Here's the complete Oracle SQL solution with table creation, data insertion, and analysis queries, all formatted for Oracle Database:

1. Table Creation (Oracle Syntax):

Drop tables if they exist (optional)

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE purchases';

EXECUTE IMMEDIATE 'DROP TABLE products';

EXECUTE IMMEDIATE 'DROP TABLE users';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

/

Create users table:

CREATE TABLE users (

userid NUMBER PRIMARY KEY,

username VARCHAR2(50) NOT NULL,

email VARCHAR2(100) UNIQUE NOT NULL,

registration\_date DATE NOT NULL,

country VARCHAR2(50)

);

Create products table:

CREATE TABLE products (

productid NUMBER PRIMARY KEY,

product\_name VARCHAR2(100) NOT NULL,

category VARCHAR2(50) NOT NULL,

base\_price NUMBER(10,2) NOT NULL,

stock\_quantity NUMBER NOT NULL

);

Create purchases table:

CREATE TABLE purchases (

purchase\_id NUMBER PRIMARY KEY,

userid NUMBER REFERENCES users(userid),

productid NUMBER REFERENCES products(productid),

category VARCHAR2(50) NOT NULL,

discount NUMBER(3,2) DEFAULT 0,

final\_price NUMBER(10,2) NOT NULL,

payment\_method VARCHAR2(20) NOT NULL,

purchase\_date TIMESTAMP NOT NULL

);

2. Data Insertion (Oracle Syntax)

Insert sample data into users table:

INSERT ALL

INTO users VALUES (1, 'john\_doe', 'john@example.com', TO\_DATE('15-JAN-2022', 'DD-MON-YYYY'), 'USA')

INTO users VALUES (2, 'jane\_smith', 'jane@example.com', TO\_DATE('20-FEB-2022', 'DD-MON-YYYY'), 'UK')

INTO users VALUES (3, 'mike\_johnson', 'mike@example.com', TO\_DATE('10-MAR-2022', 'DD-MON-YYYY'), 'Canada')

INTO users VALUES (4, 'sarah\_williams', 'sarah@example.com', TO\_DATE('05-APR-2022', 'DD-MON-YYYY'), 'Australia')

INTO users VALUES (5, 'david\_brown', 'david@example.com', TO\_DATE('12-MAY-2022', 'DD-MON-YYYY'), 'USA')

SELECT \* FROM users;

Insert sample data into products table:

INSERT ALL

INTO products VALUES (101, 'Smartphone X', 'Electronics', 799.99, 50)

INTO products VALUES (102, 'Wireless Headphones', 'Electronics', 149.99, 100)

INTO products VALUES (103, 'Coffee Maker', 'Home Appliances', 89.99, 30)

INTO products VALUES (104, 'Running Shoes', 'Sports', 129.99, 75)

INTO products VALUES (105, 'Yoga Mat', 'Sports', 39.99, 120)

INTO products VALUES (106, 'Blender', 'Home Appliances', 59.99, 40)

INTO products VALUES (107, 'Smart Watch', 'Electronics', 199.99, 60)

SELECT \* FROM products;

Insert sample data into purchases table:

INSERT ALL

INTO purchases VALUES (1001, 1, 101, 'Electronics', 0.10, 719.99, 'Credit Card', TO\_TIMESTAMP('2023-01-05 10:15:22', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1002, 2, 102, 'Electronics', 0.15, 127.49, 'PayPal', TO\_TIMESTAMP('2023-01-10 14:30:45', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1003, 3, 103, 'Home Appliances', 0.00, 89.99, 'Credit Card', TO\_TIMESTAMP('2023-01-15 09:45:12', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1004, 1, 104, 'Sports', 0.20, 103.99, 'Debit Card', TO\_TIMESTAMP('2023-02-02 16:20:33', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1005, 4, 105, 'Sports', 0.05, 37.99, 'PayPal', TO\_TIMESTAMP('2023-02-10 11:10:28', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1006, 5, 106, 'Home Appliances', 0.10, 53.99, 'Credit Card', TO\_TIMESTAMP('2023-02-15 13:45:19', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1007, 2, 107, 'Electronics', 0.25, 149.99, 'PayPal', TO\_TIMESTAMP('2023-03-01 15:30:42', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1008, 3, 101, 'Electronics', 0.00, 799.99, 'Debit Card', TO\_TIMESTAMP('2023-03-10 10:20:15', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1009, 1, 102, 'Electronics', 0.20, 119.99, 'Credit Card', TO\_TIMESTAMP('2023-03-15 14:10:37', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1010, 4, 103, 'Home Appliances', 0.15, 76.49, 'PayPal', TO\_TIMESTAMP('2023-03-20 09:30:55', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1011, 5, 104, 'Sports', 0.10, 116.99, 'Debit Card', TO\_TIMESTAMP('2023-04-05 12:40:21', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1012, 2, 105, 'Sports', 0.00, 39.99, 'Credit Card', TO\_TIMESTAMP('2023-04-10 16:25:48', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1013, 3, 106, 'Home Appliances', 0.05, 56.99, 'PayPal', TO\_TIMESTAMP('2023-04-15 11:15:33', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1014, 1, 107, 'Electronics', 0.30, 139.99, 'Credit Card', TO\_TIMESTAMP('2023-05-01 10:50:22', 'YYYY-MM-DD HH24:MI:SS'))

