1 Table Creation

Limit to 1000 rows

```
1
2       -- Creating Schema --
3
4       CREATE SCHEMA IF NOT EXISTS BikeSales;
5
6       USE BikeSales;
7
8       -- Tables Creation --
9
10      CREATE TABLE IF NOT EXISTS BikeSales.categories (
14
15      -- 2. Brands Table
16      CREATE TABLE IF NOT EXISTS BikeSales.brands (
20
21      -- 3. Stores Table
22      CREATE TABLE IF NOT EXISTS BikeSales.stores (
32
33      -- 4. Products Table
34      CREATE TABLE IF NOT EXISTS BikeSales.products (
46
47      -- 5. Customers Table
48      CREATE TABLE IF NOT EXISTS BikeSales.customers (
59
60      -- 6. Staffs Table
61      CREATE TABLE IF NOT EXISTS BikeSales.staffs (
76
77      -- 7. Orders Table
78      CREATE TABLE IF NOT EXISTS BikeSales.orders (
94
95      -- 8. Stocks Table
96      CREATE TABLE IF NOT EXISTS BikeSales.stocks (
106
107     -- 9. Order_Items Table
108     CREATE TABLE IF NOT EXISTS BikeSales.order_items (
121
```

2 Load data

Limit to 1000 rows

```
1   -- Data Transfer from CSV files
2
3   -- Brands table
4   LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/brands.csv'
5   INTO TABLE BikeSales.brands
6   FIELDS TERMINATED BY ','
7   ENCLOSED BY '"'
8   LINES TERMINATED BY '\n'
9   IGNORE 1 LINES
10  (brand_id, brand_name);
11
12  -- Categories table
13  LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/categories.csv'
14  INTO TABLE BikeSales.categories
15  FIELDS TERMINATED BY ','
16  ENCLOSED BY '"'
17  LINES TERMINATED BY '\n'
18  IGNORE 1 LINES
19  (category_id, category_name);
20
21  -- Stores table
22  LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/stores.csv'
23  INTO TABLE BikeSales.stores
24  FIELDS TERMINATED BY ','
25  ENCLOSED BY '"'
26  LINES TERMINATED BY '\n'
27  IGNORE 1 LINES
28  (store_id, store_name, phone, email, street, city, state, zip_code);
29
```

3 Data Cleaning

Limit to 1000 rows

```sql
--    Data Cleaning   --

-- 1. Identifying Missing or Null Values
SELECT * FROM BikeSales.customers WHERE email IS NULL;

-- Count total NULL values in the email column
SELECT COUNT(*) AS null_count FROM BikeSales.customers WHERE email IS NULL;


-- 2. Handling Duplicate Records
SELECT email, COUNT(*) AS count
FROM BikeSales.customers
GROUP BY email
HAVING COUNT(*) > 1;

DELETE c1
FROM BikeSales.customers c1
INNER JOIN BikeSales.customers c2
  ON c1.email = c2.email
  AND c1.customer_id > c2.customer_id;


-- 3. Standardizing Data Formats
-- Standardize phone numbers
UPDATE BikeSales.customers
SET phone = CONCAT('(', SUBSTRING(phone, 1, 3), ') ', SUBSTRING(phone, 4, 3), '-', SUBSTRING(phone, 7, 4))
WHERE phone NOT LIKE '(%) %-%';

-- Standardize first and last names
UPDATE BikeSales.customers
SET first_name = CONCAT(UCASE(LEFT(first_name, 1)), LCASE(SUBSTRING(first_name, 2))),
    last_name = CONCAT(UCASE(LEFT(last_name, 1)), LCASE(SUBSTRING(last_name, 2)));
```

3 Data Cleaning

Limit to 1000 rows

```sql

-- 4. Removing Outliers
SELECT *
FROM BikeSales.order_items
WHERE quantity > 1000;

SELECT *
FROM BikeSales.products
WHERE list_price < 1 OR list_price > 10000;


-- 5. Correcting Inconsistent Data Entries
SELECT DISTINCT category_name
FROM BikeSales.categories;

UPDATE BikeSales.categories
SET category_name = 'Children Bicycles'
WHERE category_name = 'children bicycles';


-- 6. Handling Foreign Key Inconsistencies
SELECT oi.*
FROM BikeSales.order_items oi
LEFT JOIN BikeSales.products p ON oi.product_id = p.product_id
WHERE p.product_id IS NULL;

DELETE oi
FROM BikeSales.order_items oi
LEFT JOIN BikeSales.products p ON oi.product_id = p.product_id
WHERE p.product_id IS NULL;

