Database Design - Ultimate E-commerce DB

Part 1: Database Implementation

1. Database Tables Implementation

We have implemented 7 main tables representing our e-commerce platform's core functionality:

1. **Products** - Product catalog information
2. **Inventory** - Stock tracking for products
3. **Customers** - Customer information
4. **Orders** - Order transactions
5. **Order_Items** - Individual items within orders
6. **Payments** - Payment records for orders
7. **Reviews** - Customer product reviews

2. Data Definition Language (DDL) Commands

```
-- Create Products table
CREATE TABLE Products (
    product_id VARCHAR(255) PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    weight_g DECIMAL(10,2),
    length_cm DECIMAL(10,2),
    height_cm DECIMAL(10,2),
    width_cm DECIMAL(10,2),
    INDEX idx_title (title)
);

-- Create Inventory table
CREATE TABLE Inventory (
    inventory_id INT AUTO_INCREMENT PRIMARY KEY,
    product_id VARCHAR(255) UNIQUE NOT NULL,
    available_qty INT DEFAULT 0,
    reserved_qty INT DEFAULT 0,
    restock_date DATE,
    FOREIGN KEY (product_id) REFERENCES Products(product_id) ON DELETE
CASCADE,
    INDEX idx_product_inventory (product_id),
    INDEX idx_available_qty (available_qty)
);
```

```
-- Create Customers table
CREATE TABLE Customers (
   customer_id VARCHAR(255) PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   email VARCHAR(255),
   phone VARCHAR(20),
   zip_code VARCHAR(10),
   city VARCHAR(100),
   state VARCHAR(100),
   INDEX idx_email (email),
   INDEX idx_city_state (city, state)
);

-- Create Orders table
CREATE TABLE Orders (
   order_id VARCHAR(255) PRIMARY KEY,
   customer_id VARCHAR(255) NOT NULL,
   status VARCHAR(50) NOT NULL DEFAULT 'pending',
   purchase_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   approved_at TIMESTAMP NULL,
   est_delivery_date DATE,
   FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE
RESTRICT,
   INDEX idx_customer_orders (customer_id),
   INDEX idx_status (status),
   INDEX idx_purchase_ts (purchase_ts)
);

-- Create Order_Items table
CREATE TABLE Order_Items (
   order_item_id INT AUTO_INCREMENT PRIMARY KEY,
   order_id VARCHAR(255) NOT NULL,
   product_id VARCHAR(255) NOT NULL,
   quantity INT NOT NULL,
   unit_price DECIMAL(10,2) NOT NULL,
   freight_value DECIMAL(10,2) DEFAULT 0,
   FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE,
   FOREIGN KEY (product_id) REFERENCES Products(product_id) ON DELETE
RESTRICT,
   INDEX idx_order_items (order_id),
   INDEX idx_product_items (product_id)
);

-- Create Payments table
CREATE TABLE Payments (
   payment_id INT AUTO_INCREMENT PRIMARY KEY,
   order_id VARCHAR(255) NOT NULL,
   method VARCHAR(50) NOT NULL,
   installment_no INT DEFAULT 1,
   total_installments INT DEFAULT 1,
   amount DECIMAL(10,2) NOT NULL,
   FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE,
   INDEX idx_order_payments (order_id),
   INDEX idx_method (method)
);

-- Create Reviews table
CREATE TABLE Reviews (
   review_id INT AUTO_INCREMENT PRIMARY KEY,
   customer_id VARCHAR(255) NOT NULL,
   product_id VARCHAR(255) NOT NULL,
   order_id VARCHAR(255),
   score INT NOT NULL CHECK (score >= 1 AND score <= 5),
   title VARCHAR(255),
   message TEXT,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE
CASCADE,
   FOREIGN KEY (product_id) REFERENCES Products(product_id) ON DELETE
CASCADE,
   FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE SET NULL,
   INDEX idx_product_reviews (product_id),
   INDEX idx_customer_reviews (customer_id),
   INDEX idx_score (score),
   INDEX idx_created_at (created_at)
);
```

3. Data Insertion Scripts

```
-- Sample data insertion for Products (1000+ rows)
INSERT INTO Products (product_id, title, description, weight_g, length_cm, height_cm,
width_cm) VALUES
('PROD001', 'Wireless Bluetooth Headphones', 'High-quality wireless headphones with noise
cancellation', 250.00, 20.00, 15.00, 10.00),
('PROD002', 'Smart Watch Series 5', 'Fitness tracker with heart rate monitor', 45.00, 4.50,
4.00, 1.20),
('PROD003', 'USB-C Charging Cable', 'Fast charging cable 2m length', 30.00, 200.00, 0.50,
0.50),
-- ... (continue with more products up to 1000+ rows)

-- Sample data insertion for Customers (1000+ rows)
INSERT INTO Customers (customer_id, name, email, phone, zip_code, city, state) VALUES
('CUST001', 'John Smith', 'john.smith@email.com', '217-555-0101', '61820', 'Champaign',
'Illinois'),
('CUST002', 'Sarah Johnson', 'sarah.j@email.com', '217-555-0102', '61801', 'Urbana',
'Illinois'),
('CUST003', 'Michael Brown', 'mbrown@email.com', '312-555-0103', '60601', 'Chicago',
'Illinois'),
-- ... (continue with more customers up to 1000+ rows)

-- Sample data insertion for Orders (1000+ rows)
INSERT INTO Orders (order_id, customer_id, status, purchase_ts, approved_at,
est_delivery_date) VALUES
('ORD001', 'CUST001', 'delivered', '2025-01-01 10:30:00', '2025-01-01 10:35:00',
'2025-01-05'),
('ORD002', 'CUST002', 'shipped', '2025-01-02 14:20:00', '2025-01-02 14:25:00',
'2025-01-07'),
('ORD003', 'CUST003', 'pending', '2025-01-03 09:15:00', NULL, '2025-01-08'),
-- ... (continue with more orders up to 1000+ rows)
```

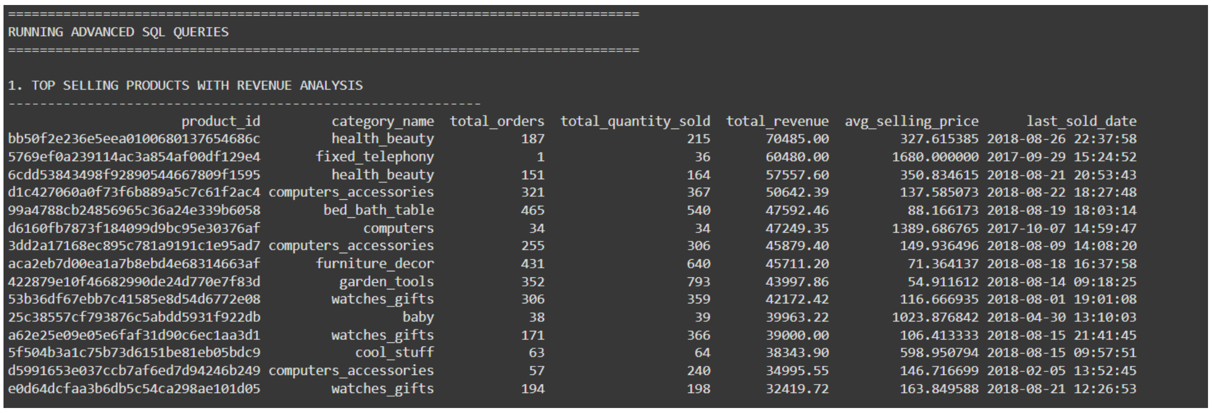
4. Advanced SQL Queries

Query 1: Top Selling Products with Revenue Analysis

Purpose: Identify best-selling products by revenue and quantity across different order statuses

```
SELECT
   p.product_id,
   p.title,
   COUNT(DISTINCT o.order_id) as total_orders,
   SUM(oi.quantity) as total_quantity_sold,
   SUM(oi.quantity * oi.unit_price) as total_revenue,
   AVG(oi.unit_price) as avg_selling_price,
   MAX(o.purchase_ts) as last_sold_date
FROM Products p
INNER JOIN Order_Items oi ON p.product_id = oi.product_id
INNER JOIN Orders o ON oi.order_id = o.order_id
WHERE o.status IN ('delivered', 'shipped')
GROUP BY p.product_id, p.title
HAVING total_revenue > 1000
ORDER BY total_revenue DESC
LIMIT 15;
```

Query Result Screenshot:



