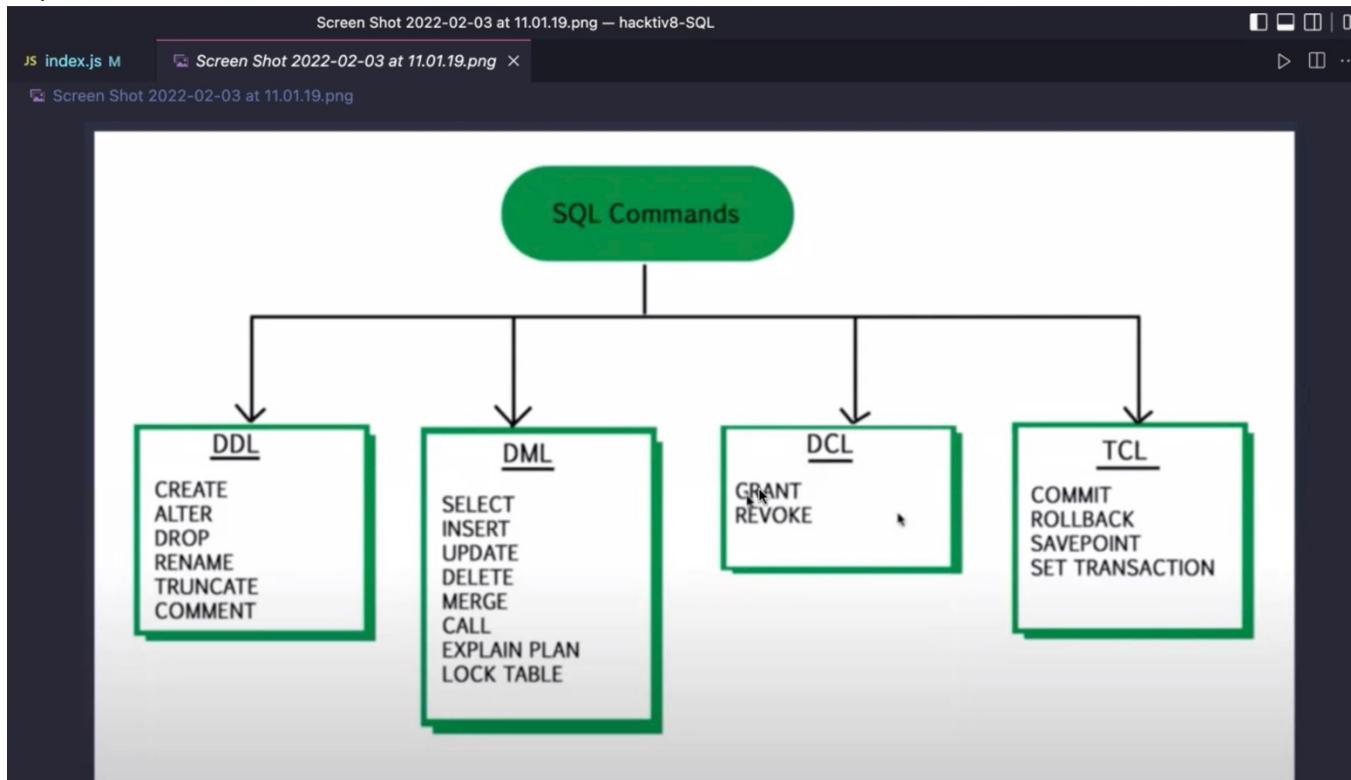


SQL



DBeaver 21.1.3 - <postgres 2> Script-2

```
--manfaatkan aggregate pada sql, lakukan:  
--penjumlahan  
--rata - rata  
--cari nilai max  
--cari nilai min pada total_price orders  
select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o ;  
  
--Lakukan Raw Query untuk:  
--Tampilkan order yang memiliki quantity pemesanan diatas 2  
--Tampilkan order yang memiliki quantity pemesanan diantara 2 sampai 3  
select * from orders o where o.quantity > 2;  
select * from orders where orders.quantity between 1 and 3;  
select * from orders o where o.quantity >= 1 and o.quantity <=3;
```