INTO purchases VALUES (1015, 4, 101, 'Electronics', 0.10, 719.99, 'Debit Card', TO\_TIMESTAMP('2023-05-10 14:35:17', 'YYYY-MM-DD HH24:MI:SS'))

SELECT \* FROM purchases;

3. Analysis Queries (Oracle Syntax):

Basic SELECT, WHERE, ORDER BY-

Get all purchases with final price > $100:

SELECT p.purchase\_id, u.username, pr.product\_name, p.final\_price,

TO\_CHAR(p.purchase\_date, 'DD-MON-YYYY HH24:MI:SS') AS purchase\_date

FROM purchases p

JOIN users u ON p.userid = u.userid

JOIN products pr ON p.productid = pr.productid

WHERE p.final\_price > 100

ORDER BY p.final\_price DESC;

Find purchases in Electronics category made in Q1 2023:

SELECT p.purchase\_id, u.username, pr.product\_name, p.final\_price,

TO\_CHAR(p.purchase\_date, 'DD-MON-YYYY') AS purchase\_date

FROM purchases p

JOIN users u ON p.userid = u.userid

JOIN products pr ON p.productid = pr.productid

WHERE p.category = 'Electronics'

AND p.purchase\_date BETWEEN TO\_TIMESTAMP('2023-01-01', 'YYYY-MM-DD')

AND TO\_TIMESTAMP('2023-03-31 23:59:59', 'YYYY-MM-DD HH24:MI:SS')

ORDER BY p.purchase\_date;

GROUP BY with Aggregate Functions:

Average final price and discount by category:

SELECT

category,

COUNT(\*) AS purchase\_count,

ROUND(AVG(final\_price), 2) AS avg\_final\_price,

ROUND(AVG(discount), 2) AS avg\_discount,

SUM(final\_price) AS total\_revenue

FROM purchases

GROUP BY category

ORDER BY total\_revenue DESC;

Monthly sales performance:

SELECT

EXTRACT(YEAR FROM purchase\_date) AS year,

EXTRACT(MONTH FROM purchase\_date) AS month,

COUNT(\*) AS total\_orders,

COUNT(DISTINCT userid) AS unique\_customers,

SUM(final\_price) AS total\_revenue,

ROUND(AVG(final\_price), 2) AS avg\_order\_value

FROM purchases

GROUP BY EXTRACT(YEAR FROM purchase\_date), EXTRACT(MONTH FROM purchase\_date)

ORDER BY year, month;

JOIN Operations:

Customer purchase history with product details:

SELECT

u.username,

u.email,

TO\_CHAR(p.purchase\_date, 'DD-MON-YYYY') AS purchase\_date,

pr.product\_name,

pr.category,

p.final\_price,

p.payment\_method

FROM purchases p

JOIN users u ON p.userid = u.userid

JOIN products pr ON p.productid = pr.productid

ORDER BY u.username, p.purchase\_date DESC;

Products never purchased (using LEFT JOIN):

SELECT

pr.productid,

pr.product\_name,

pr.category,

pr.base\_price

FROM products pr

LEFT JOIN purchases p ON pr.productid = p.productid

WHERE p.productid IS NULL;

Subqueries:

Products with above-average discounts:

SELECT

pr.productid,

pr.product\_name,

ROUND(AVG(p.discount), 2) AS avg\_discount,

COUNT(p.purchase\_id) AS times\_purchased

FROM products pr

JOIN purchases p ON pr.productid = p.productid

GROUP BY pr.productid, pr.product\_name

HAVING AVG(p.discount) > (SELECT AVG(discount) FROM purchases)

ORDER BY avg\_discount DESC;

Most recent purchase for each customer:

SELECT

u.username,

p.productid,

pr.product\_name,

p.final\_price,

TO\_CHAR(p.purchase\_date, 'DD-MON-YYYY HH24:MI:SS') AS purchase\_date

FROM purchases p

JOIN users u ON p.userid = u.userid

JOIN products pr ON p.productid = pr.productid

WHERE (p.userid, p.purchase\_date) IN (

SELECT userid, MAX(purchase\_date)