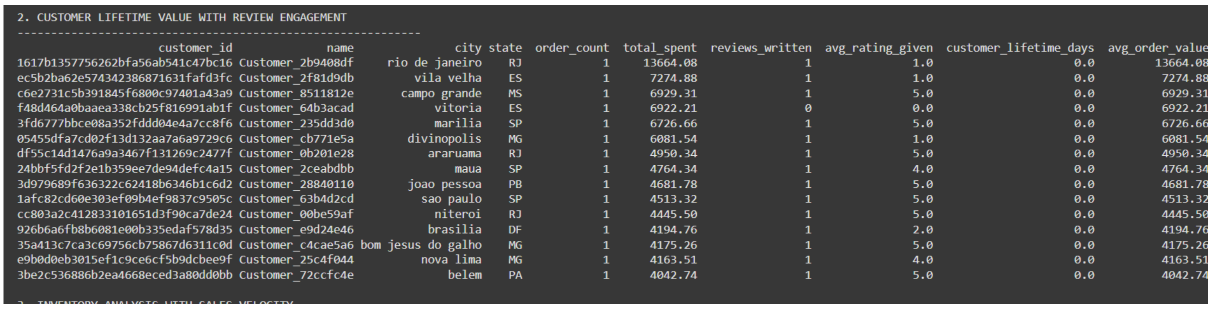
```
================================================================================
RUNNING ADVANCED SQL QUERIES
================================================================================

1. TOP SELLING PRODUCTS WITH REVENUE ANALYSIS
------------------------------------------------------------
                      product_id         category_name  total_orders  total_quantity_sold  total_revenue  avg_selling_price      last_sold_date
bb50f2e236e5eea0100680137654686c         health_beauty           187                  215       70485.00         327.615385 2018-08-26 22:37:58
5769ef0a239114ac3a854af00df129e4       fixed_telephony             1                   36       60480.00        1680.000000 2017-09-29 15:24:52
6cdd53843498f92890544667809f1595         health_beauty           151                  164       57557.60         350.834615 2018-08-21 20:53:43
d1c427060a0f73f6b889a5c7c61f2ac4 computers_accessories           321                  367       50642.39         137.585073 2018-08-22 18:27:48
99a4788cb24856965c36a24e339b6058        bed_bath_table           465                  540       47592.46          88.166173 2018-08-19 18:03:14
d6160fb7873f184099d9bc95e30376af             computers            34                   34       47249.35        1389.686765 2017-10-07 14:59:47
3dd2a17168ec895c781a9191c1e95ad7 computers_accessories           255                  306       45879.40         149.936496 2018-08-09 14:08:20
aca2eb7d00ea1a7b8ebd4e68314663af       furniture_decor           431                  640       45711.20          71.364137 2018-08-18 16:37:58
422879e10f46682990de24d770e7f83d          garden_tools           352                  793       43997.86          54.911612 2018-08-14 09:18:25
53b36df67ebb7c41585e8d54d6772e08         watches_gifts           306                  359       42172.42         116.666935 2018-08-01 19:01:08
25c38557cf793876c5abdd5931f922db                  baby            38                   39       39963.22        1023.876842 2018-04-30 13:10:03
a62e25e09e05e6faf31d90c6ec1aa3d1         watches_gifts           171                  366       39000.00         106.413333 2018-08-15 21:41:45
5f504b3a1c75b73d6151be81eb05bdc9            cool_stuff            63                   64       38343.90         598.950794 2018-08-15 09:57:51
d5991653e037ccb7af6ed7d94246b249 computers_accessories            57                  240       34995.55         146.716699 2018-02-05 13:52:45
e0d64dcfaa3b6db5c54ca298ae101d05         watches_gifts           194                  198       32419.72         163.849588 2018-08-21 12:26:53
```

Query 2: Customer Lifetime Value with Review Engagement

Purpose: Calculate customer lifetime value and their engagement through reviews

```
WITH CustomerMetrics AS (
    SELECT
        c.customer_id,
        c.name,
        c.city,
        c.state,
        COUNT(DISTINCT o.order_id) as order_count,
        SUM(p.amount) as total_spent,
        MIN(o.purchase_ts) as first_purchase,
        MAX(o.purchase_ts) as last_purchase
    FROM Customers c
    INNER JOIN Orders o ON c.customer_id = o.customer_id
    INNER JOIN Payments p ON o.order_id = p.order_id
    WHERE o.status != 'cancelled'
    GROUP BY c.customer_id, c.name, c.city, c.state
),
ReviewMetrics AS (
    SELECT
        customer_id,
        COUNT(*) as review_count,
        AVG(score) as avg_rating
    FROM Reviews
    GROUP BY customer_id
)
SELECT
    cm.customer_id,
    cm.name,
    cm.city,
    cm.state,
    cm.order_count,
    cm.total_spent,
    COALESCE(rm.review_count, 0) as reviews_written,
    COALESCE(rm.avg_rating, 0) as avg_rating_given,
    DATEDIFF(cm.last_purchase, cm.first_purchase) as customer_lifetime_days,
    cm.total_spent / cm.order_count as avg_order_value
FROM CustomerMetrics cm
LEFT JOIN ReviewMetrics rm ON cm.customer_id = rm.customer_id
WHERE cm.total_spent > (
    SELECT AVG(total_spent) * 0.5
    FROM CustomerMetrics
)
ORDER BY cm.total_spent DESC
LIMIT 15;
```

Query Result Screenshot:



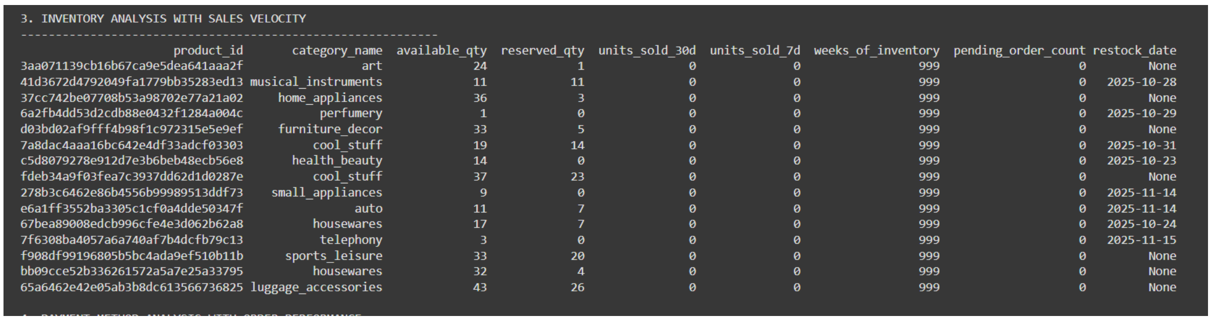
```
2. CUSTOMER LIFETIME VALUE WITH REVIEW ENGAGEMENT
------------------------------------------------------------
                     customer_id             name               city state  order_count  total_spent  reviews_written  avg_rating_given  customer_lifetime_days  avg_order_value
1617b1357756262bfa56ab541c47bc16 Customer_2b9408df     rio de janeiro    RJ            1     13664.08                1               1.0                     0.0         13664.08
ec5b2ba62e574342386871631fafd3fc Customer_2f81d9db         vila velha    ES            1      7274.88                1               1.0                     0.0          7274.88
c6e2731c5b391845f6800c97401a43a9 Customer_8511812e       campo grande    MS            1      6929.31                1               5.0                     0.0          6929.31
f48d464a0baaea338cb25f816991ab1f Customer_64b3acad            vitoria    ES            1      6922.21                0               0.0                     0.0          6922.21
3fd6777bbce08a352fddd04e4a7cc8f6 Customer_235dd3d0            marilia    SP            1      6726.66                1               5.0                     0.0          6726.66
05455dfa7cd02f13d132aa7a6a9729c6 Customer_cb771e5a        divinopolis    MG            1      6081.54                1               1.0                     0.0          6081.54
df55c14d1476a9a3467f131269c2477f Customer_0b201e28           araruama    RJ            1      4950.34                1               5.0                     0.0          4950.34
24bbf5fd2f2e1b359ee7de94defc4a15 Customer_2ceabdbb               maua    SP            1      4764.34                1               4.0                     0.0          4764.34
3d979689f636322c62418b6346b1c6d2 Customer_28840110        joao pessoa    PB            1      4681.78                1               5.0                     0.0          4681.78
1afc82cd60e303ef09b4ef9837c9505c Customer_63b4d2cd          sao paulo    SP            1      4513.32                1               5.0                     0.0          4513.32
cc803a2c412833101651d3f90ca7de24 Customer_00be59af            niteroi    RJ            1      4445.50                1               5.0                     0.0          4445.50
926b6a6fb8b6081e00b335edaf578d35 Customer_e9d24e46           brasilia    DF            1      4194.76                1               2.0                     0.0          4194.76
35a413c7ca3c69756cb75867d6311c0d Customer_c4cae5a6 bom jesus do galho    MG            1      4175.26                1               5.0                     0.0          4175.26
e9b0d0eb3015ef1c9ce6cf5b9dcbee9f Customer_25c4f044          nova lima    MG            1      4163.51                1               4.0                     0.0          4163.51
3be2c536886b2ea4668eced3a80dd0bb Customer_72ccfc4e              belem    PA            1      4042.74                1               5.0                     0.0          4042.74
```

