATE OR REPLACE SEQUENCE daily_qa_pairs_seq START = 1 INCREMENT = 1;
+-------------------------------------------------+
| status                                          |
|-------------------------------------------------|
| Sequence DAILY_QA_PAIRS_SEQ successfully created. |
+-------------------------------------------------+
CREATE OR REPLACE TABLE daily_qa_pairs (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP()
);
+-----------------------------------------+
| status                                  |
|-----------------------------------------|
| Table DAILY_QA_PAIRS successfully created. |
+-----------------------------------------+
SELECT 'Database objects created successfully' as status;
+--------------------------------------+
| STATUS                               |
```

```
|--------------------------------------|
| Database objects created successfully |
+--------------------------------------+
\checkmark Creating database objects completed successfully

==== Executing Creating functions and procedures ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
CREATE OR REPLACE PROCEDURE add_qa_pair(question TEXT, answer TEXT)
RETURNS NUMBER
LANGUAGE SQL
AS
$$
DECLARE
    new_id NUMBER;
BEGIN
    INSERT INTO daily_qa_pairs (id, question, answer)
    VALUES (daily_qa_pairs_seq.NEXTVAL, :question, :answer);
    SELECT daily_qa_pairs_seq.CURRVAL INTO :new_id;
    RETURN :new_id;
END;
$$;
+-------------------------------------------+
| status                                    |
|-------------------------------------------|
| Function ADD_QA_PAIR successfully created. |
+-------------------------------------------+
CREATE OR REPLACE FUNCTION get_all_qa_pairs()
RETURNS TABLE (
```

```
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
ORDER BY created_at DESC';
+-------------------------------------------------+
| status                                          |
|-------------------------------------------------|
| Function GET_ALL_QA_PAIRS successfully created. |
+-------------------------------------------------+
CREATE OR REPLACE FUNCTION get_qa_pair_by_id(qa_id NUMBER)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
WHERE id = qa_id';
+-------------------------------------------------+
| status                                          |
|-------------------------------------------------|
| Function GET_QA_PAIR_BY_ID successfully created. |
+-------------------------------------------------+
CREATE OR REPLACE PROCEDURE update_qa_pair(
    qa_id NUMBER,
    new_question TEXT,
    new_answer TEXT
```

```
)
RETURNS BOOLEAN
LANGUAGE SQL
AS
$$
BEGIN
    UPDATE daily_qa_pairs
    SET
        question = :new_question,
        answer = :new_answer,
        updated_at = CURRENT_TIMESTAMP()
    WHERE id = :qa_id;
    RETURN TRUE;
END;
$$;
+-----------------------------------------------+
| status                                        |
|-----------------------------------------------|
| Function UPDATE_QA_PAIR successfully created. |
+-----------------------------------------------+
CREATE OR REPLACE PROCEDURE delete_qa_pair(qa_id NUMBER)
RETURNS BOOLEAN
LANGUAGE SQL
AS
$$
BEGIN
    DELETE FROM daily_qa_pairs
    WHERE id = :qa_id;
    RETURN TRUE;
END;
$$;
+-----------------------------------------------+
| status                                        |
|-----------------------------------------------|
| Function DELETE_QA_PAIR successfully created. |
+-----------------------------------------------+
CREATE OR REPLACE FUNCTION search_qa_pairs(keyword TEXT)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
```

```
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
WHERE
    LOWER(question) LIKE ''%'' || LOWER(keyword) || ''%'' OR
    LOWER(answer) LIKE ''%'' || LOWER(keyword) || ''%''
ORDER BY created_at DESC';
+-----------------------------------------------+
| status                                        |
|-----------------------------------------------|
| Function SEARCH_QA_PAIRS successfully created. |
+-----------------------------------------------+
CREATE OR REPLACE FUNCTION get_qa_pairs_by_date_range(
    start_date TIMESTAMP_NTZ,
    end_date TIMESTAMP_NTZ
)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM daily_qa_pairs
WHERE created_at BETWEEN start_date AND end_date
ORDER BY created_at DESC';
+-------------------------------------------------------+
| status                                                |
|-------------------------------------------------------|
| Function GET_QA_PAIRS_BY_DATE_RANGE successfully created. |
```

