RELIANCE MART ANALYSIS

(Used Advanced Queries including nested joins, CTE, sub-query, Window Functions)

Reliance Mart Dataset Description

1. Calendar.csv

- **Purpose**: Provides date-related attributes for mapping transactions over time.
- **Key Fields**:
    - date: Specific calendar date.
    - day_of_week: Name of the day (e.g., Monday).
    - month: Month number (1–12).
    - year: Year (1997 or 1998).
    - quarter: Quarter of the year (1–4)

2. Customers.csv

- **Purpose**: Holds information about customers shopping at Reliance Mart.
- **Key Fields**:
    - customer_id: Unique ID for each customer.
    - customer_name: Name of the customer.
    - gender: Gender of the customer.
    - age: Age of the customer.
    - city: City of residence.
    - state: State of residence.

3. Products.csv

- **Purpose**: Contains information about the products available at the stores.
- **Key Fields**:
    - product_id: Unique ID for each product.
    - product_name: Name of the product.
    - category: Product category (e.g., electronics, grocery).
    - subcategory: Specific product subcategory.
    - price: Retail price of the product.

4. Regions.csv

- **Purpose**: Defines regional classification for the stores.
- **Key Fields**:
    - region_id: Unique ID for each region.
    - region_name: Name of the region (e.g., North, South).

5. Returns-1997-1998.csv

- **Purpose**: Captures returned transactions by customers.
- **Key Fields**:
    - return_id: Unique ID for the return.
    - transaction_id: Related transaction that was returned.
    - return_date: Date when return happened.
    - return_reason: Reason provided for the return.

6. Stores.csv

- **Purpose**: Contains metadata about the physical store locations.
- **Key Fields**:
    - store_id: Unique ID for each store.
    - store_name: Name of the store.
    - region_id: Foreign key linking to the Regions table.
    - city: City where the store is located.
    - state: State where the store is located.

7. Transactions-1997.csv and Transactions-1998.csv

- **Purpose**: Logs of customer purchase activities in 1997 and 1998.
- **Key Fields**:
    - transaction_id: Unique ID for each transaction.
    - customer_id: Customer who made the purchase.
    - store_id: Store where the transaction occurred.
    - product_id: Product that was purchased.
    - quantity: Quantity of products bought.
    - sales_amount: Total sales amount (price × quantity).
    - transaction_date: Date of the transaction.

```
create database Reliance_Mart;
select * from customers;
select * from products;
select * from regions;
select * from returns_1997_1998;
select * from stores;
select * from transactions_1997;
select * from transactions_1998;
```

1. Monthly Sales Rank by Store and Product Category

-- Get the monthly sales ranking of each product category per store for the year 1998 using a window function.

```
with store_quantity as (
select t.store_id, sum(t.quantity) as quantity
from products p
inner join
(
select * from transactions_1997
union all
select * from transactions_1998
) t
on p.product_id = t.product_id
group by 1)
select *,row_number() over (
order by quantity desc) as sales_rank
from store_quantity ;
```

	store_id	quantity	sales_rank
▶	13	80762	1
	17	74347	2
	15	54865	3
	11	54526	4
	16	52489	5
	7	52475	6
	24	52417	7
	3	51914	8
	6	46129	9
	12	41808	10
	8	40994	11
	19	40152	12
	21	38608	13
	10	27334	14

2. Find Products with Increasing Monthly Sales Trend in 1998

-- Identify products whose monthly sales showed a strictly increasing pattern using `LAG()` and `SUM()`.

```
with product_trans_1998 as (
select
p.product_id,p.product_cost,p.product_name,t.transaction_date,t.
quantity
from products p
inner join transactions_1998 t
on p.product_id = t.product_id),
product_sales as (
select product_id, product_name, month(transaction_date) as
months, sum(quantity) as monthly_quantity
from product_trans_1998
group by 1,2,3),
sales_performance as (select *,
(monthly_quantity - lag(monthly_quantity,1,0) over (partition by
product_id order by months)) as monthly_sale_performance
from product_sales)
select product_id, product_name, sum(monthly_sale_performance)
as overall_performance
from sales_performance
group by 1,2
order by overall_performance desc;
```