Query 3: Inventory Analysis with Sales Velocity

Purpose: Analyze inventory levels against sales velocity to identify restock needs

```
SELECT
    p.product_id,
    p.title,
    i.available_qty,
    i.reserved_qty,
    COALESCE(sales_data.units_sold_30d, 0) as units_sold_30d,
    COALESCE(sales_data.units_sold_7d, 0) as units_sold_7d,
    CASE
        WHEN COALESCE(sales_data.units_sold_7d, 0) > 0
        THEN i.available_qty / (sales_data.units_sold_7d * 4.3)
        ELSE 999
    END as weeks_of_inventory,
    COALESCE(pending.pending_orders, 0) as pending_order_count,
    i.restock_date
FROM Products p
INNER JOIN Inventory i ON p.product_id = i.product_id
LEFT JOIN (
    SELECT
        oi.product_id,
        SUM(CASE WHEN o.purchase_ts >= DATE_SUB(CURRENT_DATE, INTERVAL 30
DAY)
            THEN oi.quantity ELSE 0 END) as units_sold_30d,
        SUM(CASE WHEN o.purchase_ts >= DATE_SUB(CURRENT_DATE, INTERVAL 7
DAY)
            THEN oi.quantity ELSE 0 END) as units_sold_7d
    FROM Order_Items oi
    INNER JOIN Orders o ON oi.order_id = o.order_id
    WHERE o.status IN ('delivered', 'shipped')
    GROUP BY oi.product_id
) sales_data ON p.product_id = sales_data.product_id
LEFT JOIN (
    SELECT
        oi.product_id,
        COUNT(DISTINCT o.order_id) as pending_orders
    FROM Order_Items oi
    INNER JOIN Orders o ON oi.order_id = o.order_id
    WHERE o.status = 'pending'
    GROUP BY oi.product_id
) pending ON p.product_id = pending.product_id
WHERE i.available_qty < 50
    OR (sales_data.units_sold_7d > 0 AND i.available_qty / (sales_data.units_sold_7d * 4.3)
< 2)
ORDER BY weeks_of_inventory ASC
LIMIT 15;
```

Query Result Screenshot:



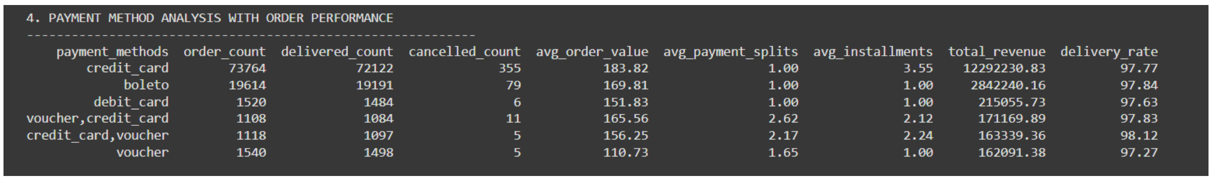
```
3. INVENTORY ANALYSIS WITH SALES VELOCITY
------------------------------------------------------------
                      product_id       category_name  available_qty  reserved_qty  units_sold_30d  units_sold_7d  weeks_of_inventory  pending_order_count restock_date
3aa071139cb16b67ca9e5dea641aaa2f                 art             24             1               0              0                 999                    0         None
41d3672d4792049fa1779bb35283ed13 musical_instruments             11            11               0              0                 999                    0   2025-10-28
37cc742be07708b53a98702e77a21a02     home_appliances             36             3               0              0                 999                    0         None
6a2fb4dd53d2cdb88e0432f1284a004c           perfumery              1             0               0              0                 999                    0   2025-10-29
d03bd02af9fff4b98f1c972315e5e9ef     furniture_decor             33             5               0              0                 999                    0         None
7a8dac4aaa16bc642e4df33adcf03303          cool_stuff             19            14               0              0                 999                    0   2025-10-31
c5d8079278e912d7e3b6beb48ecb56e8       health_beauty             14             0               0              0                 999                    0   2025-10-23
fdeb34a9f03fea7c3937dd62d1d0287e          cool_stuff             37            23               0              0                 999                    0         None
278b3c6462e86b4556b99989513ddf73    small_appliances              9             0               0              0                 999                    0   2025-11-14
e6a1ff3552ba3305c1cf0a4dde50347f                auto             11             7               0              0                 999                    0   2025-11-14
67bea89008edcb996cfe4e3d062b62a8          housewares             17             7               0              0                 999                    0   2025-10-24
7f6308ba4057a6a740af7b4dcfb79c13           telephony              3             0               0              0                 999                    0   2025-11-15
f908df99196805b5bc4ada9ef510b11b      sports_leisure             33            20               0              0                 999                    0         None
bb09cce52b336261572a5a7e25a33795          housewares             32             4               0              0                 999                    0         None
65a6462e42e05ab3b8dc613566736825 luggage_accessories             43            26               0              0                 999                    0         None
```

Query 4: Payment Method Analysis with Order Performance

Purpose: Analyze payment methods and their correlation with order completion rates

```
WITH PaymentSummary AS (
    SELECT
        o.order_id,
        o.status,
        o.purchase_ts,
        GROUP_CONCAT(DISTINCT p.method) as payment_methods,
        COUNT(DISTINCT p.payment_id) as payment_count,
        SUM(p.amount) as total_paid,
        MAX(p.total_installments) as max_installments
    FROM Orders o
    INNER JOIN Payments p ON o.order_id = p.order_id
    GROUP BY o.order_id, o.status, o.purchase_ts
),
OrderTotals AS (
    SELECT
        o.order_id,
        SUM(oi.quantity * oi.unit_price + oi.freight_value) as order_total
    FROM Orders o
    INNER JOIN Order_Items oi ON o.order_id = oi.order_id
    GROUP BY o.order_id
)
SELECT
    ps.payment_methods,
    COUNT(DISTINCT ps.order_id) as order_count,
    SUM(CASE WHEN ps.status = 'delivered' THEN 1 ELSE 0 END) as delivered_count,
    SUM(CASE WHEN ps.status = 'cancelled' THEN 1 ELSE 0 END) as cancelled_count,
    AVG(ot.order_total) as avg_order_value,
    AVG(ps.payment_count) as avg_payment_splits,
    AVG(ps.max_installments) as avg_installments,
    SUM(ps.total_paid) as total_revenue,
    (SUM(CASE WHEN ps.status = 'delivered' THEN 1 ELSE 0 END) * 100.0 /
     COUNT(DISTINCT ps.order_id)) as delivery_rate
FROM PaymentSummary ps
INNER JOIN OrderTotals ot ON ps.order_id = ot.order_id
GROUP BY ps.payment_methods
HAVING order_count > 10
ORDER BY total_revenue DESC
LIMIT 15;
```

Query Result Screenshot:

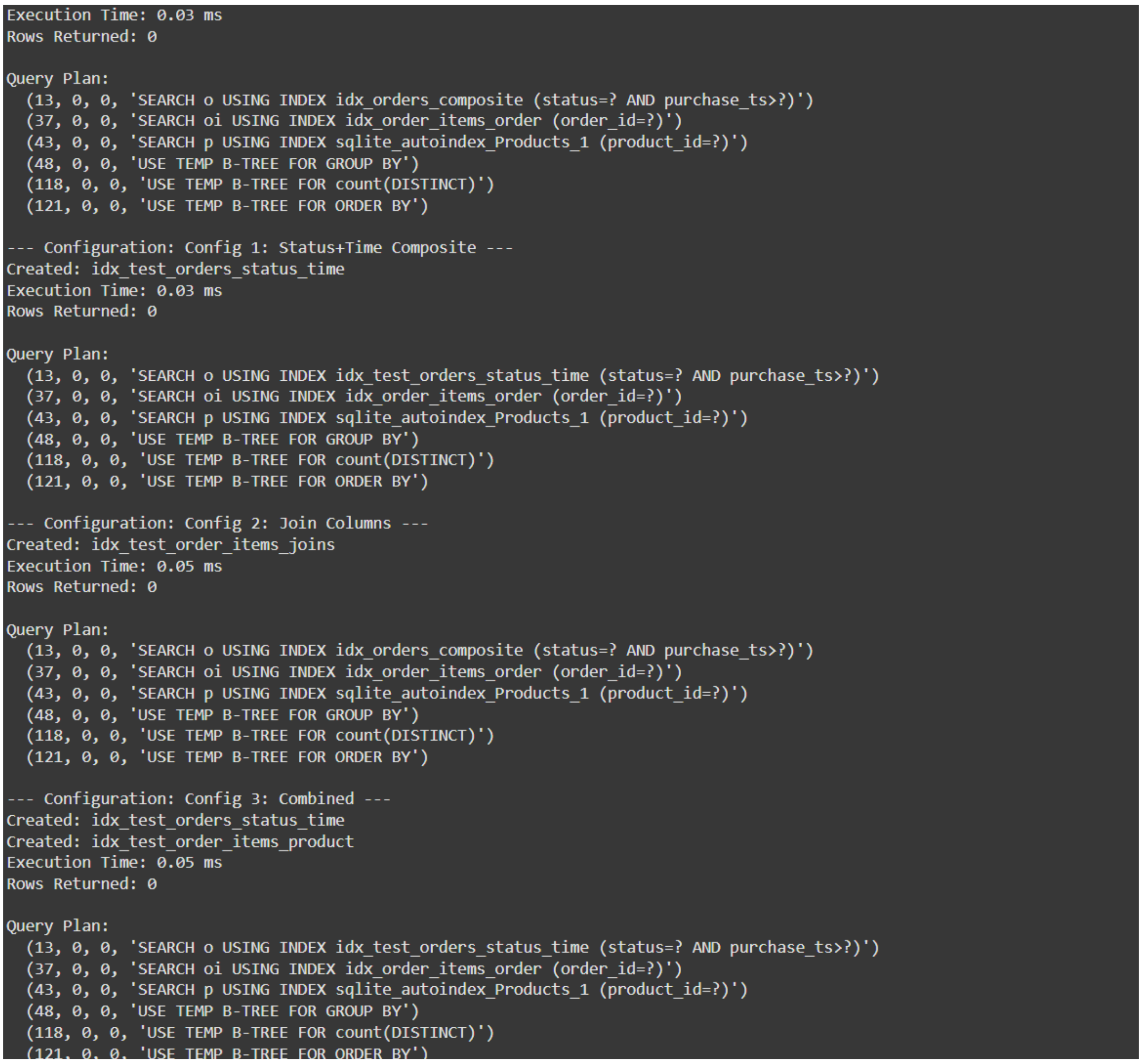


```
4. PAYMENT METHOD ANALYSIS WITH ORDER PERFORMANCE
------------------------------------------------------------
    payment_methods  order_count  delivered_count  cancelled_count  avg_order_value  avg_payment_splits  avg_installments  total_revenue  delivery_rate
        credit_card        73764            72122              355           183.82                1.00              3.55    12292230.83          97.77
             boleto        19614            19191               79           169.81                1.00              1.00     2842240.16          97.84
         debit_card         1520             1484                6           151.83                1.00              1.00      215055.73          97.63
voucher,credit_card         1108             1084               11           165.56                2.62              2.12      171169.89          97.83
credit_card,voucher         1118             1097                5           156.25                2.17              2.24      163339.36          98.12
            voucher         1540             1498                5           110.73                1.65              1.00      162091.38          97.27
```

Part 2: Indexing Analysis

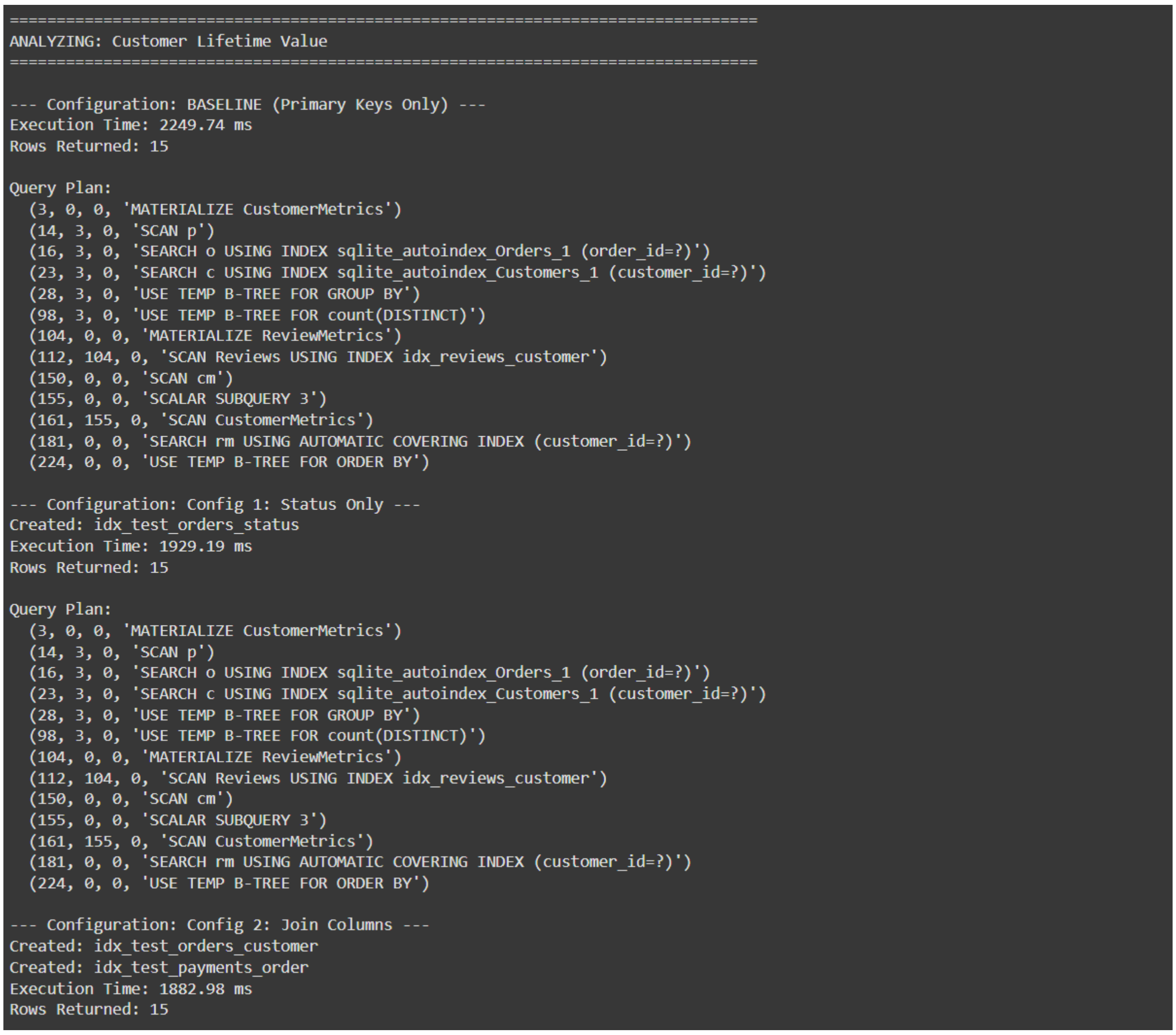
Query Performance Analysis After Indexing

