Lab 2: Assignment_2, SQL Queries

Lab – 2 13-Aug-2024	Assignment_2, SQL Queries
IT615 Database Management System, Autumn'2024; Instructor: minal_bhise@daiict	

Objectives: I) Run complex queries.
II) Using & Updating VIEWS.

Submission: Each student needs to upload a single .pdf file which will contain the following things for all the queries.

1. English query and SQL Query in the given sequence.
2. Screenshot of results.
3. Count of tuples in the results.

I. Run sample queries – Consider E-Commerce Schema for the below examples:

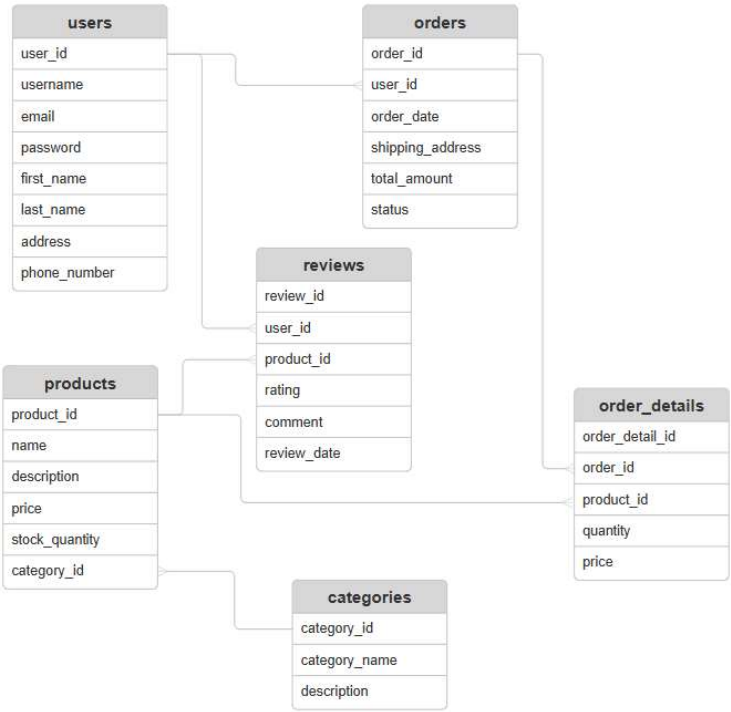

Figure: E-Commerce Schema

1. **Use of GROUP BY:**

   Ex. Find the number of products in each category.

   SELECT COUNT(*), category_name FROM "EC_DB".products JOIN "EC_DB".categories ON products.category_id = categories.category_id GROUP BY category_name;

	count bigint	category_name character varying (100)
1	5	Sports & Fitness
2	5	Home Appliances
3	5	Electronics
4	5	Beauty & Personal Care
5	5	Books
6	5	Fashion
7	5	Automotive
8	5	Furniture
9	5	Toys & Baby Products
10	5	Groceries

Total rows: 10 of 10 Query complete 00:00:00.791

2. **Use of JOIN & ORDER BY to find a product having maximum price with details:**

   Ex. Find product name and the highest price with category.

   SELECT products.name, products.price, category_name

   FROM "EC_DB".products

   JOIN "EC_DB".categories ON products.category_id = categories.category_id

   ORDER BY products.price DESC

   LIMIT 1;

	name character varying (100)	price numeric (10,2)	category_name character varying (100)
1	Laptop	56000.00	Electronics

3. **Use of SUBQUERY to find a product having maximum price with details:**

   Step 1: Write a subquery to get the maximum price.

   SELECT MAX(price) FROM "EC_DB".products;

   Step 2: Use the subquery to get the product ID with the maximum price.

   SELECT product_id, price FROM "EC_DB".products

   WHERE price = (SELECT MAX(price) FROM "EC_DB".products);

	product_id [PK] integer	price numeric (10,2)
1	2	56000.00

   Step 3: Write a subquery as a part of the main query having joins.

   SELECT products.name, products.price, categories.category_name

   FROM "EC_DB".products

   JOIN "EC_DB".categories ON products.category_id = categories.category_id

   WHERE products.price = (SELECT MAX(price) FROM "EC_DB".products);

	name character varying (100)	price numeric (10,2)	category_name character varying (100)
1	Laptop	56000.00	Electronics

4. **Use of LIMIT, ORDER BY & SUBQUERY as Tables to find a product having maximum price with details:**

   Step 1: Write a subquery to order by price and limit the result.

   SELECT product_id, price

   FROM "EC_DB".products

   ORDER BY price DESC

   LIMIT 1;

	product_id [PK] integer	price numeric (10,2)
1	2	56000.00

Step 2: Join the above query with another table.

```
SELECT products.name, T1.price
FROM "EC_DB".products
JOIN (
    SELECT product_id, price
    FROM "EC_DB".products
    ORDER BY price DESC
    LIMIT 1
) AS T1 ON products.product_id = T1.product_id;
```

	name character varying (100)	price numeric (10,2)
1	Laptop	56000.00

II. Run sample queries with VIEWs.

1. Use of a View as Tables to find a product having maximum price with details:

Step 1: Create a view.

```
CREATE OR REPLACE VIEW "EC_DB".MaxPrice_Product_VIEW AS
SELECT product_id, price
FROM "EC_DB".products
ORDER BY price DESC
LIMIT 1;
```

Step 2: Use the view as a table.

```
SELECT products.name, MaxPrice_Product_VIEW.price
```

FROM "EC_DB".products

JOIN "EC_DB".MaxPrice_Product_VIEW ON products.product_id = MaxPrice_Product_VIEW.product_id;

2. **Auto-updates of VIEWs:**

Step 1: Create a view for the average price.