```
+-------------------------------------------------------------+
CREATE OR REPLACE FUNCTION get_recent_qa_pairs(limit_count NUMBER)
RETURNS TABLE (
    id NUMBER,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ,
    updated_at TIMESTAMP_NTZ
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT id, question, answer, created_at, updated_at
FROM (
    SELECT id, question, answer, created_at, updated_at,
           ROW_NUMBER() OVER (ORDER BY created_at DESC) as rn
    FROM daily_qa_pairs
)
WHERE rn <= limit_count';
+-----------------------------------------------------+
| status                                              |
|-----------------------------------------------------|
| Function GET_RECENT_QA_PAIRS successfully created. |
+-----------------------------------------------------+
CREATE OR REPLACE FUNCTION get_qa_pairs_count()
RETURNS NUMBER
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT COUNT(*) FROM daily_qa_pairs';
+-----------------------------------------------------+
| status                                              |
|-----------------------------------------------------|
| Function GET_QA_PAIRS_COUNT successfully created. |
+-----------------------------------------------------+
CREATE OR REPLACE FUNCTION get_qa_pairs_stats()
RETURNS TABLE (
    total_pairs NUMBER,
    oldest_pair_date TIMESTAMP_NTZ,
    newest_pair_date TIMESTAMP_NTZ,
```

```
    average_answer_length NUMBER
)
LANGUAGE SQL
RETURNS NULL ON NULL INPUT
AS
'SELECT
    COUNT(*) as total_pairs,
    MIN(created_at) as oldest_pair_date,
    MAX(created_at) as newest_pair_date,
    AVG(LENGTH(answer)) as average_answer_length
FROM daily_qa_pairs';
+--------------------------------------------------+
| status                                           |
|--------------------------------------------------|
| Function GET_QA_PAIRS_STATS successfully created. |
+--------------------------------------------------+
\checkmark Creating functions and procedures completed successfully

==== Executing Creating batch insert procedure ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
CREATE OR REPLACE PROCEDURE batch_insert_qa_pairs_proc(json_data VARIANT)
RETURNS ARRAY
LANGUAGE PYTHON
RUNTIME_VERSION = '3.8'
PACKAGES = ('snowflake-snowpark-python')
HANDLER = 'run'
AS
$$
```

```
def run(session, json_data):
    inserted_ids = []

    # json_data is already a list of dictionaries
    for item in json_data:
        question = item.get('QUESTION') or item.get('question')
        answer = item.get('ANSWER') or item.get('answer')
        if not question or not answer:
            continue

        # Call add_qa_pair using parameterized query
        result = session.sql("CALL add_qa_pair(?, ?)", ).collect()
        inserted_id = result[0][0]
        inserted_ids.append(inserted_id)

    return inserted_ids
$$;
+------------------------------------------------------------+
| status                                                     |
|------------------------------------------------------------|
| Function BATCH_INSERT_QA_PAIRS_PROC successfully created. |
+------------------------------------------------------------+
\checkmark Creating batch insert procedure completed successfully

==== Step 2: Inserting Sample Data ====


==== Executing Inserting sample data ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
```

```
+----------------------------------+
INSERT INTO daily_qa_pairs (id, question, answer) VALUES
(daily_qa_pairs_seq.NEXTVAL, 'What is Snowflake?',
 'Snowflake is a cloud-native data platform that offers data warehouse, data lake, data engineering, and data sharing capabilities.'),
(daily_qa_pairs_seq.NEXTVAL, 'How does Snowflake handle data storage?',
 'Snowflake uses a unique architecture that separates storage and compute, allowing for independent scaling and cost optimization.'),
(daily_qa_pairs_seq.NEXTVAL, 'What are Snowflake warehouses?',
 'Snowflake warehouses are compute resources that process queries and DML operations. They can be started, stopped, and scaled independent
(daily_qa_pairs_seq.NEXTVAL, 'What is data sharing in Snowflake?',
 'Snowflake enables secure data sharing between accounts without copying or transferring data, using a unique approach called Secure Data
(daily_qa_pairs_seq.NEXTVAL, 'How does Snowflake handle data security?',
 'Snowflake provides comprehensive security features including encryption at rest and in transit, role-based access control, and network s
+-------------------------+
| number of rows inserted |
|-------------------------|
| 5                       |
+-------------------------+
SELECT 'Sample data inserted successfully' as status;
+----------------------------------+
| STATUS                           |
|----------------------------------|
| Sample data inserted successfully |
+----------------------------------+
\checkmark Inserting sample data completed successfully