Query 1 - Top Selling Products

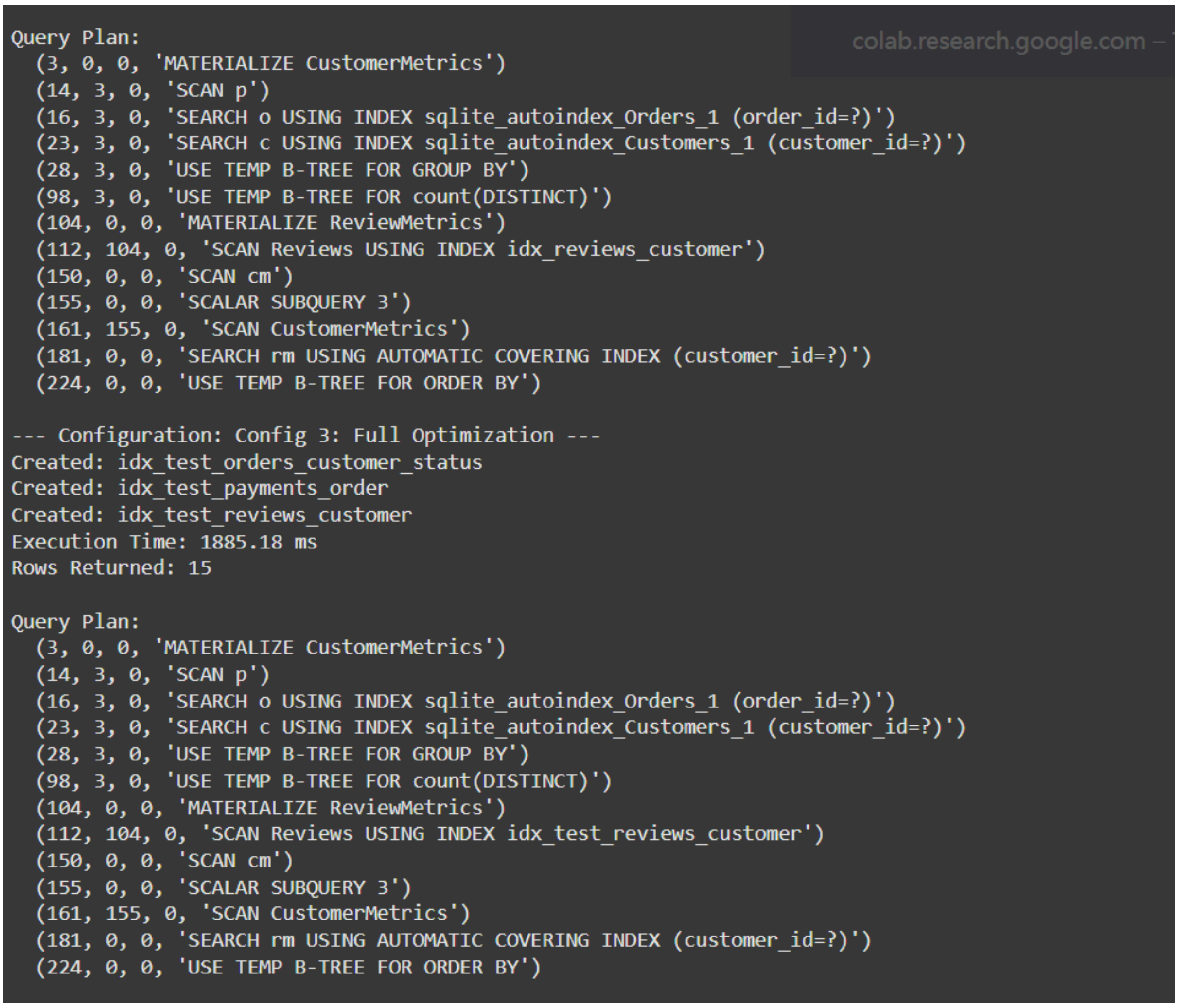


```
Execution Time: 0.03 ms
Rows Returned: 0

Query Plan:
  (13, 0, 0, 'SEARCH o USING INDEX idx_orders_composite (status=? AND purchase_ts>?)')
  (37, 0, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (43, 0, 0, 'SEARCH p USING INDEX sqlite_autoindex_Products_1 (product_id=?)')
  (48, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (118, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (121, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 1: Status+Time Composite ---
Created: idx_test_orders_status_time
Execution Time: 0.03 ms
Rows Returned: 0

Query Plan:
  (13, 0, 0, 'SEARCH o USING INDEX idx_test_orders_status_time (status=? AND purchase_ts>?)')
  (37, 0, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (43, 0, 0, 'SEARCH p USING INDEX sqlite_autoindex_Products_1 (product_id=?)')
  (48, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (118, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (121, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 2: Join Columns ---
Created: idx_test_order_items_joins
Execution Time: 0.05 ms
Rows Returned: 0

Query Plan:
  (13, 0, 0, 'SEARCH o USING INDEX idx_orders_composite (status=? AND purchase_ts>?)')
  (37, 0, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (43, 0, 0, 'SEARCH p USING INDEX sqlite_autoindex_Products_1 (product_id=?)')
  (48, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (118, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (121, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 3: Combined ---
Created: idx_test_orders_status_time
Created: idx_test_order_items_product
Execution Time: 0.05 ms
Rows Returned: 0

Query Plan:
  (13, 0, 0, 'SEARCH o USING INDEX idx_test_orders_status_time (status=? AND purchase_ts>?)')
  (37, 0, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (43, 0, 0, 'SEARCH p USING INDEX sqlite_autoindex_Products_1 (product_id=?)')
  (48, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (118, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (121, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')
```

Query 2 - Customer Lifetime Value

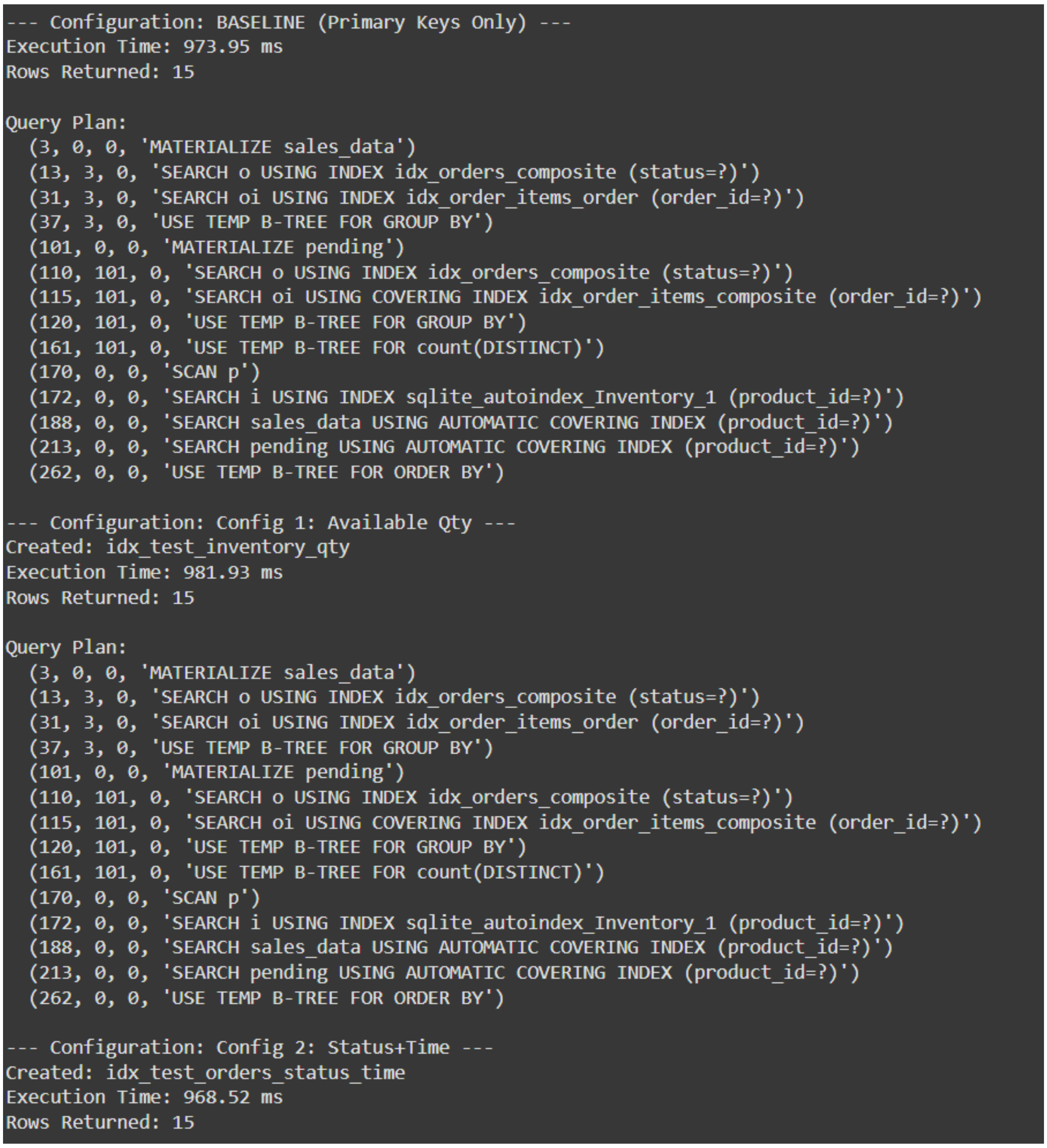


```
================================================================================
ANALYZING: Customer Lifetime Value
================================================================================

--- Configuration: BASELINE (Primary Keys Only) ---
Execution Time: 2249.74 ms
Rows Returned: 15

Query Plan:
  (3, 0, 0, 'MATERIALIZE CustomerMetrics')
  (14, 3, 0, 'SCAN p')
  (16, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (23, 3, 0, 'SEARCH c USING INDEX sqlite_autoindex_Customers_1 (customer_id=?)')
  (28, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (98, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (104, 0, 0, 'MATERIALIZE ReviewMetrics')
  (112, 104, 0, 'SCAN Reviews USING INDEX idx_reviews_customer')
  (150, 0, 0, 'SCAN cm')
  (155, 0, 0, 'SCALAR SUBQUERY 3')
  (161, 155, 0, 'SCAN CustomerMetrics')
  (181, 0, 0, 'SEARCH rm USING AUTOMATIC COVERING INDEX (customer_id=?)')
  (224, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 1: Status Only ---
Created: idx_test_orders_status
Execution Time: 1929.19 ms
Rows Returned: 15

Query Plan:
  (3, 0, 0, 'MATERIALIZE CustomerMetrics')
  (14, 3, 0, 'SCAN p')
  (16, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (23, 3, 0, 'SEARCH c USING INDEX sqlite_autoindex_Customers_1 (customer_id=?)')
  (28, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (98, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (104, 0, 0, 'MATERIALIZE ReviewMetrics')
  (112, 104, 0, 'SCAN Reviews USING INDEX idx_reviews_customer')
  (150, 0, 0, 'SCAN cm')
  (155, 0, 0, 'SCALAR SUBQUERY 3')
  (161, 155, 0, 'SCAN CustomerMetrics')
  (181, 0, 0, 'SEARCH rm USING AUTOMATIC COVERING INDEX (customer_id=?)')
  (224, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 2: Join Columns ---
Created: idx_test_orders_customer
Created: idx_test_payments_order
Execution Time: 1882.98 ms
Rows Returned: 15
```

