Task 4: SQL for Data Analysis — Oracle (Step-by-Step)

Task 4: SQL for Data Analysis — Oracle (Step■by■Step + Solved SQL)

Reference: I used your uploaded Task 4 PDF as the task description. ■filecite■turn0file0■

OVERVIEW
You will find below:
1) A small example schema (DDL) suitable for an ecommerce dataset.
2) How to load data (CSV hints) into Oracle.
3) Step-by-step solved Oracle SQL queries for common analysis questions asked in the task.
4) Oracle-specific tips (NVL, analytic functions, indexes, EXPLAIN PLAN).
5) How to deliver the required files (SQL file, screenshots, README).

SCHEMA (Example)

```
-- Customers, Products, Orders, Order_Items, Categories
CREATE TABLE customers (
  customer_id     NUMBER PRIMARY KEY,
  first_name      VARCHAR2(100),
  last_name       VARCHAR2(100),
  email           VARCHAR2(200),
  created_date    DATE
);

CREATE TABLE categories (
  category_id NUMBER PRIMARY KEY,
  category_name VARCHAR2(100)
);

CREATE TABLE products (
  product_id   NUMBER PRIMARY KEY,
  product_name VARCHAR2(200),
  category_id  NUMBER REFERENCES categories(category_id),
  cost_price   NUMBER(12,2),
  list_price   NUMBER(12,2)
);

CREATE TABLE orders (
  order_id     NUMBER PRIMARY KEY,
  customer_id  NUMBER REFERENCES customers(customer_id),
  order_date   DATE,
  status       VARCHAR2(50)
);

CREATE TABLE order_items (
  order_item_id NUMBER PRIMARY KEY,
  order_id      NUMBER REFERENCES orders(order_id),
  product_id    NUMBER REFERENCES products(product_id),
  quantity      NUMBER,
  unit_price    NUMBER(12,2)
);
```

HOW TO LOAD CSV DATA (brief)
1. Use SQL Developer: Tools -> Import Data -> pick CSV -> map columns -> finish.
2. Or use SQL*Loader with a simple control file.
3. Ensure dates are in 'YYYY-MM-DD' or use TO_DATE('2024-08-01','YYYY-MM-DD') on insert.