CREATE OR REPLACE VIEW "EC_DB".AVG_Price_VIEW AS

SELECT AVG(price) AS avg_price

FROM "EC_DB".products;

Data Output Messages Notifications

```
CREATE VIEW

Query returned successfully in 320 msec.
```

Step 2: Insert a new record and see if the view updates.

INSERT INTO "EC_DB".products (product_id, name, description, price, stock_quantity, category_id)

VALUES (501, 'New Product', 'Description of new product', 1500, 50, 1);

*SELECT * FROM "EC_DB".AVG_Price_VIEW;*

The view will automatically update the data.

3. **Updating VIEW** data **manually**:

Step 1: Create a simple view.

CREATE OR REPLACE VIEW "EC_DB".VIEW_PRODUCTS AS

*SELECT * FROM "EC_DB".products;*

Data Output Messages Notifications

```
CREATE VIEW

Query returned successfully in 325 msec.
```

Step 2: Insert new records into the view.

INSERT INTO "EC_DB".VIEW_PRODUCTS (product_id, name, description, price, stock_quantity, category_id)

VALUES (502, 'Another New Product', 'Description of another new product', 2000, 30, 2);

*SELECT * FROM "EC_DB".products ORDER BY product_id ASC;*

	product_id [PK] integer	name character varying (100)	description text	price numeric (10,2)	stock_quantity integer	category_id integer
1	1	Mobile Phone	Portable communication device with internet access and app...	7498.65	28	1
2	2	Laptop	Compact computer for personal and professional use.	56000.00	69	1
3	3	Smart Watch	Wearable device for tracking fitness and notifications.	2784.21	72	1
4	4	Headphones	Audio device for listening to music or calls privately.	6992.10	18	1
5	5	Tablet	Touchscreen device for media consumption and productivity.	1181.34	95	1
6	6	Shirt	Upper body garment with sleeves and a collar.	5909.62	32	2
7	7	Dress	One-piece garment for formal or casual wear.	602.58	68	2
8	8	Shoes	Footwear for protection and style.	2334.15	30	2
9	9	Jacket	Outerwear garment for warmth and protection.	619.66	53	2
10	10	Jeans	Durable pants made from denim fabric.	3369.91	61	2
11	11	Air Conditioner	Device for cooling and controlling indoor climate.	1095.39	99	3

Total rows: 52 of 52 Query complete 00:00:00.339

4. **Updating Partial VIEW of a Table which has not null constraint on some attributes:**

Step 1: Create a view including PK and FK attributes.

CREATE OR REPLACE VIEW "EC_DB".VIEW2_PRODUCTS AS

SELECT product_id, price FROM "EC_DB".products;

Data Output Messages Notifications

```
CREATE VIEW

Query returned successfully in 351 msec.
```

Step 2: Insert new records into the view.

INSERT INTO VIEW2_PRODUCTS (product_id, price)

VALUES (503, 2500);

```
ERROR:  Failing row contains (503, null, null, 2500.00, null, null).null value in column "name" of relation "products" violates not-null constraint

ERROR:  null value in column "name" of relation "products" violates not-null constraint
SQL state: 23502
Detail: Failing row contains (503, null, null, 2500.00, null, null).
```

Generates error because some attributes have NOT NULL constraint.

III. Run Complex Queries

Q1. Find the name, price and stock of the most expensive product.

Q2. Get the total number of orders placed by each user.

Q3. Find the product with the highest number of reviews.

Q4. Get the average rating of each product.

Q5. Get the total revenue generated by each category.

Q6. Find the most popular product (most ordered).

Q7. List users who have spent more than $1000 in total.

Q8. Get the details of the largest order (by total amount).

Q9. List all products that have never been ordered.

Q10. Find the most active user (by the number of reviews).

Q11. Get the average order amount per user.

Q12. List all users who have not placed any orders.

Q13. Find the most recent review for each product.

Q14. Get the list of users who have reviewed all products they have purchased.

Q15. Create a view that shows the total amount spent by each user.

Q16. Get the average price of products in each category that have been reviewed.

Q17. List all users who have placed more than 2 orders.

Q18. Find users who have made purchases from all categories.

Q19. List all users who have placed orders in the last 30 days but haven't made a purchase in the last 7 days.

Q20. Identify the products with a stock quantity below the average stock level.

Q21. Create a view ‘**product_avg_rating**’ to show the average rating of each product.

Q22. List products with an average rating above 4.5 using the '**product_avg_rating'** view.

Q23. Create a view '**product_review_count'** to show the number of reviews per product.

Q24. List products with more than 10 reviews using the '**product_review_count'** view.

Q25. Create a view '**recent_orders'** to show all orders placed in the last 30 days.

Q26. List the most expensive orders placed in the last 30 days using the '**recent_orders'** view.

Q27. Create a view '**product_sales_quantity'** to show the total quantity sold for each product.

Q28. List the top 5 best-selling products using the '**product_sales_quantity'** view.

Q29. Create a view '**product_sales_quantity'** to show the total number of orders each user has placed.

Q30. List users who have placed more than 5 orders using the '**user_order_count'** view.