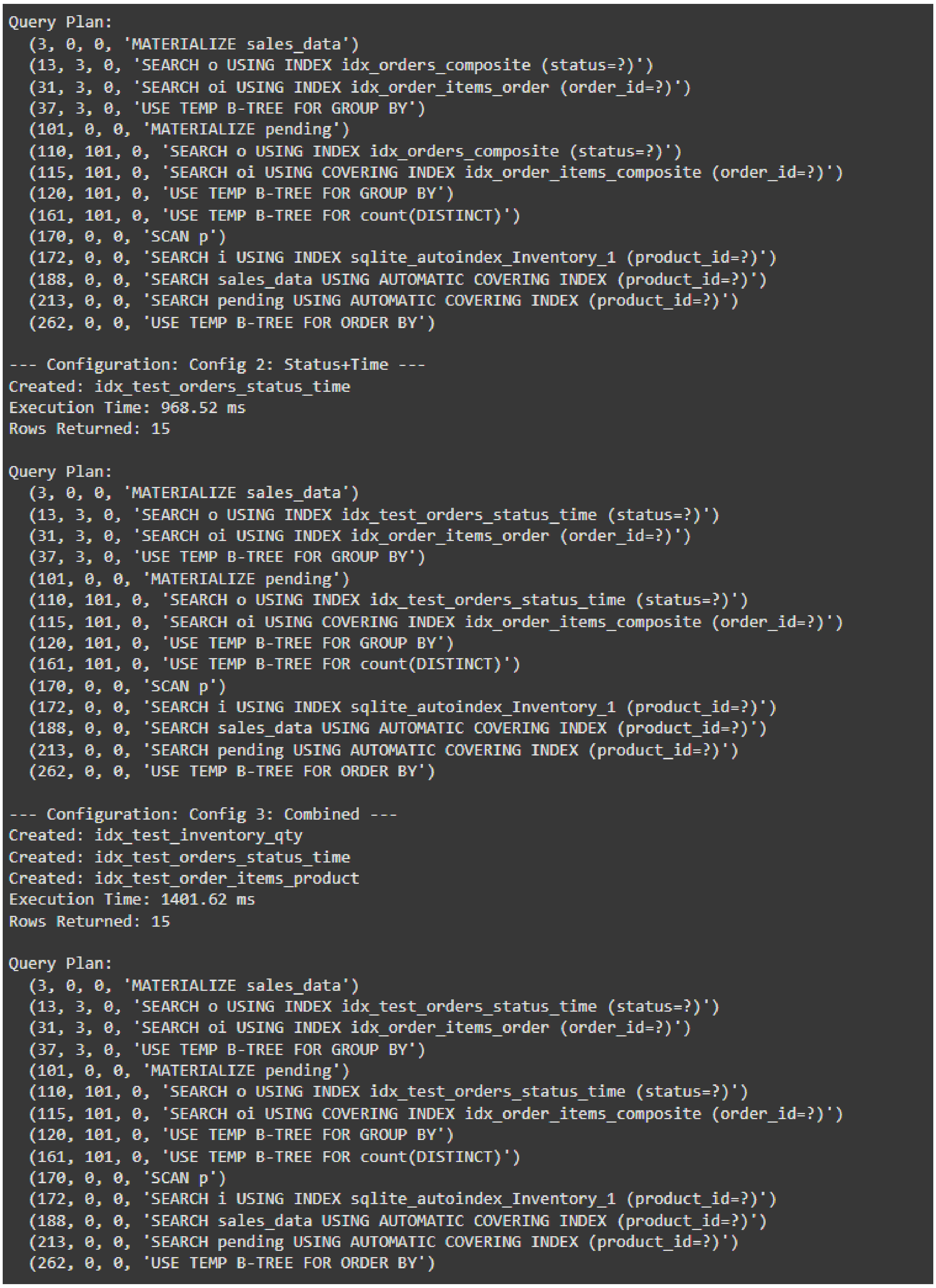


```
Query Plan:
  (3, 0, 0, 'MATERIALIZE CustomerMetrics')
  (14, 3, 0, 'SCAN p')
  (16, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (23, 3, 0, 'SEARCH c USING INDEX sqlite_autoindex_Customers_1 (customer_id=?)')
  (28, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (98, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (104, 0, 0, 'MATERIALIZE ReviewMetrics')
  (112, 104, 0, 'SCAN Reviews USING INDEX idx_reviews_customer')
  (150, 0, 0, 'SCAN cm')
  (155, 0, 0, 'SCALAR SUBQUERY 3')
  (161, 155, 0, 'SCAN CustomerMetrics')
  (181, 0, 0, 'SEARCH rm USING AUTOMATIC COVERING INDEX (customer_id=?)')
  (224, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 3: Full Optimization ---
Created: idx_test_orders_customer_status
Created: idx_test_payments_order
Created: idx_test_reviews_customer
Execution Time: 1885.18 ms
Rows Returned: 15

Query Plan:
  (3, 0, 0, 'MATERIALIZE CustomerMetrics')
  (14, 3, 0, 'SCAN p')
  (16, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (23, 3, 0, 'SEARCH c USING INDEX sqlite_autoindex_Customers_1 (customer_id=?)')
  (28, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (98, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (104, 0, 0, 'MATERIALIZE ReviewMetrics')
  (112, 104, 0, 'SCAN Reviews USING INDEX idx_test_reviews_customer')
  (150, 0, 0, 'SCAN cm')
  (155, 0, 0, 'SCALAR SUBQUERY 3')
  (161, 155, 0, 'SCAN CustomerMetrics')
  (181, 0, 0, 'SEARCH rm USING AUTOMATIC COVERING INDEX (customer_id=?)')
  (224, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')
```