product_id	product_name	overall_performance
1292	Booker Low Fat Cottage Cheese	74
304	Super Grape Jam	72
1012	American Sliced Ham	72
1373	Hilltop 200 MG Ibuprofen	71
1423	Hermanos Sweet Peas	68
391	Urban Large Eggs	67
988	Even Better Havarti Cheese	67
24	Blue Label Regular Ramen Soup	66
266	Good Imported Beer	66
179	High Top Green Pepper	65
545	Fast Salted Pretzels	65
617	Landslide Strawberry Jam	65
1390	Sunset 75 Watt Lightbulb	65
1521	Top Measure White Zinfandel ...	65

3. Detect Anomalies in Returns: Stores with Unusually High Return Rate

-- Use a subquery to calculate average return rate per store and flag those above 2x the global average.

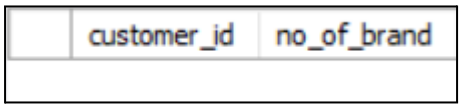
```
with returned as (select r.store_id, sum(r.quantity) as returned
from returns_1997_1998 r
inner join stores s
on r.store_id = s.store_id
group by 1),
avg_return as (
select round(avg(returned),2) as avg_return
from returned)
select *
from returned
cross join avg_return
where returned > 1.5*avg_return;
```

store_id	returned	avg_return
13	736	414.45
17	764	414.45

4. List Customers Who Made Purchases Across All Product Categories

-- Use a `HAVING` clause and `COUNT(DISTINCT category)` approach to find such customers.

```
with transactions as (
select * from transactions_1997
union all
select * from transactions_1998)
select t.customer_id, count(distinct p.product_brand) as
no_of_brand
from transactions t
inner join products p
on t.product_id = p.product_id
group by 1
having no_of_brand > (select count(distinct product_brand) from
products);
```



	customer_id	no_of_brand

5. Year-over-Year Sales Growth by Region and Product Category

-- Join 1997 and 1998 data to compute percentage growth for region-category pairs.

```
with trans as (
select * from transactions_1997
union all
select * from transactions_1998 ),
yoy as (select product_id, year(transaction_date) as years,
sum(quantity) as quantity
from trans
group by 1,2),
yoy_cte as (
select product_id, years, quantity,
lag(quantity,1,0) over (partition by product_id order by years)
as yoy_growth
from yoy)
select product_id, (quantity - yoy_growth)*100/yoy_growth as
perc_growth
from yoy_cte
where years = 1998;
```

product_id	perc_growth
1	95.1807
2	75.4386
3	96.0265
4	96.1290
5	128.3582
6	94.3820
7	65.0000
8	68.3938
9	160.7692
10	87.0370
11	35.2941
12	91.7526
13	102.5157
14	151.9380

6. Monthly Average Basket Size by Store (Number of Products per Transaction)

-- Use `AVG()` and window functions partitioned by store and month.

```
with transactions as (
select * from transactions_1997
union all
select * from transactions_1998),
cte_trans_store as (select month(t.transaction_date) as months,
s.store_id, count( distinct t.product_id) as products
from transactions t
inner join stores s
on t.store_id = s.store_id
group by 1,2),
product_store as (select store_id, avg(products) as avg_basket
from cte_trans_store
group by 1)
select *
from product_store;
```

store_id	avg_basket
1	570.5000
2	216.6667
3	916.5000
4	558.4167
5	118.0000
6	855.8333
7	926.3333
8	788.5000
9	289.2500
10	584.4167
11	944.0833
12	797.9167
13	1160.0000
14	217.1667