SOLUTION STEPS & EXAMPLE QUERIES (Oracle SQL syntax)

```
-- 1) Total Sales (sum of quantity * unit_price)
-- Step: use SUM over order_items
SELECT SUM(oi.quantity * oi.unit_price) AS total_sales
FROM order_items oi;

-- 2) Total Quantity sold
SELECT SUM(oi.quantity) AS total_quantity
FROM order_items oi;

-- 3) Total Profit
-- Requires cost_price in products table
SELECT SUM((oi.unit_price - p.cost_price) * oi.quantity) AS total_profit
```

```sql
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id;

-- 4) Profit Margin %
-- Use NVL to avoid division by zero
WITH totals AS (
  SELECT
    SUM(oi.quantity * oi.unit_price) AS total_sales,
    SUM((oi.unit_price - p.cost_price) * oi.quantity) AS total_profit
  FROM order_items oi
  JOIN products p ON oi.product_id = p.product_id
)
SELECT
  total_sales,
  total_profit,
  CASE
    WHEN NVL(total_sales,0) = 0 THEN 0
    ELSE ROUND((total_profit / total_sales) * 100, 2)
  END AS profit_margin_percent
FROM totals;

-- 5) Top 10 products by sales (Oracle 12c+ FETCH)
SELECT p.product_id, p.product_name,
       SUM(oi.quantity * oi.unit_price) AS product_sales
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
GROUP BY p.product_id, p.product_name
ORDER BY product_sales DESC
FETCH FIRST 10 ROWS ONLY;

-- 6) Average Revenue Per User (ARPU)
SELECT ROUND(SUM(oi.quantity * oi.unit_price) / COUNT(DISTINCT o.customer_id),2) AS arpu
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id;

-- 7) Joins — Example: customer order summary
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS customer_name,
       COUNT(DISTINCT o.order_id) AS orders_count,
       ROUND(SUM(oi.quantity * oi.unit_price),2) AS total_spent
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
LEFT JOIN 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;

-- 8) Subquery example: Products with sales greater than average product sales
SELECT product_id, product_name, product_sales
FROM (
  SELECT p.product_id, p.product_name,
         SUM(oi.quantity * oi.unit_price) AS product_sales
  FROM products p
  JOIN order_items oi ON p.product_id = oi.product_id
  GROUP BY p.product_id, p.product_name
)
WHERE product_sales > (
  SELECT AVG(product_sales) FROM (
    SELECT SUM(oi2.quantity * oi2.unit_price) AS product_sales
    FROM order_items oi2
    GROUP BY oi2.product_id
  )
);

-- 9) Create a view for monthly sales (deliverable: view)
CREATE OR REPLACE VIEW monthly_sales AS
SELECT
  TO_CHAR(o.order_date,'YYYY-MM') AS month_key,
  SUM(oi.quantity * oi.unit_price) AS monthly_revenue,
  SUM(oi.quantity) AS monthly_quantity
FROM orders o
```

```sql
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY TO_CHAR(o.order_date,'YYYY-MM');

-- Use the view:
SELECT * FROM monthly_sales ORDER BY month_key DESC;

-- 10) Analytic functions: top product per category
SELECT *
FROM (
  SELECT c.category_name, p.product_name,
         SUM(oi.quantity * oi.unit_price) AS sales_amount,
         RANK() OVER (PARTITION BY c.category_name ORDER BY SUM(oi.quantity * oi.unit_price) DESC) AS rnk
  FROM products p
  JOIN categories c ON p.category_id = c.category_id
  JOIN order_items oi ON p.product_id = oi.product_id
  GROUP BY c.category_name, p.product_name
)
WHERE rnk = 1;

-- 11) Useful null handling (Oracle uses NVL / NVL2)
SELECT order_id,
       NVL(shipping_cost,0) AS shipping_cost_zero,
       NVL2(discount, discount, 0) AS discount_present
FROM orders;

-- 12) Pagination (Oracle 12c+)
SELECT p.product_name, SUM(oi.quantity * oi.unit_price) AS sales
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_name
ORDER BY sales DESC
OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY; -- rows 21-30

-- 13) Creating Indexes (optimization)
CREATE INDEX idx_orderitems_productid ON order_items(product_id);
CREATE INDEX idx_orders_orderdate ON orders(order_date);

-- 14) Explain Plan (how to check)
EXPLAIN PLAN FOR
SELECT p.product_name, SUM(oi.quantity * oi.unit_price) sales
FROM products p JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_name;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY());

-- 15) Simple PL/SQL: stored procedure to print monthly revenue for a given month
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE print_monthly_revenue(p_month IN VARCHAR2) AS
  v_revenue NUMBER;
BEGIN
  SELECT NVL(SUM(oi.quantity * oi.unit_price),0) INTO v_revenue
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  WHERE TO_CHAR(o.order_date,'YYYY-MM') = p_month;

  DBMS_OUTPUT.PUT_LINE('Revenue for ' || p_month || ' = ' || TO_CHAR(v_revenue,'9999990.00'));
END;
/
-- Run: EXEC print_monthly_revenue('2024-07');
```

DELIVERABLES (what to submit)
1. SQL file: task4_oracle_queries.sql — include DDL + INSERT samples (if allowed) + all queries above.
2. Screenshots: Run key queries (Total Sales, Top 10 products, monthly_sales view, EXPLAIN PLAN) and capture outputs in SQL Developer.
3. README.md: Explain how you ran the script, Oracle version used, and brief explanations for each query.
4. (Optional) A small CSV loader script or SQL*Loader control file if you loaded from CSV.

TROUBLESHOOTING & ORACLE TIPS
- Use NVL to replace NULLs; NVL(expr,0).
- For better performance on large datasets, create indexes on join/filter columns (customer_id, product_id, order_date).

- Avoid wrapping indexed columns in functions in WHERE clause (e.g., don't use TO_CHAR(order_date,'YYYY') = '2024' on the indexed column — instead use a date range).
- Use bind variables in repeated PL/SQL queries to allow Oracle to reuse execution plans.
- Use ANALYZE or DBMS_STATS to keep optimizer stats up-to-date:

```
  BEGIN DBMS_STATS.GATHER_TABLE_STATS('YOUR_SCHEMA','ORDER_ITEMS'); END;
  /
```

WHAT I INCLUDED FOR YOU
- Clear step-by-step queries with Oracle-specific syntax.
- A ready-to-run SQL script outline (put it in a .sql file).
- Instructions for creating screenshots and submitting to GitHub.

END OF DOCUMENT