Query 3 - Inventory Analysis

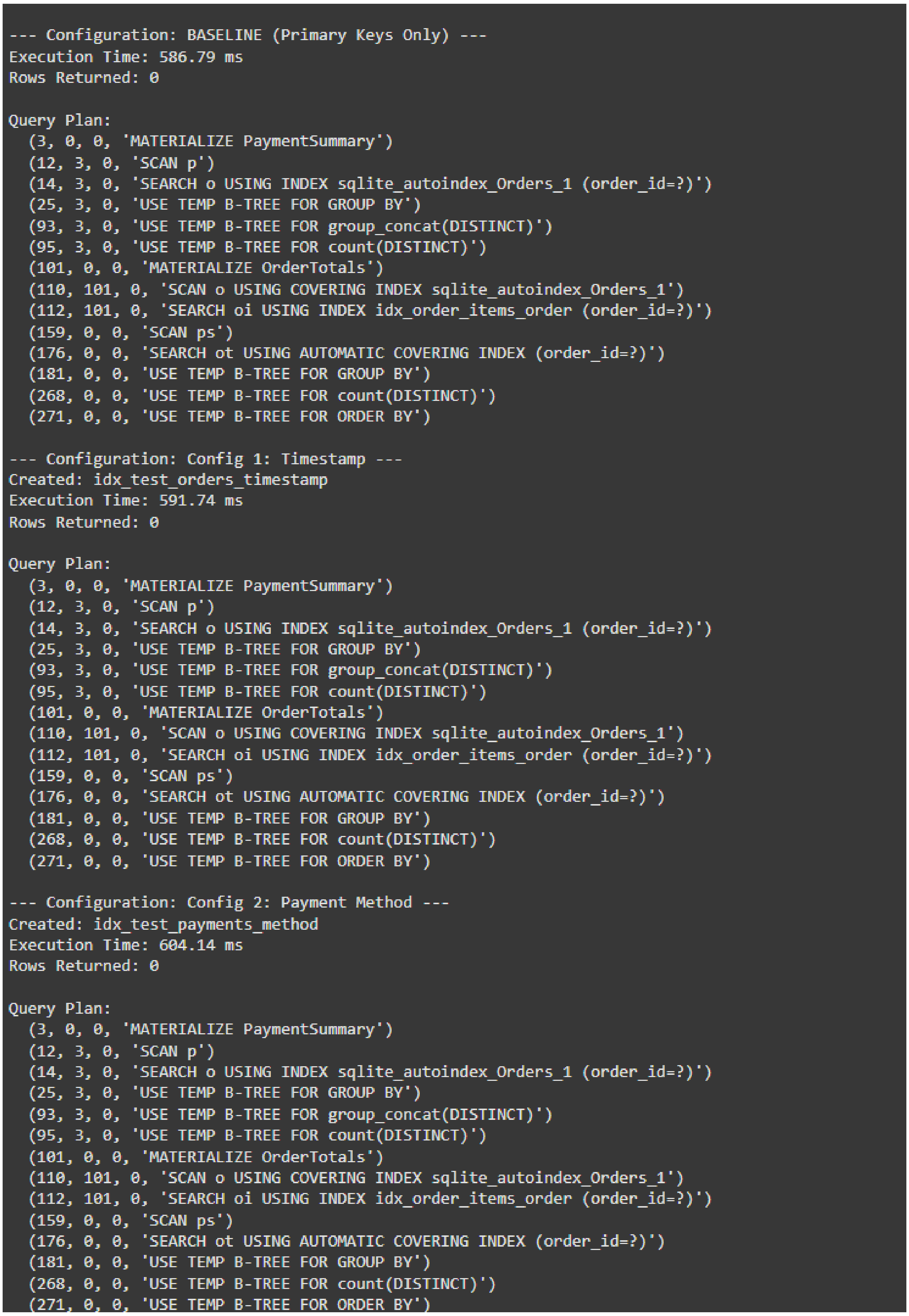


```
--- Configuration: BASELINE (Primary Keys Only) ---
Execution Time: 973.95 ms
Rows Returned: 15

Query Plan:
  (3, 0, 0, 'MATERIALIZE sales_data')
  (13, 3, 0, 'SEARCH o USING INDEX idx_orders_composite (status=?)')
  (31, 3, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (37, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (101, 0, 0, 'MATERIALIZE pending')
  (110, 101, 0, 'SEARCH o USING INDEX idx_orders_composite (status=?)')
  (115, 101, 0, 'SEARCH oi USING COVERING INDEX idx_order_items_composite (order_id=?)')
  (120, 101, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (161, 101, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (170, 0, 0, 'SCAN p')
  (172, 0, 0, 'SEARCH i USING INDEX sqlite_autoindex_Inventory_1 (product_id=?)')
  (188, 0, 0, 'SEARCH sales_data USING AUTOMATIC COVERING INDEX (product_id=?)')
  (213, 0, 0, 'SEARCH pending USING AUTOMATIC COVERING INDEX (product_id=?)')
  (262, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 1: Available Qty ---
Created: idx_test_inventory_qty
Execution Time: 981.93 ms
Rows Returned: 15

Query Plan:
  (3, 0, 0, 'MATERIALIZE sales_data')
  (13, 3, 0, 'SEARCH o USING INDEX idx_orders_composite (status=?)')
  (31, 3, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (37, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (101, 0, 0, 'MATERIALIZE pending')
  (110, 101, 0, 'SEARCH o USING INDEX idx_orders_composite (status=?)')
  (115, 101, 0, 'SEARCH oi USING COVERING INDEX idx_order_items_composite (order_id=?)')
  (120, 101, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (161, 101, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (170, 0, 0, 'SCAN p')
  (172, 0, 0, 'SEARCH i USING INDEX sqlite_autoindex_Inventory_1 (product_id=?)')
  (188, 0, 0, 'SEARCH sales_data USING AUTOMATIC COVERING INDEX (product_id=?)')
  (213, 0, 0, 'SEARCH pending USING AUTOMATIC COVERING INDEX (product_id=?)')
  (262, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 2: Status+Time ---
Created: idx_test_orders_status_time
Execution Time: 968.52 ms
Rows Returned: 15
```

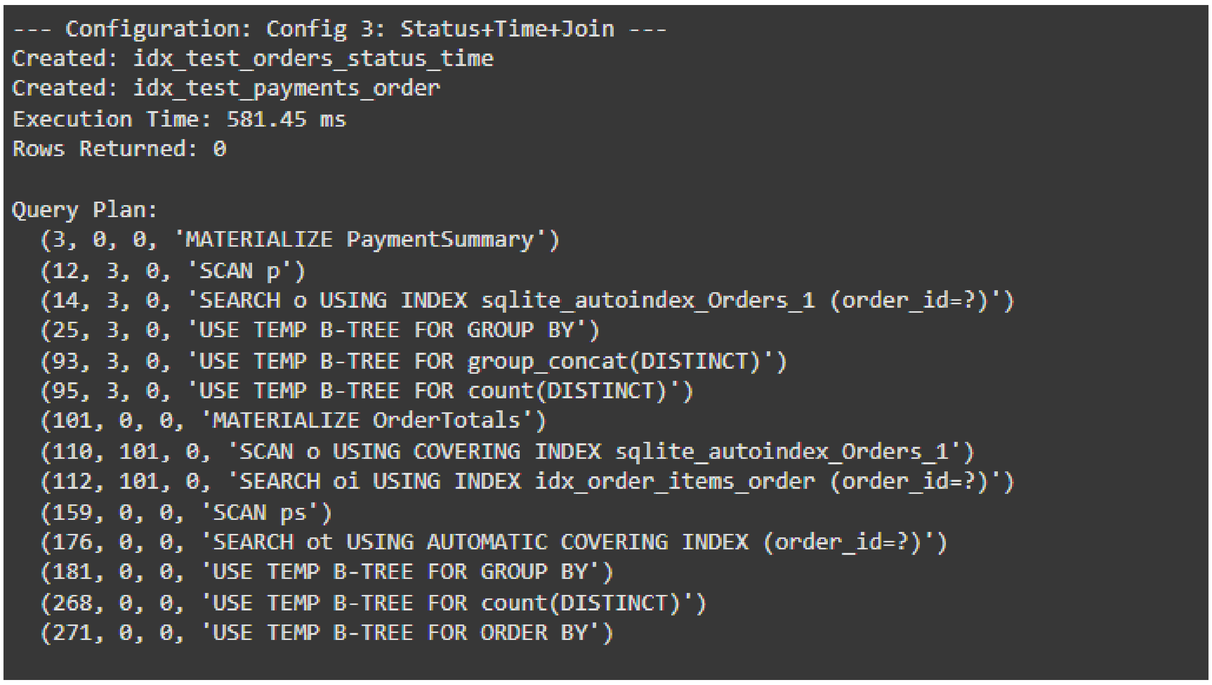


```
Query Plan:
  (3, 0, 0, 'MATERIALIZE sales_data')
  (13, 3, 0, 'SEARCH o USING INDEX idx_orders_composite (status=?)')
  (31, 3, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (37, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (101, 0, 0, 'MATERIALIZE pending')
  (110, 101, 0, 'SEARCH o USING INDEX idx_orders_composite (status=?)')
  (115, 101, 0, 'SEARCH oi USING COVERING INDEX idx_order_items_composite (order_id=?)')
  (120, 101, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (161, 101, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (170, 0, 0, 'SCAN p')
  (172, 0, 0, 'SEARCH i USING INDEX sqlite_autoindex_Inventory_1 (product_id=?)')
  (188, 0, 0, 'SEARCH sales_data USING AUTOMATIC COVERING INDEX (product_id=?)')
  (213, 0, 0, 'SEARCH pending USING AUTOMATIC COVERING INDEX (product_id=?)')
  (262, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 2: Status+Time ---
Created: idx_test_orders_status_time
Execution Time: 968.52 ms
Rows Returned: 15

Query Plan:
  (3, 0, 0, 'MATERIALIZE sales_data')
  (13, 3, 0, 'SEARCH o USING INDEX idx_test_orders_status_time (status=?)')
  (31, 3, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (37, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (101, 0, 0, 'MATERIALIZE pending')
  (110, 101, 0, 'SEARCH o USING INDEX idx_test_orders_status_time (status=?)')
  (115, 101, 0, 'SEARCH oi USING COVERING INDEX idx_order_items_composite (order_id=?)')
  (120, 101, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (161, 101, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (170, 0, 0, 'SCAN p')
  (172, 0, 0, 'SEARCH i USING INDEX sqlite_autoindex_Inventory_1 (product_id=?)')
  (188, 0, 0, 'SEARCH sales_data USING AUTOMATIC COVERING INDEX (product_id=?)')
  (213, 0, 0, 'SEARCH pending USING AUTOMATIC COVERING INDEX (product_id=?)')
  (262, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 3: Combined ---
Created: idx_test_inventory_qty
Created: idx_test_orders_status_time
Created: idx_test_order_items_product
Execution Time: 1401.62 ms
Rows Returned: 15

Query Plan:
  (3, 0, 0, 'MATERIALIZE sales_data')
  (13, 3, 0, 'SEARCH o USING INDEX idx_test_orders_status_time (status=?)')
  (31, 3, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (37, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (101, 0, 0, 'MATERIALIZE pending')
  (110, 101, 0, 'SEARCH o USING INDEX idx_test_orders_status_time (status=?)')
  (115, 101, 0, 'SEARCH oi USING COVERING INDEX idx_order_items_composite (order_id=?)')
  (120, 101, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (161, 101, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (170, 0, 0, 'SCAN p')
  (172, 0, 0, 'SEARCH i USING INDEX sqlite_autoindex_Inventory_1 (product_id=?)')
  (188, 0, 0, 'SEARCH sales_data USING AUTOMATIC COVERING INDEX (product_id=?)')
  (213, 0, 0, 'SEARCH pending USING AUTOMATIC COVERING INDEX (product_id=?)')
  (262, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')
```