==== Step 3: Demonstrating Batch Insert ====


==== Executing Demonstrating batch insert functionality ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
```

```
| Statement executed successfully. |
+----------------------------------+
DROP PROCEDURE IF EXISTS add_qa_pair(TEXT, TEXT);
+----------------------------------+
| status                           |
|----------------------------------|
| ADD_QA_PAIR successfully dropped. |
+----------------------------------+
DROP PROCEDURE IF EXISTS batch_insert_qa_pairs_sp(VARIANT);
+-----------------------------------------------+
| status                                        |
|-----------------------------------------------|
| BATCH_INSERT_QA_PAIRS_SP successfully dropped. |
+-----------------------------------------------+
DROP FUNCTION IF EXISTS batch_insert_qa_pairs_udf(VARIANT);
+-------------------------------------------------------------------+
| status                                                            |
|-------------------------------------------------------------------|
| Drop statement executed successfully (BATCH_INSERT_QA_PAIRS_UDF   |
| already dropped).                                                 |
+-------------------------------------------------------------------+
DROP SEQUENCE IF EXISTS daily_qa_pairs_seq;
+----------------------------------------+
| status                                 |
|----------------------------------------|
| DAILY_QA_PAIRS_SEQ successfully dropped. |
+----------------------------------------+
DROP TABLE IF EXISTS daily_qa_pairs;
+-------------------------------------+
| status                              |
|-------------------------------------|
| DAILY_QA_PAIRS successfully dropped. |
+-------------------------------------+
CREATE OR REPLACE SEQUENCE daily_qa_pairs_seq START = 1 INCREMENT = 1;
+------------------------------------------------+
| status                                         |
|------------------------------------------------|
| Sequence DAILY_QA_PAIRS_SEQ successfully created. |
+------------------------------------------------+
CREATE OR REPLACE TABLE daily_qa_pairs (
```

```
    id NUMBER PRIMARY KEY,
    question TEXT,
    answer TEXT,
    created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
    updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP()
);
+-------------------------------------------+
| status                                    |
|-------------------------------------------|
| Table DAILY_QA_PAIRS successfully created. |
+-------------------------------------------+
CREATE OR REPLACE PROCEDURE add_qa_pair(question TEXT, answer TEXT)
RETURNS NUMBER
LANGUAGE SQL
AS
$$
DECLARE
    new_id NUMBER;
BEGIN
    SELECT daily_qa_pairs_seq.NEXTVAL INTO :new_id;

    INSERT INTO daily_qa_pairs (id, question, answer)
    VALUES (:new_id, :question, :answer);

    RETURN :new_id;
END;
$$;
+-------------------------------------------+
| status                                    |
|-------------------------------------------|
| Function ADD_QA_PAIR successfully created. |
+-------------------------------------------+
CREATE OR REPLACE PROCEDURE batch_insert_qa_pairs_sp(json_data VARIANT)
RETURNS ARRAY
LANGUAGE PYTHON
RUNTIME_VERSION = '3.8'
PACKAGES = ('snowflake-snowpark-python')
HANDLER = 'run'
AS
$$
```

```
def run(session, json_data):
    inserted_ids = []

    # json_data is already a list of dictionaries
    for item in json_data:
        question = item.get('QUESTION') or item.get('question')
        answer = item.get('ANSWER') or item.get('answer')
        if not question or not answer:
            continue