FROM purchases

GROUP BY userid

)

ORDER BY u.username;

Views for Analysis:

Create customer spending summary view:

CREATE OR REPLACE VIEW customer\_spending AS

SELECT

u.userid,

u.username,

u.email,

u.registration\_date,

COUNT(p.purchase\_id) AS total\_purchases,

NVL(SUM(p.final\_price), 0) AS total\_spent,

ROUND(NVL(AVG(p.final\_price), 0), 2) AS avg\_order\_value,

TO\_CHAR(MIN(p.purchase\_date), 'DD-MON-YYYY') AS first\_purchase,

TO\_CHAR(MAX(p.purchase\_date), 'DD-MON-YYYY') AS last\_purchase

FROM users u

LEFT JOIN purchases p ON u.userid = p.userid

GROUP BY u.userid, u.username, u.email, u.registration\_date;

Create product performance view:

CREATE OR REPLACE VIEW product\_performance AS

SELECT

pr.productid,

pr.product\_name,

pr.category,

COUNT(p.purchase\_id) AS times\_purchased,

NVL(SUM(p.final\_price), 0) AS total\_revenue,

ROUND(NVL(AVG(p.final\_price), 0), 2) AS avg\_selling\_price,

ROUND(NVL(AVG(p.discount), 0), 2) AS avg\_discount,

COUNT(DISTINCT p.userid) AS unique\_customers

FROM products pr

LEFT JOIN purchases p ON pr.productid = p.productid

GROUP BY pr.productid, pr.product\_name, pr.category;

Query Optimization with Indexes:

Create indexes for performance optimization:

CREATE INDEX idx\_purchases\_userid ON purchases(userid);

CREATE INDEX idx\_purchases\_productid ON purchases(productid);

CREATE INDEX idx\_purchases\_category ON purchases(category);

CREATE INDEX idx\_purchases\_payment\_method ON purchases(payment\_method);

CREATE INDEX idx\_purchases\_date ON purchases(purchase\_date);

CREATE INDEX idx\_purchases\_price ON purchases(final\_price);

CREATE INDEX idx\_purchases\_discount ON purchases(discount);

Composite index for date-range queries with filtering:

CREATE INDEX idx\_purchases\_date\_category ON purchases(purchase\_date, category);

Advanced Analysis Queries:

Customer cohort retention analysis:

WITH first\_purchases AS (

SELECT

userid,

TRUNC(MIN(purchase\_date), 'MONTH') AS cohort\_month

FROM purchases

GROUP BY userid

),

monthly\_activity AS (

SELECT

fp.cohort\_month,

TRUNC(p.purchase\_date, 'MONTH') AS activity\_month,

COUNT(DISTINCT p.userid) AS active\_users,

MONTHS\_BETWEEN(TRUNC(p.purchase\_date, 'MONTH'), fp.cohort\_month) AS month\_number

FROM purchases p

JOIN first\_purchases fp ON p.userid = fp.userid

GROUP BY fp.cohort\_month, TRUNC(p.purchase\_date, 'MONTH'), MONTHS\_BETWEEN(TRUNC(p.purchase\_date, 'MONTH'), fp.cohort\_month)

)

SELECT

TO\_CHAR(cohort\_month, 'YYYY-MM') AS cohort,

TO\_CHAR(activity\_month, 'YYYY-MM') AS month,

month\_number,

active\_users,

FIRST\_VALUE(active\_users) OVER (PARTITION BY cohort\_month ORDER BY activity\_month) AS cohort\_size,

ROUND(active\_users / FIRST\_VALUE(active\_users) OVER (PARTITION BY cohort\_month ORDER BY activity\_month) \* 100, 1) AS retention\_rate

FROM monthly\_activity

ORDER BY cohort\_month, activity\_month;

Discount effectiveness by product category:

SELECT

category,

CASE

WHEN discount = 0 THEN 'No discount'

WHEN discount < 0.1 THEN '0-10%'

WHEN discount < 0.2 THEN '10-20%'

WHEN discount < 0.3 THEN '20-30%'

ELSE '30%+'

END AS discount\_range,

COUNT(\*) AS transaction\_count,

SUM(final\_price) AS total\_revenue,

ROUND(AVG(final\_price), 2) AS avg\_order\_value,

COUNT(DISTINCT userid) AS unique\_customers

FROM purchases

GROUP BY category,

CASE

WHEN discount = 0 THEN 'No discount'

WHEN discount < 0.1 THEN '0-10%'

WHEN discount < 0.2 THEN '10-20%'

WHEN discount < 0.3 THEN '20-30%'

ELSE '30%+'

END

ORDER BY category, discount\_range;