SQL Important Syntaxes

JOINS, WINDOW Functions, CASE Statement

Prerequisites

```
SELECT order_date  SUM(total_revenue)

FROM restaurant_1_orders

WHERE

Order_date BETWEEN  ‘2022-01-01’ AND ‘2022-01-31’

GROUP BY 1

ORDER BY 1
```

Inner Join

left table

id	val_l
1	20
2	30
3	40
4	50

right table

id	val_r
1	10
3	20
4	25
6	50

id	val_l	val_r
1	20	10
3	40	20
4	50	25

Left Join

left table

id	val_l
1	20
2	30
3	40
4	50

right table

id	val_r
1	10
3	20
4	25
6	50

id	val_l	val_r
1	20	10
2	30	-
3	40	20
4	50	25

Right Join

left table

id	val_l
1	20
2	30
3	40
4	50

right table

id	val_r
1	10
3	20
4	25
6	50

id	val_r	val_l
1	10	20
3	20	40
4	25	50

Full Join/Full Outer Join

left table

id	val_l
1	20
2	30
3	40
4	50

right table

id	val_r
1	10
3	20
4	25
6	50

id	val_l	val_r
1	20	10
2	30	-
3	40	20
4	50	25
6	-	50

Inner Join
- Syntax and query

Note: table.column_name format must be used in order to avoid ambiguous column name error

```
SELECT restaurant_1_orders.order_date
SUM(restaurant_1_orders.total_revenue)

FROM restaurant_1_orders

INNER JOIN/JOIN restaurant_2_orders

ON

restaurant_1_orders.order_date =
restaurant_2_orders.order_date

WHERE

Order_date BETWEEN  ‘2022-01-01’ AND ‘2022-01-31’

GROUP BY 1

ORDER BY 1
```

Inner Join using Alias

Note: table.column_name format must be used in order to avoid ambiguous column name error

```
SELECT res1_ord.order_date
SUM(res1_ord.total_revenue)

FROM restaurant_1_orders res1_ord

INNER JOIN/JOIN restaurant_2_orders res2_ord

ON

res1_ord.order_date = res2_ord.order_date

WHERE

Order_date BETWEEN  ‘2022-01-01’ AND ‘2022-01-31’

GROUP BY 1

ORDER BY 1
```

Inner Join with USING keyword

Note: table.column_name format must be used in order to avoid ambiguous column name error

```
SELECT order_date  SUM(total_revenue)

FROM restaurant_1_orders

INNER JOIN/JOIN restaurant_2_orders

USING (order_date)

WHERE

Order_date BETWEEN  ‘2022-01-01’ AND ‘2022-01-31’

GROUP BY 1

ORDER BY 1
```

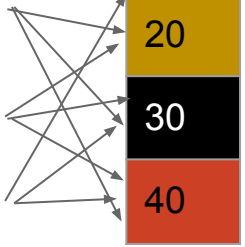
CROSS JOIN

table 1

id1
1
2
3

table 2

id2
20
30
40



id1	id2
1	20
1	30
1	40
2	20
2	30
2	40
3	20
3	30
3	40

CROSS JOIN - Syntax and query

```
SELECT res1_ord.order_number,
res2_ord.order_number

FROM

    restaurant_1_orders res1_ord

CROSS JOIN

    restaurant_2_orders res2_ord
```

CROSS JOIN - Syntax and query

```
SELECT res1_ord.order_number, res2_ord.order_number
FROM
    restaurant_1_orders res1_ord
CROSS JOIN
    restaurant_2_orders res2_ord
WHERE
    res1_ord.order_number IN (16118)
    AND res1_ord.order_number IN (25583)
```

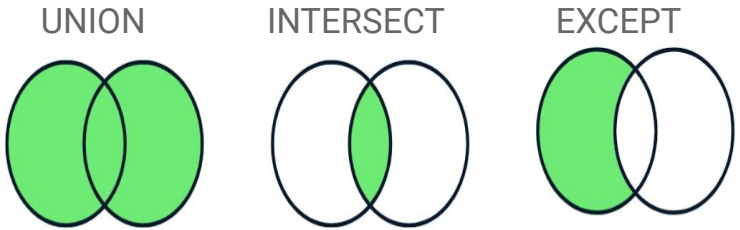
SET Theory, Venn diagram and SQL syntaxes

UNION

UNION ALL

INTERSECT

EXCEPT



UNION/UNION ALL

```
SELECT *
FROM restaurant_1_orders res1_ord
UNION/UNION ALL
SELECT *,
FROM    restaurant_2_orders res2_ord
```

INTERSECT

```
SELECT *
FROM restaurant_1_orders res1_ord
INTERSECT
SELECT *,
FROM    restaurant_2_orders res2_ord
```

EXCEPT

All data that are present in the left table and not present on the right table

```
SELECT *

FROM restaurant_1_orders res1_ord

INTERSECT

SELECT *,

FROM    restaurant_2_orders res2_ord
```

CASE STATEMENTS

Contains a **WHEN , THEN , ELSE** statement.

Finish it giving a name using keyword **END**

```
CASE  WHEN res1_ord.total_revenue_per_item > 5.0  THEN 'hot_dish_item'

 WHEN res1_ord.total_revenue_per_item < 5.0 THEN 'cold_dish_item'

ELSE 'medium_dish_item'  END AS

 item_hotness_category_based_on_rev
```

CASE STATEMENTS – Aggregation

```
SELECT

res1_ord.order_date, res1_ord.item_name,
res1_ord.month, res1_ord.day_name,

COUNT/SUM(

CASE  WHEN res1_ord.total_revenue_per_item > 5.0
THEN 1

ELSE 0 END)  AS

 max_rev_generating_items_count

GROUP BY

1,2,3,4
```

Window Functions

WHY do we need ?

Aggregate values requires to use **GROUP BY** to get values based on all non-aggregate columns.

What if you want to compare aggregate values to non-aggregate data.

WINDOW FUNCTIONS!!

Industry Uses of Window Functions

Window functions perform calculations on already generated result set like a window and thus the name.

How used in Businesses?

Rankings

Moving Averages

Running Totals

Reducing False Positives (Randomization)

Why window function?

What was the revenue per item in 2019 and how can we compare that to the avg?

```
SELECT  order_date,item_name,
total_revenue_per_item,

( SELECT

        AVG(total_revenue_per_item)

FROM restaurant_1_orders

WHERE order_date >= ‘2019-01-01’ ) avg_rev

FROM  restaurant_1_orders

WHERE order_date >= ‘2019-01-01’

)
```

Why window function?

What was the revenue per item in 2019 and how can we compare that to the avg?

```
SELECT  order_date,item_name,
total_revenue_per_item,

AVG(total_revenue_per_item) OVER() AS
total_revenue_avg

FROM restaurant_1_orders

WHERE order_date >= '2019-01-01'
```

RANK

What is the rank of each item based on the total revenue in 2019 ?

```
SELECT  order_date,item_name,
total_revenue_per_item,

RANK() OVER(ORDER BY
total_revenue_per_item DESC)

FROM restaurant_1_orders

WHERE order_date >= '2019-01-01'
```

RANK WITH Partition BY

What is the rank of each item based on the total revenue in 2019 ?

```
SELECT  order_date,item_name,
total_revenue_per_item,

RANK() OVER(PARTITION BY order_date
ORDER BY total_revenue_per_item DESC)

FROM restaurant_1_orders

WHERE order_date >= '2019-01-01'
```

Commonly used Window functions

RANK()

DENSE_RANK()

ROW_NUMBER()

LAG()

LEAD()

NTILE() (not so commonly used)