Query 4 - Payment Method Analysis

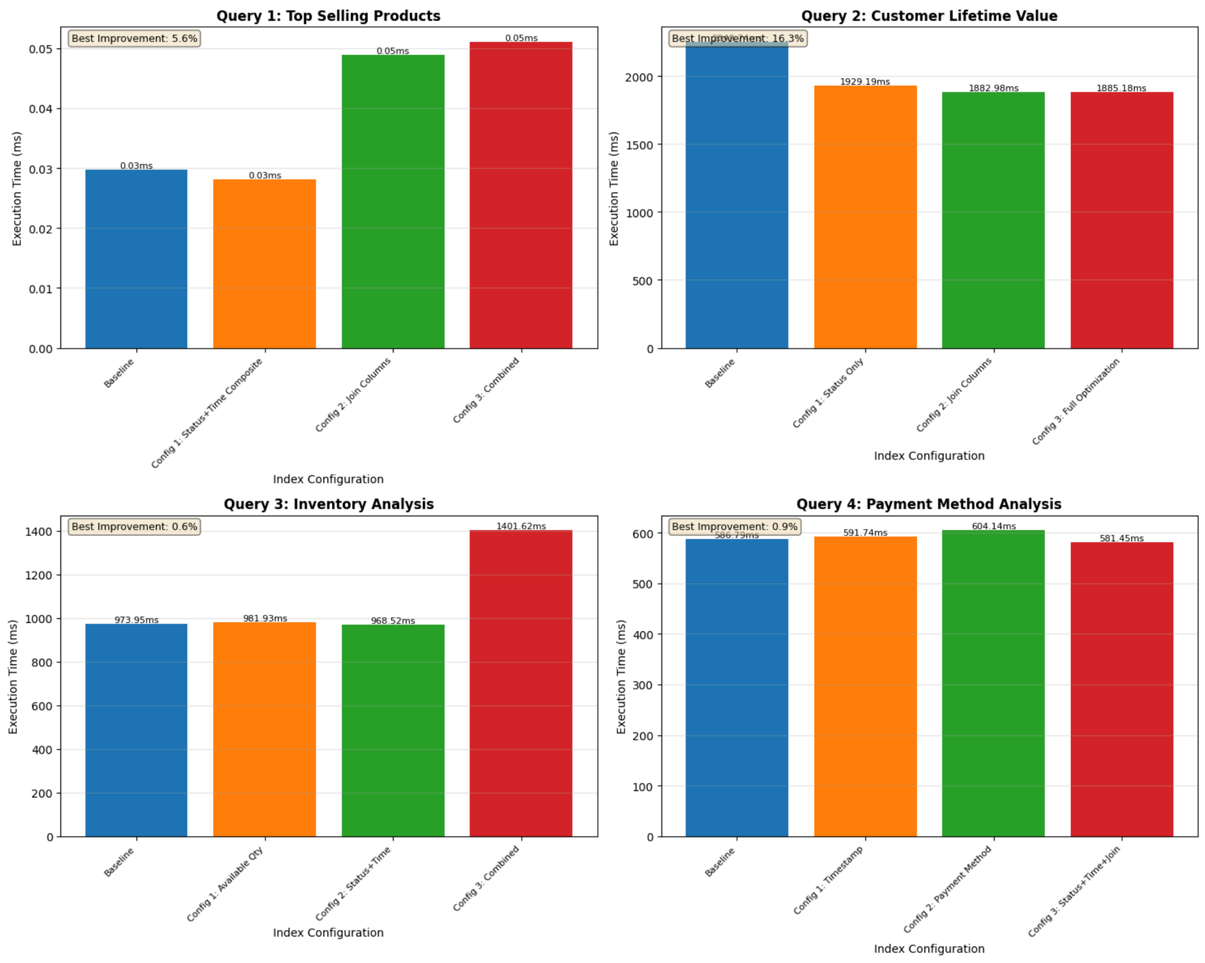


```
--- Configuration: BASELINE (Primary Keys Only) ---
Execution Time: 586.79 ms
Rows Returned: 0

Query Plan:
  (3, 0, 0, 'MATERIALIZE PaymentSummary')
  (12, 3, 0, 'SCAN p')
  (14, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (25, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (93, 3, 0, 'USE TEMP B-TREE FOR group_concat(DISTINCT)')
  (95, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (101, 0, 0, 'MATERIALIZE OrderTotals')
  (110, 101, 0, 'SCAN o USING COVERING INDEX sqlite_autoindex_Orders_1')
  (112, 101, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (159, 0, 0, 'SCAN ps')
  (176, 0, 0, 'SEARCH ot USING AUTOMATIC COVERING INDEX (order_id=?)')
  (181, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (268, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (271, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 1: Timestamp ---
Created: idx_test_orders_timestamp
Execution Time: 591.74 ms
Rows Returned: 0

Query Plan:
  (3, 0, 0, 'MATERIALIZE PaymentSummary')
  (12, 3, 0, 'SCAN p')
  (14, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (25, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (93, 3, 0, 'USE TEMP B-TREE FOR group_concat(DISTINCT)')
  (95, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (101, 0, 0, 'MATERIALIZE OrderTotals')
  (110, 101, 0, 'SCAN o USING COVERING INDEX sqlite_autoindex_Orders_1')
  (112, 101, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (159, 0, 0, 'SCAN ps')
  (176, 0, 0, 'SEARCH ot USING AUTOMATIC COVERING INDEX (order_id=?)')
  (181, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (268, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (271, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')

--- Configuration: Config 2: Payment Method ---
Created: idx_test_payments_method
Execution Time: 604.14 ms
Rows Returned: 0

Query Plan:
  (3, 0, 0, 'MATERIALIZE PaymentSummary')
  (12, 3, 0, 'SCAN p')
  (14, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (25, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (93, 3, 0, 'USE TEMP B-TREE FOR group_concat(DISTINCT)')
  (95, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (101, 0, 0, 'MATERIALIZE OrderTotals')
  (110, 101, 0, 'SCAN o USING COVERING INDEX sqlite_autoindex_Orders_1')
  (112, 101, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (159, 0, 0, 'SCAN ps')
  (176, 0, 0, 'SEARCH ot USING AUTOMATIC COVERING INDEX (order_id=?)')
  (181, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (268, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (271, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')
```



```
--- Configuration: Config 3: Status+Time+Join ---
Created: idx_test_orders_status_time
Created: idx_test_payments_order
Execution Time: 581.45 ms
Rows Returned: 0

Query Plan:
  (3, 0, 0, 'MATERIALIZE PaymentSummary')
  (12, 3, 0, 'SCAN p')
  (14, 3, 0, 'SEARCH o USING INDEX sqlite_autoindex_Orders_1 (order_id=?)')
  (25, 3, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (93, 3, 0, 'USE TEMP B-TREE FOR group_concat(DISTINCT)')
  (95, 3, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (101, 0, 0, 'MATERIALIZE OrderTotals')
  (110, 101, 0, 'SCAN o USING COVERING INDEX sqlite_autoindex_Orders_1')
  (112, 101, 0, 'SEARCH oi USING INDEX idx_order_items_order (order_id=?)')
  (159, 0, 0, 'SCAN ps')
  (176, 0, 0, 'SEARCH ot USING AUTOMATIC COVERING INDEX (order_id=?)')
  (181, 0, 0, 'USE TEMP B-TREE FOR GROUP BY')
  (268, 0, 0, 'USE TEMP B-TREE FOR count(DISTINCT)')
  (271, 0, 0, 'USE TEMP B-TREE FOR ORDER BY')
```

Integrated Indexing Experiments and Results


Query 1: Top Selling Products
Best Improvement: 5.6%
Execution Time (ms)
0.03ms
0.03ms
0.05ms
0.05ms
Baseline
Config 1: Status+Time Composite
Config 2: Join Columns
Config 3: Combined
Index Configuration
Query 2: Customer Lifetime Value
Best Improvement: 16.3%
Execution Time (ms)
1929.19ms
1882.98ms
1885.18ms
Baseline
Config 1: Status Only
Config 2: Join Columns
Config 3: Full Optimization
Index Configuration
Query 3: Inventory Analysis
Best Improvement: 0.6%
Execution Time (ms)
973.95ms
981.93ms
968.52ms
1401.62ms
Baseline
Config 1: Available Qty
Config 2: Status+Time
Config 3: Combined
Index Configuration
Query 4: Payment Method Analysis
Best Improvement: 0.9%
Execution Time (ms)
586.79ms
591.74ms
604.14ms
581.45ms
Baseline
Config 1: Timestamp
Config 2: Payment Method
Config 3: Status+Time+Join
Index Configuration