        # Call add_qa_pair using parameterized query
        result = session.sql("CALL add_qa_pair(?, ?)", ).collect()
        inserted_id = result[0][0]
        inserted_ids.append(inserted_id)

    return inserted_ids
$$;
+-----------------------------------------------------------+
| status                                                    |
|-----------------------------------------------------------|
| Function BATCH_INSERT_QA_PAIRS_SP successfully created. |
+-----------------------------------------------------------+
CALL batch_insert_qa_pairs_sp(PARSE_JSON('[
    {
        "question": "What is Snowflake Time Travel?",
        "answer": "Time Travel is a feature that allows you to access historical data at any point within a defined period."
    },
    {
        "question": "How does Snowflake handle data loading?",
        "answer": "Snowflake supports multiple data loading methods including bulk loading, continuous loading, and Snowpipe."
    }
]'));
+--------------------------+
| BATCH_INSERT_QA_PAIRS_SP |
|--------------------------|
| [                        |
|   1,                     |
|   2                      |
| ]                        |
+--------------------------+
```

```
SELECT * FROM daily_qa_pairs ORDER BY id;
+-----------------------------------------------------------------------+
| ID | QUESTION      | ANSWER        | CREATED_AT   | UPDATED_AT    |
|----+---------------+---------------+--------------+---------------|
| 1  | What is       | Time Travel   | 2025-04-07   | 2025-04-07    |
|    | Snowflake     | is a feature  | 21:57:20.819 | 21:57:20.8190 |
|    | Time Travel?  | that allows   | 000          | 00            |
|    |               | you to access |              |               |
|    |               | historical    |              |               |
|    |               | data at any   |              |               |
|    |               | point within  |              |               |
|    |               | a defined     |              |               |
|    |               | period.       |              |               |
| 2  | How does      | Snowflake     | 2025-04-07   | 2025-04-07    |
|    | Snowflake     | supports      | 21:57:21.303 | 21:57:21.3030 |
|    | handle data   | multiple data | 000          | 00            |
|    | loading?      | loading       |              |               |
|    |               | methods       |              |               |
|    |               | including     |              |               |
|    |               | bulk loading, |              |               |
|    |               | continuous    |              |               |
|    |               | loading, and  |              |               |
|    |               | Snowpipe.     |              |               |
+-----------------------------------------------------------------------+
\checkmark Demonstrating batch insert functionality completed successfully

==== Step 4: Demonstrating Functionality ====


==== Executing Getting all QA pairs ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
```

```
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
SELECT 'All QA Pairs:' as message;
+---------------+
| MESSAGE       |
|---------------|
| All QA Pairs: |
+---------------+
SELECT * FROM TABLE(get_all_qa_pairs());
+-----------------------------------------------------------------------+
| ID | QUESTION      | ANSWER        | CREATED_AT   | UPDATED_AT    |
|----+---------------+---------------+--------------+---------------|
| 2  | How does      | Snowflake     | 2025-04-07   | 2025-04-07    |
|    | Snowflake     | supports      | 21:57:21.303 | 21:57:21.3030 |
|    | handle data   | multiple data | 000          | 00            |
|    | loading?      | loading       |              |               |
|    |               | methods       |              |               |
|    |               | including     |              |               |
|    |               | bulk loading, |              |               |
|    |               | continuous    |              |               |
|    |               | loading, and  |              |               |
|    |               | Snowpipe.     |              |               |
| 1  | What is       | Time Travel   | 2025-04-07   | 2025-04-07    |
|    | Snowflake     | is a feature  | 21:57:20.819 | 21:57:20.8190 |
|    | Time Travel?  | that allows   | 000          | 00            |
|    |               | you to access |              |               |
|    |               | historical    |              |               |
|    |               | data at any   |              |               |
|    |               | point within  |              |               |
|    |               | a defined     |              |               |
|    |               | period.       |              |               |
+-----------------------------------------------------------------------+
\checkmark Getting all QA pairs completed successfully

==== Executing Searching for QA pairs containing 'security' ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
```

```
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
SELECT 'QA Pairs containing "security":' as message;
+---------------------------------+
| MESSAGE                         |
|---------------------------------|
| QA Pairs containing "security": |
+---------------------------------+
SELECT * FROM TABLE(search_qa_pairs('security'));
No data