```

3 Data Cleaning

Limit to 1000 rows

```sql

-- 7. Removing Unnecessary Whitespace
UPDATE BikeSales.customers
SET first_name = TRIM(first_name),
    last_name = TRIM(last_name);


-- 8. Handling NULL Values
UPDATE BikeSales.customers
SET phone = 'Unknown'
WHERE phone IS NULL;

UPDATE BikeSales.order_items
SET discount = 0
WHERE discount IS NULL;


-- 9. Validating Data Integrity
SELECT *
FROM BikeSales.orders
WHERE shipped_date < order_date;


-- 10. Merging Duplicate or Fragmented Data
UPDATE BikeSales.customers c1
JOIN BikeSales.customers c2
  ON LCASE(c1.email) = LCASE(c2.email)
  AND c1.customer_id > c2.customer_id
SET c1.first_name = IF(c1.first_name IS NULL, c2.first_name, c1.first_name),
    c1.last_name = IF(c1.last_name IS NULL, c2.last_name, c1.last_name),
    c1.phone = IF(c1.phone IS NULL, c2.phone, c1.phone);

```

BP Query 1

Limit to 1000 rows

```sql
-- 1. Sales Performance by Store --

SELECT
    s.store_name,
    SUM(oi.quantity * oi.list_price * (1 - oi.discount)) AS total_revenue,
    SUM(oi.quantity) AS total_units_sold,
    COUNT(o.order_id) AS total_orders
FROM
    BikeSales.orders o
        JOIN
    BikeSales.order_items oi ON o.order_id = oi.order_id
        JOIN
    BikeSales.stores s ON o.store_id = s.store_id
GROUP BY s.store_name
ORDER BY total_revenue DESC;
```

BP Query 2

Limit to 1000 rows

```sql
-- 2. Customer Segmentation --

SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    COUNT(o.order_id) AS total_orders,
    SUM(oi.quantity * oi.list_price * (1 - oi.discount)) AS total_spent
FROM
    BikeSales.customers c
        JOIN
    BikeSales.orders o ON c.customer_id = o.customer_id
        JOIN
    BikeSales.order_items oi ON o.order_id = oi.order_id
GROUP BY c.customer_id , c.first_name , c.last_name
ORDER BY total_spent DESC;

```

BP Query 3

Limit to 1000 rows

```sql
-- 3. Inventory Optimization --

SELECT
    p.product_name,
    s.store_name,
    st.quantity AS stock_quantity,
    SUM(oi.quantity) AS total_units_sold
FROM
    BikeSales.stocks st
        JOIN
    BikeSales.products p ON st.product_id = p.product_id
        JOIN
    BikeSales.stores s ON st.store_id = s.store_id
        LEFT JOIN
    BikeSales.order_items oi ON st.product_id = oi.product_id
GROUP BY p.product_name , s.store_name
HAVING stock_quantity < 10
ORDER BY total_units_sold DESC;

```

BP Query 4

Limit to 1000 rows

```sql
-- 4. Employee Performance --

SELECT
    st.first_name,
    st.last_name,
    s.store_name,
    COUNT(o.order_id) AS total_orders_handled,
    SUM(oi.quantity * oi.list_price * (1 - oi.discount)) AS total_sales_handled
FROM
    BikeSales.orders o
        JOIN
    BikeSales.order_items oi ON o.order_id = oi.order_id
        JOIN
    BikeSales.staffs st ON o.staff_id = st.staff_id
        JOIN
    BikeSales.stores s ON st.store_id = s.store_id
GROUP BY st.first_name , st.last_name , s.store_name
ORDER BY total_sales_handled DESC;
```

BP Query 5

Limit to 1000 rows

```sql
-- 5. Discount Impact on Sales --

SELECT
    p.product_name,
    AVG(oi.discount) AS avg_discount,
    SUM(oi.quantity) AS total_units_sold,
    SUM(oi.quantity * oi.list_price * (1 - oi.discount)) AS total_revenue
FROM
    BikeSales.order_items oi
        JOIN
    BikeSales.products p ON oi.product_id = p.product_id
GROUP BY p.product_name
ORDER BY total_revenue DESC;
```