JENSON USA

WE KEEP YOU PEDALING

“SQL DATASET ANALYSIS”

MySQL

INTRODUCTION

- Jenson USA is one of the leading retailers of bicycles, parts, and accessories.
- Founded in 1994, it started as a small bike shop and has grown into a trusted online and offline cycling store.
- The company focuses on providing quality products and excellent customer service to cycling enthusiasts.
- With thousands of products and loyal customers, Jenson USA continues to be a key player in the cycling industry.

MySQL

OBJECTIVE

- ***To analyze the Jenson USA dataset using SQL.***
- ***To identify top customers, best-selling products, and categories.***
- ***To evaluate sales performance of staff and stores.***
- ***To detect unsold products and sales trends.***
- ***To provide data-driven insights for better decisions.***

FIND THE TOTAL NUMBER OF PRODUCTS SOLD BY EACH STORE ALONG WITH THE STORE NAME.

```
2
3   select   stores.store_name,
4              sum(order_items.quantity)        total_products_sold
5            from stores
6        join orders  on stores.store_id = orders.store_id
7       join order_items  on orders.order_id = order_items.order_id
8    group by stores.store_name;
9
10
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

store_name	total_products_sold
Santa Cruz Bikes	1516
Baldwin Bikes	4779
Rowlett Bikes	783

- JOIN → Combined data from stores, orders, and order_items.
- SUM() → Calculated the total products sold.
- GROUP BY → Summarized results store-wise.

MySQL

CALCULATE THE CUMULATIVE SUM OF QUANTITIES SOLD FOR EACH PRODUCT OVER TIME.

```
13 •  select  products.product_id, orders.order_date,quantity,
14        sum(order_items.quantity)
15        over
16        (partition by  products.product_id order by orders.order_date)  cumulative_quantity
17
18             from products
19            join order_items  on products.product_id = order_items.product_id
20            join orders  on order_items.order_id = orders.order_id;
21
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_id	order_date	quantity	cumulative_quantity
2	2016-01-03	2	2
2	2016-01-14	2	4
2	2016-01-18	1	5
2	2016-02-05	1	6
2	2016-02-09	1	7
2	2016-02-26	1	8
2	2016-02-28	1	10
2	2016-02-28	1	10
2	2016-03-08	1	11
2	2016-03-14	2	13

- JOIN → Linked products, orders & order_items.
- SUM() OVER → Used window function.
- Cumulative Quantity → Shows running total (date-wise).

MySQL

FIND THE PRODUCT WITH THE HIGHEST TOTAL SALES FOR EACH CATEGORY.

```sql
with a as (
  select
    categories.category_name,
    products.product_id,
    products.product_name,
    sum(order_items.quantity * (order_items.list_price - order_items.discount)) as highest_sales,
    rank() over (
      partition by categories.category_name
      order by sum(order_items.quantity * (order_items.list_price - order_items.discount)) desc
    ) as rnk
  from categories
  join products
    on categories.category_id = products.category_id
  join order_items
    on order_items.product_id = products.product_id
  group by categories.category_name, products.product_id, products.product_name
)
select category_name, product_id, product_name, highest_sales, rnk
from a
where rnk = 1;
```

We want to find the top-selling product in each category.

First, we calculate total sales of every product = (Quantity × (Price – Discount))

Then, we rank products inside each category by their sales (highest first).

Finally, we take only Rank = 1, which gives the best product of that category.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category_name	product_id	product_name	highest_sales	rnk
Children Bicycles	23	Electra Girl's Hawaii 1 (20-inch) - 2015/2016	4619278.00	1
Comfort Bicycles	26	Electra Townie Original 7D EQ - 2016	8039320.00	1
Cruisers Bicycles	16	Electra Townie Original 7D EQ - 2016	9359304.00	1
Cyclocross Bicycles	11	Surly Straggler 650b - 2016	25382383.00	1
Electric Bikes	9	Trek Conduit+ - 2016	43499347.00	1
Mountain Bikes	7	Trek Slash 8 275 - 2016	61599226.00	1
Road Bikes	56	Trek Domane SLR 6 Disc - 2017	23649774.00	1