\checkmark Searching for QA pairs containing 'security' completed successfully

==== Executing Getting 3 most recent QA pairs ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
SELECT '3 Most Recent QA Pairs:' as message;
+-------------------------+
| MESSAGE                 |
|-------------------------|
| 3 Most Recent QA Pairs: |
+-------------------------+
```

```
SELECT * FROM TABLE(get_recent_qa_pairs(3));
+-----------------------------------------------------------------------+
| ID | QUESTION      | ANSWER        | CREATED_AT   | UPDATED_AT    |
|----+---------------+---------------+--------------+---------------|
| 2  | How does      | Snowflake     | 2025-04-07   | 2025-04-07    |
|    | Snowflake     | supports      | 21:57:21.303 | 21:57:21.3030 |
|    | handle data   | multiple data | 000          | 00            |
|    | loading?      | loading       |              |               |
|    |               | methods       |              |               |
|    |               | including     |              |               |
|    |               | bulk loading, |              |               |
|    |               | continuous    |              |               |
|    |               | loading, and  |              |               |
|    |               | Snowpipe.     |              |               |
| 1  | What is       | Time Travel   | 2025-04-07   | 2025-04-07    |
|    | Snowflake     | is a feature  | 21:57:20.819 | 21:57:20.8190 |
|    | Time Travel?  | that allows   | 000          | 00            |
|    |               | you to access |              |               |
|    |               | historical    |              |               |
|    |               | data at any   |              |               |
|    |               | point within  |              |               |
|    |               | a defined     |              |               |
|    |               | period.       |              |               |
+-----------------------------------------------------------------------+
\checkmark Getting 3 most recent QA pairs completed successfully

==== Executing Getting QA pairs statistics ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
```

```
SELECT 'QA Pairs Statistics:' as message;
+----------------------+
| MESSAGE              |
|----------------------|
| QA Pairs Statistics: |
+----------------------+
SELECT * FROM TABLE(get_qa_pairs_stats());
+-------------+-----------------+-----------------+-----------------+
|             | OLDEST_PAIR_DAT | NEWEST_PAIR_DAT | AVERAGE_ANSWER_ |
| TOTAL_PAIRS | E               | E               | LENGTH          |
|-------------+-----------------+-----------------+-----------------|
| 2           | 2025-04-07      | 2025-04-07      | 105             |
|             | 21:57:20.819000 | 21:57:21.303000 |                 |
+-------------+-----------------+-----------------+-----------------+
\checkmark Getting QA pairs statistics completed successfully

==== Executing Updating a QA pair ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
SELECT 'Updating QA pair with ID 1:' as message;
+-----------------------------+
| MESSAGE                     |
|-----------------------------|
| Updating QA pair with ID 1: |
+-----------------------------+
CALL update_qa_pair(1, 'What is Snowflake?',
  'Snowflake is a cloud-native data platform that offers data warehouse, data lake, data engineering, data sharing, and application develo
+----------------+
| UPDATE_QA_PAIR |
```

```
|----------------|
| True           |
+----------------+
SELECT 'Updated QA pair:' as message;
+------------------+
| MESSAGE          |
|------------------|
| Updated QA pair: |
+------------------+
SELECT * FROM TABLE(get_qa_pair_by_id(1));
+-------------------------------------------------------------------+
| ID | QUESTION      | ANSWER        | CREATED_AT   | UPDATED_AT    |
|----+---------------+---------------+--------------+---------------|
| 1  | What is       | Snowflake is  | 2025-04-07   | 2025-04-07    |
|    | Snowflake?    | a             | 21:57:20.819 | 21:57:30.2590 |
|    |               | cloud-native  | 000          | 00            |
|    |               | data platform |              |               |
|    |               | that offers   |              |               |
|    |               | data          |              |               |
|    |               | warehouse,    |              |               |
|    |               | data lake,    |              |               |
|    |               | data          |              |               |
|    |               | engineering,  |              |               |
|    |               | data sharing, |              |               |
|    |               | and           |              |               |
|    |               | application   |              |               |
|    |               | development   |              |               |
|    |               | capabilities  |              |               |
|    |               | through       |              |               |
|    |               | Snowpark.     |              |               |
+-------------------------------------------------------------------+
\checkmark Updating a QA pair completed successfully

