GenAI-Assisted Data Migration Report

Run Timestamp: 2025-09-18 03:15:38.564515

schema_sql

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

CREATE TABLE INVENTORY (
  product_id INT PRIMARY KEY,
  product_name VARCHAR(255) NOT NULL,
  category VARCHAR(255) NOT NULL,
  quantity_in_stock INT NOT NULL DEFAULT 0,
  price_per_unit FLOAT NOT NULL
);

CREATE TABLE SALES (
  sale_id INT PRIMARY KEY,
  customer_id INT NOT NULL,
  product_id INT NOT NULL,
  quantity INT NOT NULL DEFAULT 1,
  sale_date DATE NOT NULL,
  total_amount FLOAT 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", 0], ["INVENTORY", 50], ["SALES", 0]] }, { "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": [[null]] }]

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: MySQL does not support `OUT` parameters for stored procedures like Oracle does. Instead, the result set is returned directly. To call this procedure and retrieve the result set, you can use the following query:

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

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: MySQL does not support `BOOLEAN` data type. Instead, you can use `TINYINT(1)` or `INT` to represent boolean values (0 for FALSE and 1 for TRUE). However, in this example, I have used `BOOLEAN` for simplicity. To call this function, you can use the following query:

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

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: MySQL uses the `LIMIT` clause to limit the number of rows returned, whereas Oracle uses `FETCH FIRST` clause.

~~~
## 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 * oi.unit_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
GROUP BY
    c.customer_name
ORDER BY
    total_revenue DESC
LIMIT 5;

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