MySQL

FIND THE CUSTOMER WHO SPENT THE MOST MONEY ON ORDERS.

```sql
with a as (
  select
    concat(customers.first_name, " ", customers.last_name)     as  Customer_Name,
    orders.order_id,
    sum(order_items.quantity * (order_items.list_price - order_items.discount)) as sales
  from orders
    join customers on orders.customer_id = customers.customer_id
    join order_items on orders.order_id = order_items.order_id
  group by concat(customers.first_name, " ", customers.last_name), orders.order_id
)
select *
from (
  select *, rank() over(order by sales desc) as rnk
  from a
) temp
where rnk = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer_Name	order_id	sales	rnk
Jacquline Duncan	1541	3232863.00	1

- JOIN → Combined customers, orders, and order_items tables.
- SUM() → Calculated total sales amount per customer order.
- RANK() OVER (ORDER BY … DESC) → Ranked customers by highest spend.
- WHERE rank = 1 → Selected the top spender.

MySQL

FIND THE HIGHEST-PRICED PRODUCT FOR EACH CATEGORY NAME.

```sql
select *
from (
    select
        categories.category_id,
        categories.category_name,
        products.product_name,
        products.list_price,
        rank() over (
            partition by categories.category_id
            order by products.list_price desc)     as rnk
    from categories
    join products on categories.category_id = products.category_id) as rank_product

    where rnk = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

category_id	category_name	product_name	list_price	rnk
1	Children Bicycles	Electra Straight 8 3i (20-inch) - Boy's - 2017	48999.00	1
1	Children Bicycles	Electra Townie 3i EQ (20-inch) - Boys' - 2017	48999.00	1
1	Children Bicycles	Trek Superfly 24 - 2017/2018	48999.00	1
2	Comfort Bicycles	Electra Townie Go! 8i - 2017/2018	259999.00	1
3	Cruisers Bicycles	Electra Townie Commute Go! - 2018	299999.00	1
3	Cruisers Bicycles	Electra Townie Commute Go! Ladies' - 2018	299999.00	1
4	Cyclocross Bicycles	Trek Boone 7 Disc - 2018	399999.00	1

Result 6

- JOIN → Linked categories with products.
- RANK() with PARTITION BY → Ranked products within each category.
- ORDER BY list_price DESC → Sorted by highest price.
- WHERE rnk = 1 → Selected the top-priced product per category.

MySQL

FIND THE TOTAL NUMBER OF ORDERS PLACED BY EACH CUSTOMER PER STORE.

```
96 •    select
97          concat(customers.first_name, ' ', customers.last_name) as  cust_name,
98        stores.store_name,
99        COUNT(orders.order_id) as   total_orders
100   from customers
101       join orders on  customers.customer_id = orders.customer_id
102       join stores   on stores.store_id = orders.store_id
103   group by
104       concat(customers.first_name, ' ', customers.last_name),
105       stores.store_name,
106       stores.store_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

cust_name	store_name	total_orders
Earl Stanley	Santa Cruz Bikes	1
Marquerite Dawson	Santa Cruz Bikes	2
Damien Dorsey	Santa Cruz Bikes	2
Arvilla Osborn	Santa Cruz Bikes	2
Erma Salinas	Santa Cruz Bikes	1
Felicidad Golden	Santa Cruz Bikes	1
Chanel May	Santa Cruz Bikes	1

Result 8

- JOIN → Combined customers, orders, and stores tables.
- COUNT(order_id) → Counted total number of orders.
- GROUP BY customer & store → Calculated orders per customer per store.

MySQL

FIND THE NAMES OF STAFF MEMBERS WHO HAVE NOT MADE ANY SALES.

```
111 •    SELECT
112         staffs.staff_id,
113         CONCAT(staffs.first_name, ' ', staffs.last_name) AS full_name
114       FROM staffs
115          WHERE NOT EXISTS (
116    SELECT *
117         FROM orders
118              WHERE orders.staff_id = staffs.staff_id
119    );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

staff_id	full_name
1	Fabiola Jackson
4	Virgie Wiggins
5	Jannette David
10	Bernardine Houston

Result 9

- NOT EXISTS → Checked which staff do not have any orders in the orders table.
- CONCAT() → Merged first name and last name into a single full_name column.
- WHERE → Filtered staff based on the absence of orders.

MySQL

FIND THE TOP 3 MOST SOLD PRODUCTS IN TERMS OF QUANTITY.

```sql
124
125 •  select product_name, product_id
126       from (
127        select
128            products.product_id,
129        products.product_name,
130            sum(order_items.quantity)   as  total_quantity,
131            rank() over (order by sum(order_items.quantity) desc) as rnk
132
133        from order_items
134        join products ON products.product_id = order_items.product_id
135
136  group by products.product_id, products.product_name)   as temp
137     where rnk <= 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_name	product_id
Surly Ice Cream Truck Frameset - 2016	6
Electra Cruiser 1 (24-Inch) - 2016	13
Electra Townie Original 7D EQ - 2016	16

- JOIN → Combined order_items with products to get product details for each order.
- SUM(quantity) → Calculated total quantity sold per product.
- RANK() → Ranked products based on total sales quantity and selected top 3 products.

MySQL

FIND THE MEDIAN VALUE OF THE PRICE LIST.