==== Executing Getting QA pairs by date range ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
```

```
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
SELECT 'QA Pairs created in the last hour:' as message;
+-----------------------------------+
| MESSAGE                           |
|-----------------------------------|
| QA Pairs created in the last hour: |
+-----------------------------------+
SELECT * FROM TABLE(get_qa_pairs_by_date_range(
  CONVERT_TIMEZONE('UTC', DATEADD(hours, -1, CURRENT_TIMESTAMP()))::TIMESTAMP_NTZ,
  CONVERT_TIMEZONE('UTC', CURRENT_TIMESTAMP())::TIMESTAMP_NTZ
));
No data

\checkmark Getting QA pairs by date range completed successfully

==== Executing Deleting a QA pair ====

USE DATABASE KV_STORE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
USE SCHEMA PUBLIC;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
SELECT 'Deleting QA pair with ID 5:' as message;
+----------------------------+
| MESSAGE                    |
|----------------------------|
| Deleting QA pair with ID 5: |
```

```
+-----------------------------+
CALL delete_qa_pair(5);
+----------------+
| DELETE_QA_PAIR |
|----------------|
| True           |
+----------------+
SELECT 'Remaining QA pairs:' as message;
+---------------------+
| MESSAGE             |
|---------------------|
| Remaining QA pairs: |
+---------------------+
SELECT * FROM TABLE(get_all_qa_pairs());
+----+---------------+---------------+--------------+---------------+
| ID | QUESTION      | ANSWER        | CREATED_AT   | UPDATED_AT    |
|----+---------------+---------------+--------------+---------------|
| 2  | How does      | Snowflake     | 2025-04-07   | 2025-04-07    |
|    | Snowflake     | supports      | 21:57:21.303 | 21:57:21.3030 |
|    | handle data   | multiple data | 000          | 00            |
|    | loading?      | loading       |              |               |
|    |               | methods       |              |               |
|    |               | including     |              |               |
|    |               | bulk loading, |              |               |
|    |               | continuous    |              |               |
|    |               | loading, and  |              |               |
|    |               | Snowpipe.     |              |               |
| 1  | What is       | Snowflake is  | 2025-04-07   | 2025-04-07    |
|    | Snowflake?    | a             | 21:57:20.819 | 21:57:30.2590 |
|    |               | cloud-native  | 000          | 00            |
|    |               | data platform |              |               |
|    |               | that offers   |              |               |
|    |               | data          |              |               |
|    |               | warehouse,    |              |               |
|    |               | data lake,    |              |               |
|    |               | data          |              |               |
|    |               | engineering,  |              |               |
|    |               | data sharing, |              |               |
|    |               | and           |              |               |
|    |               | application   |              |               |
```

```
|    |               | development   |             |               |
|    |               | capabilities  |             |               |
|    |               | through       |             |               |
|    |               | Snowpark.     |             |               |
+-------------------------------------------------------------------+
\checkmark Deleting a QA pair completed successfully

==== Demo Completed Successfully ====

All steps completed successfully!
```

5 Advanced Notes and Considerations - DRAFT and under work for now!

5.1 Schema Validation

```
-- Create a schema registry table
CREATE OR REPLACE TABLE schema_registry (
attribute_name VARCHAR NOT NULL,
data_type VARCHAR NOT NULL,
is_required BOOLEAN DEFAULT FALSE,
default_value VARIANT,
validation_rules VARIANT,
version NUMBER DEFAULT 1,
created_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (attribute_name, version)
);

-- Create a validation function
CREATE OR REPLACE FUNCTION validate_against_schema(
p_attribute_name VARCHAR,
p_value VARIANT,
p_version NUMBER DEFAULT NULL
)
RETURNS BOOLEAN
AS
$$
DECLARE
v_schema RECORD;
BEGIN
-- Get schema definition
SELECT * INTO v_schema
FROM schema_registry
WHERE attribute_name = p_attribute_name
AND (version = p_version OR p_version IS NULL)
ORDER BY version DESC
LIMIT 1;

