GenAI-Assisted Data Migration Report

Run Timestamp: 2025-09-18 03:26:29.169167

schema_sql

```
CREATE TABLE CUSTOMERS (
  customer_id BIGINT PRIMARY KEY,
  customer_name VARCHAR(255) NOT NULL,
  address VARCHAR(255) NOT NULL,
  phone_number BIGINT NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  join_date DATE NOT NULL
);

CREATE TABLE INVENTORY (
  product_id BIGINT PRIMARY KEY,
  product_name VARCHAR(255) NOT NULL,
  category VARCHAR(255) NOT NULL,
  quantity_in_stock BIGINT NOT NULL DEFAULT 0,
  price_per_unit DECIMAL(10, 2) NOT NULL
);

CREATE TABLE SALES (
  sale_id BIGINT PRIMARY KEY,
  customer_id BIGINT NOT NULL,
  product_id BIGINT NOT NULL,
  quantity BIGINT NOT NULL DEFAULT 1,
  sale_date DATE NOT NULL,
  total_amount DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (customer_id) REFERENCES CUSTOMERS(customer_id),
    FOREIGN KEY (product_id) REFERENCES INVENTORY(product_id)
);
```

validation_sql

```
SELECT 'CUSTOMERS' AS table_name, COUNT(*) AS row_count FROM CUSTOMERS
UNION ALL
SELECT 'INVENTORY' AS table_name, COUNT(*) AS row_count FROM INVENTORY
UNION ALL
SELECT 'SALES' AS table_name, COUNT(*) AS row_count FROM SALES;

SELECT * FROM SALES WHERE customer_id NOT IN (SELECT customer_id FROM CUSTO

SELECT * FROM SALES WHERE product_id NOT IN (SELECT product_id FROM INVENTO

SELECT SUM(total_amount) AS total_sales FROM SALES;
```

validation_results

[{ "query": "SELECT 'CUSTOMERS' AS table_name, COUNT() *AS row_count FROM CUSTOMERS\nUNION ALL\nSELECT 'INVENTORY' AS table_name, COUNT()* AS row_count FROM INVENTORY\nUNION ALL\nSELECT 'SALES' AS table_name, COUNT(*) AS row_count FROM SALES", "result": [["CUSTOMERS", 55], ["INVENTORY", 50], ["SALES", 60]] }, { "query": "SELECT * FROM SALES WHERE customer_id NOT IN (SELECT customer_id FROM CUSTOMERS)", "result": [] }, { "query": "SELECT * FROM SALES WHERE product_id NOT IN (SELECT product_id FROM INVENTORY)", "result": [] }, { "query": "SELECT SUM(total_amount) AS total_sales FROM SALES", "result": [[35544.29]] }]

translated_sql

~~~
### MySQL Equivalent Procedures and Functions

Below are the equivalent MySQL stored procedures and functions for the prov

#### Procedure to Get Monthly Sales

```sql
DELIMITER //

CREATE PROCEDURE GetMonthlySales(IN p_month INT, IN p_year INT)
BEGIN
    SELECT DATE_FORMAT(sale_date, '%Y-%m') AS sale_month,
~~~

```
           SUM(total_amount) AS total_sales
    FROM SALES
    WHERE MONTH(sale_date) = p_month
      AND YEAR(sale_date) = p_year
    GROUP BY DATE_FORMAT(sale_date, '%Y-%m');
END //

DELIMITER ;
```

Note: In MySQL, we don't need to specify an OUT parameter for the result set. Instead, the result set is returned directly by the procedure.

Function to Check Reorder Point for Inventory

```
DELIMITER //

CREATE FUNCTION NeedReorder(p_product_id INT) RETURNS BOOLEAN
BEGIN
    DECLARE qty INT;
    SELECT quantity_in_stock INTO qty
    FROM INVENTORY
    WHERE product_id = p_product_id;
    IF qty < 100 THEN
       RETURN TRUE;
    ELSE
       RETURN FALSE;
    END IF;
END //

DELIMITER ;
```

Note: In MySQL, we need to declare the variable `qty` before using it.

Sample Business Query: Get Top 5 Customers by Total Purchase

```
SELECT c.customer_name, SUM(s.total_amount) AS total_purchase
FROM SALES s
JOIN CUSTOMERS c ON s.customer_id = c.customer_id
GROUP BY c.customer_name
ORDER BY total_purchase DESC
LIMIT 5;
```

Note: In MySQL, we use the `LIMIT` clause instead of `FETCH FIRST` to limit the number of rows returned.

Example Usage

To call the `GetMonthlySales` procedure:

```sql
CALL GetMonthlySales(1, 2022);
```

To call the `NeedReorder` function:

```sql
SELECT NeedReorder(1) AS need_reorder;
```

To execute the sample business query:

```sql
SELECT c.customer_name, SUM(s.total_amount) AS total_purchase
FROM SALES s
JOIN CUSTOMERS c ON s.customer_id = c.customer_id
GROUP BY c.customer_name
ORDER BY total_purchase DESC
LIMIT 5;
```

~~~

bi_sql

```sql
SELECT
    YEAR(order_date) AS year,
    MONTH(order_date) AS month,
    SUM(order_total) AS total_sales
FROM
    orders
GROUP BY
    YEAR(order_date),
    MONTH(order_date)
ORDER BY
    year,
    month;

SELECT
    c.customer_name,
    SUM(oi.quantity * p.product_price) AS total_revenue
FROM
    customers c
JOIN
    orders o ON c.customer_id = o.customer_id
JOIN
    order_items oi ON o.order_id = oi.order_id
JOIN
    products p ON oi.product_id = p.product_id
GROUP BY
    c.customer_name
~~~

```
ORDER BY
    total_revenue DESC
LIMIT 5;

SELECT
    p.product_name,
    p.product_id,
    i.quantity
FROM
    products p
JOIN
    inventory i ON p.product_id = i.product_id
WHERE
    i.quantity < 100;
```