```sql
select
        avg(list_price) AS median_price
from (
    select list_price,
           ROW_NUMBER() OVER (ORDER BY list_price) AS row_num,
           COUNT(*) OVER () AS total_count
    from products
) t
WHERE row_num IN ( (total_count + 1)/2 , (total_count + 2)/2 );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

median_price
74999.000000

- *ROW_NUMBER() & COUNT() → Assigned sequential row numbers to products ordered by price and calculated total number of products.*
- *WHERE row_num IN (…) → Selected the middle row(s) to handle odd or even number of products.*
- *AVG(list_price) → Calculated the median price by averaging the middle value(s).*

"This method calculate median in SQL since most database don't have a built-in MEDIAN() FUNCTION."

MySQL

LIST ALL PRODUCTS THAT HAVE NEVER BEEN ORDERED.

```sql
select
        products.product_id,
    products.product_name
FROM products
   where not exists (
      select order_items.product_id
     from order_items
where order_items.product_id = products.product_id
);
```

Result Grid | Filter Rows: | Edit: | Export/Import:

product_id	product_name
1	Trek 820 - 2016
121	Surly Krampus Frameset - 2018
125	Trek Kids' Dual Sport - 2018
154	Trek Domane SLR 6 Disc Women's - 2018
195	Electra Townie Go! 8i Ladies' - 2018
267	Trek Precaliber 12 Girl's - 2018
284	Electra Savannah 1 (20-inch) - Girl's - 2018
291	Electra Sweet Ride 1 (20-inch) - Girl's - 2018
316	Trek Checkpoint ALR 4 Women's - 2019
317	Trek Checkpoint ALR 5 - 2019

products 12

- NOT EXISTS → Checked which products have never been sold (no matching records in order_items).
- WHERE → Filtered the products based on the absence of orders.
- SELECT & product info → Returned product_id and product_name of unsold products.

LIST THE NAMES OF STAFF MEMBERS WHO HAVE MADE MORE SALES THAN THE AVERAGE NUMBER OF SALES BY ALL STAFF MEMBERS.

```sql
with a as (
    select
        concat(staffs.first_name, ' ', staffs.last_name) as full_name,
        sum(order_items.quantity * order_items.list_price) as Sales
    from staffs
    left join orders
        on staffs.staff_id = orders.staff_id
    left join order_items
        on orders.order_id = order_items.order_id
    group by concat(staffs.first_name, ' ', staffs.last_name)
)
select *
from a
where Sales > (select avg(Sales) from a);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

full_name	Sales
Marcelene Boyer	293888873.00
Venita Daniel	288735348.00

- Calculate total sales for each staff using SUM(quantity * list_price).
- Use GROUP BY to aggregate sales by staff name.
- Compare each staff's sales with the overall average sales.
- Display only staff whose sales are above the average.

MySQL

IDENTIFY THE CUSTOMERS WHO HAVE ORDERED ALL TYPES OF PRODUCTS.

```
152 •  select  customers.customer_id
153        from customers
154         join orders ON customers.customer_id = orders.customer_id
155       join order_items ON orders.order_id = order_items.order_id
156        join products p ON p.product_id = order_items.product_id
157    group by customers.customer_id
158    having count(distinct p.category_id) = (
159        select count(categories.category_id)
160          from categories
161    );
162
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id
9

- *JOIN → Combined customers, orders, order_items, and products to link customers to the products they purchased.*
- *GROUP BY & HAVING → Grouped by customer and filtered those who bought products from all categories.*
- *COUNT & Subquery → Compared the number of distinct categories bought by each customer to the total number of categories in the categories table.*

MySQL

CONCLUSION

- Analysis of the Jenson USA dataset using SQL revealed top-performing and underperforming stores.
- Identified best-selling products, unsold products, and sales trends.
- Highlighted top-spending customers and loyal buyers.
- Calculated pricing insights like highest-priced items and median price.
- Overall, the project showed how SQL can turn raw data into meaningful business insights to support better decisions.

THANK YOU!

JENSON USA
WE KEEP YOU PEDALING

MySQL

SUBMITTED BY : GAJENDRA SINGH PANWAR