-- Basic validation
IF v_schema.is_required AND p_value IS NULL THEN
RETURN FALSE;
END IF;

-- Type validation
```

```
IF p_value IS NOT NULL THEN
BEGIN
CASE v_schema.data_type
WHEN 'STRING' THEN
p_value::STRING;
WHEN 'NUMBER' THEN
p_value::NUMBER;
WHEN 'BOOLEAN' THEN
p_value::BOOLEAN;
WHEN 'DATE' THEN
p_value::DATE;
WHEN 'TIMESTAMP' THEN
p_value::TIMESTAMP_NTZ;
ELSE
NULL;
END CASE;
RETURN TRUE;
EXCEPTION
WHEN OTHER THEN
RETURN FALSE;
END;
END IF;

RETURN TRUE;
END;
$$;
```

5.2 Performance Optimization

```
-- Create clustering keys for better query performance
ALTER TABLE hybrid_storage CLUSTER BY (created_at);

-- Create materialized views for common queries
CREATE OR REPLACE MATERIALIZED VIEW mv_common_queries AS
SELECT
id,
common_field1,
attributes:color::STRING as color,
attributes:size::NUMBER as size,
attributes:category::STRING as category
FROM hybrid_storage
WHERE attributes:category::STRING = 'electronics';
```

```sql
-- Create a search optimization view
CREATE OR REPLACE VIEW vw_search_optimized AS
SELECT
id,
common_field1,
attributes:color::STRING as color,
attributes:size::NUMBER as size,
attributes:category::STRING as category,
tags
FROM hybrid_storage;

-- Create a secure view with row-level security
CREATE OR REPLACE VIEW vw_secure_data AS
SELECT *
FROM hybrid_storage
WHERE CURRENT_ROLE() IN ('ADMIN_ROLE', 'ANALYST_ROLE');
```

5.3 Data Migration

```sql
-- Migrate from EAV to JSON
INSERT INTO json_storage (data)
SELECT
OBJECT_CONSTRUCT(
'attribute_id', av.attribute_id,
'attribute_name', a.name,
'value', av.value,
'data_type', a.data_type
)
FROM attribute_values av
JOIN attributes a ON av.attribute_id = a.attribute_id;

-- Migrate from JSON to Hybrid
INSERT INTO hybrid_storage (common_field1, attributes)
SELECT
data:common_field::STRING,
OBJECT_CONSTRUCT(
'color', data:color,
'size', data:size,
'price', data:price,
'category', data:category
)
FROM json_storage;
```

```
-- Create a migration audit table
CREATE OR REPLACE TABLE migration_audit (
migration_id NUMBER AUTOINCREMENT,
source_table VARCHAR,
target_table VARCHAR,
rows_migrated NUMBER,
migration_date TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP
);
```

5.4 Monitoring and Maintenance

```
-- Monitor storage usage
SELECT
TABLE_NAME,
BYTES,
ROW_COUNT,
LAST_ALTERED
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'PUBLIC';

-- Monitor query performance
SELECT
QUERY_ID,
QUERY_TEXT,
EXECUTION_STATUS,
QUEUED_PROVISIONING_TIME,
QUEUED_REPAIR_TIME,
QUEUED_OVERLOAD_TIME,
TRANSACTION_BLOCKED_TIME,
OUTBOUND_DATA_TRANSFER_BYTES,
INBOUND_DATA_TRANSFER_BYTES,
BYTES_SCANNED,
BYTES_WRITTEN,
ROWS_PRODUCED,
COMPILATION_TIME,
EXECUTION_TIME,
QUEUED_TIME,
TRANSACTION_BLOCKED_TIME
FROM TABLE(INFORMATION_SCHEMA.QUERY_HISTORY())
WHERE QUERY_TEXT LIKE '%hybrid_storage%'
ORDER BY START_TIME DESC
LIMIT 10;
```

```
-- Clean up old data with time travel
DELETE FROM hybrid_storage
WHERE created_at < DATEADD(month, -6, CURRENT_TIMESTAMP());

-- Optimize storage
ALTER TABLE hybrid_storage CLUSTER BY (created_at);