7. Find the First and Last Transaction Date per Customer

-- Use `MIN()` and `MAX()` with `GROUP BY` or window function.

```
with transactions as (
select * from transactions_1997
union all
select * from transactions_1998)
select t.customer_id, max(t.transaction_date) as max_date,
min(t.transaction_date) as min_date
from transactions t
inner join customers c
on t.customer_id = c.customer_id
group by 1;
```

customer_id	max_date	min_date
3	1998-04-17	1997-04-27
5	1997-01-04	1997-01-04
6	1998-12-16	1997-07-23
8	1998-11-27	1998-03-02
9	1998-10-09	1998-04-27
10	1998-12-23	1997-01-20
11	1998-07-20	1998-07-20
12	1998-12-08	1998-12-08
14	1998-12-25	1997-07-12
17	1997-12-16	1997-08-01
18	1998-01-17	1998-01-17
19	1997-07-06	1997-07-06
20	1998-11-02	1997-01-06
21	1998-11-18	1997-03-15

8. Repeat Customers: Bought Same Product in Both 1997 and 1998

-- Use an `INTERSECT` or join to compare customer-product pairs in both years.

```
with cte_1997 as (select t.transaction_date, p.product_id,
c.customer_id
from transactions_1997 t
inner join products p
on t.product_id = p.product_id
inner join customers c
on t.customer_id = c.customer_id),
cte_1998 as (
select t.transaction_date, p.product_id, c.customer_id
from transactions_1998 t
inner join products p
on t.product_id = p.product_id
inner join customers c
on t.customer_id = c.customer_id)
select c1.product_id, c1.customer_id,
c1.transaction_date, c2.transaction_date
from cte_1997 c1
inner join cte_1998 c2
on c1.product_id = c2.product_id and
c1.customer_id = c2.customer_id ;
```

product_id	customer_id	transaction_date	transaction_date
679	1706	1997-01-07	1998-01-02
1222	1706	1997-01-07	1998-01-02
493	2663	1997-01-13	1998-01-04
993	1533	1997-01-21	1998-01-10
1452	2769	1997-01-21	1998-01-21
1247	924	1997-01-21	1998-02-03
1406	4194	1997-01-15	1998-02-06
872	2086	1997-01-20	1998-02-10
1398	2086	1997-01-20	1998-02-10
832	769	1997-01-21	1998-02-13
931	3549	1997-01-11	1998-02-15
813	543	1997-01-07	1998-02-17
30	221	1997-01-13	1998-02-21
375	2201	1997-01-21	1998-02-21

9. Customer Churn Analysis: Customers Who Stopped Buying in 1998

-- Identify customers who made purchases in 1997 but not in 1998 using `NOT EXISTS`.

```
with cte_1997 as (
select distinct customer_id
from transactions_1997),
cte_1998 as (
select distinct customer_id
from transactions_1998)
select *
from cte_1997 t1
left join cte_1998 t2
on t1.customer_id = t2.customer_id
where t2.customer_id is null;
```

customer_id	customer_id
5262	NULL
3175	NULL
6062	NULL
6613	NULL
5005	NULL
7962	NULL
1215	NULL
255	NULL
2869	NULL
4016	NULL
2208	NULL
8648	NULL
5	NULL
8762	NULL

10. Top Product per Store by Total Revenue (1998 Only)

-- Use `RANK()` to find the highest-selling product per store.

```
with cte1 as (select t.store_id, t.product_id, count(t.quantity)
as quantities
from transactions_1998 t
inner join stores s
on t.store_id  = s.store_id
group by 1,2),
cte2 as (
select store_id, product_id, max(quantities) as max_sold
from cte1
group by 1,2),
store_rank as (select store_id,
product_id,
max_sold,
rank() over (partition by product_id order by max_sold desc) as
ranks
from cte2)
select store_id, product_id, max_sold, ranks
from store_rank
where ranks = 1;
```

store_id	product_id	max_sold	ranks
16	1	7	1
17	2	9	1
21	2	9	1
15	3	10	1
19	4	9	1
8	4	9	1
13	5	10	1
17	6	12	1
8	7	11	1
21	8	8	1
8	8	8	1
19	8	8	1
12	9	9	1
1	9	9	1