-- Create a maintenance schedule
CREATE OR REPLACE TASK maintenance_task
WAREHOUSE = COMPUTE_WH
SCHEDULE = 'USING CRON 0 0 * * 0'  -- Run weekly
AS
ALTER TABLE hybrid_storage CLUSTER BY (created_at);
```

5.4.1 Time Travel and Data Recovery

```
-- Enable time travel
ALTER TABLE daily_qa_pairs SET DATA_RETENTION_TIME_IN_DAYS = 90;

-- Query historical data
SELECT *
FROM daily_qa_pairs
AT(TIMESTAMP => '2024-01-01 00:00:00'::TIMESTAMP_NTZ);

-- Create a point-in-time recovery procedure
CREATE OR REPLACE PROCEDURE recover_qa_pair(
    p_id NUMBER,
    p_timestamp TIMESTAMP_NTZ
)
RETURNS BOOLEAN
AS
$$
DECLARE
    v_recovered BOOLEAN;
BEGIN
    -- Attempt to recover the record
    INSERT INTO daily_qa_pairs (
        question,
        answer,
        metadata,
        tags,
        created_at,
        updated_at
```

```
    )
    SELECT
        question,
        answer,
        metadata,
        tags,
        CURRENT_TIMESTAMP(),
        CURRENT_TIMESTAMP()
    FROM daily_qa_pairs
    AT(TIMESTAMP => p_timestamp)
    WHERE id = p_id;

    v_recovered := SQLROWCOUNT > 0;
    RETURN v_recovered;
END;
$$;
```

5.4.2 Zero-Copy Cloning

```
-- Create development environment
CREATE OR REPLACE DATABASE dev_kv_store CLONE prod_kv_store;

-- Create testing environment
CREATE OR REPLACE DATABASE test_kv_store CLONE prod_kv_store;

-- Create a staging environment with time travel
CREATE OR REPLACE DATABASE staging_kv_store CLONE prod_kv_store
    DATA_RETENTION_TIME_IN_DAYS = 7;

-- Create a procedure for environment management
CREATE OR REPLACE PROCEDURE refresh_environment(
    p_environment VARCHAR,
    p_source VARCHAR DEFAULT 'prod'
)
RETURNS STRING
AS
$$
DECLARE
    v_sql VARCHAR;
    v_result STRING;
BEGIN
    v_sql := 'CREATE OR REPLACE DATABASE ' || p_environment || '_kv_store CLONE ' || p_source || '_kv_store';
    EXECUTE IMMEDIATE v_sql;
```

```
    v_result := 'Environment ' || p_environment || ' refreshed from ' || p_source;
    RETURN v_result;
END;
$$;
```

5.4.3 Performance Optimization

```
-- Create clustering keys
ALTER TABLE daily_qa_pairs CLUSTER BY (created_at);

-- Create materialized views for common queries
CREATE OR REPLACE MATERIALIZED VIEW mv_recent_qa_pairs AS
SELECT *
FROM daily_qa_pairs
WHERE created_at >= DATEADD(day, -7, CURRENT_TIMESTAMP());

-- Create a search optimization view
CREATE OR REPLACE VIEW vw_search_optimized AS
SELECT
    id,
    question,
    answer,
    metadata:category::STRING as category,
    metadata:difficulty::NUMBER as difficulty,
    metadata:source::STRING as source,
    tags
FROM daily_qa_pairs;

-- Create a procedure for performance monitoring
CREATE OR REPLACE PROCEDURE monitor_performance()
RETURNS TABLE (
    query_id STRING,
    query_text STRING,
    execution_time NUMBER,
    bytes_scanned NUMBER,
    rows_produced NUMBER
)
AS
$$
BEGIN
    RETURN TABLE(
        SELECT
            QUERY_ID,
```

```
            QUERY_TEXT,
            EXECUTION_TIME,
            BYTES_SCANNED,
            ROWS_PRODUCED
        FROM TABLE(INFORMATION_SCHEMA.QUERY_HISTORY())
        WHERE QUERY_TEXT LIKE '%daily_qa_pairs%'
        ORDER BY EXECUTION_TIME DESC
        LIMIT 10
    );
END;
$$;
```