orders 1

| id | user_id | product_id | quantity | total_price | is_paid | order_date | paid_date | created_at | updated_at |
|----|---------|------------|----------|-------------|---------|---------------------|---------------------|---------------------|---------------------|
| 1 | 2 | 5 | 2 | 474 | [v] | 2020-09-07 00:09:07 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 | 2022-08-25 08:42:21 |
| 2 | 3 | 1 | 3 | 621 | [v] | 2020-05-02 15:42:35 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 | 2022-08-25 08:42:21 |
| 3 | 4 | 3 | 1 | 1,332 | [v] | 2020-09-08 20:39:33 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 | 2022-08-25 08:42:21 |
| 4 | 5 | 4 | 5 | 1 | 833 | 2020-07-05 16:55:17 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 | 2022-08-25 08:42:21 |
| 5 | 6 | 4 | 5 | 3 | 2,499 | [v] | 2020-07-24 19:00:42 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |
| 6 | 7 | 1 | 7 | 1 | 771 | [v] | 2021-08-08 04:41:08 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |
| 7 | 8 | 1 | 7 | 1 | 771 | [v] | 2021-02-17 22:21:05 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |
| 8 | 9 | 3 | 1 | 2 | 888 | [v] | 2020-09-16 14:40:27 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |
| 9 | 10 | 4 | 3 | 3 | 621 | [v] | 2020-06-16 12:25:48 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |
| 10 | 11 | 2 | 5 | 3 | 2,499 | [v] | 2020-05-11 05:01:43 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |
| 11 | 12 | 3 | 7 | 2 | 1,542 | [v] | 2020-01-27 06:39:28 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |
| 12 | 1 | 1 | 5 | 1 | 833 | [] | 2020-05-17 05:58:24 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 |

DBeaver 21.1.3 - <postgres 2> Script-2

public orders *<postgres 2> Script-2

```

--manfaatkan aggregate pada sql, lakukan:
--penjumlahan
--rata - rata
--cari nilai max
--cari nilai min pada total_price orders

select SUM(total_price) as sum_price, avg(total_price), max(total_price) from orders o ;

--Lakukan Raw Query untuk:
--Tampilkan order yang memiliki quantity pemesanan diatas 2
--Tampilkan order yang memiliki quantity pemesanan diantara 2 sampai 3

select * from orders o where o.quantity > 2;
select * from orders where orders.quantity between 1 and 3;
select * from orders o where o.quantity >= 1 and o.quantity <=3;

--ambilah data total_quantity dari penjumlahan 'quantity' order, data jumlah total_price dari penjumlahan 'total_price' order, serta product_id , kelompokkan berdasarkan product id nya
--data yang didapatkan sudah benar, namun kita butuh menampilkan product namanya juga !
--data yang didapatkan sudah benar, namun kita butuh menampilkan categorynya juga !

select SUM(quantity) as sum_quantity, sum(total_price) as sum_total_price, product_id from orders group by product_id ;

```

orders 1

| | sum_quantity | sum_total_price | product_id |
|---|--------------|-----------------|------------|
| 1 | 6 | 1,242 | 3 |
| 2 | 8 | 6,664 | 5 |
| 3 | 2 | 474 | 2 |
| 4 | 4 | 3,084 | 7 |
| 5 | 5 | 2,220 | 1 |

JOINS

<https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-joins/>

```
public orders *<postgres 2> Script-2 products
--Lakukan Raw Query untuk:
--Tampilkan order yang memiliki quantity pemesanan diatas 2
--Tampilkan order yang memiliki quantity pemesanan diantara 2 sampai 3

select * from orders o where o.quantity > 2;
select * from orders where orders.quantity between 1 and 3;
select * from orders o where o.quantity >= 1 and o.quantity <=3;

--ambilah data total_quantity dari penjumlahan 'quantity' order, data jumlah total_price dari penjumlahan 'total_price' order, serta product_id , kelompokkan berdasarkan product id nya
--data yang didapatkan sudah benar, namun kita butuh menampilkan product namanya juga !
--data yang didapatkan sudah benar, namun kita butuh menampilkan categorynya juga !

select
    SUM(quantity) as sum_quantity,
    sum(total_price) as sum_total_price,
    product_id,
    p.name
from
    orders
inner join
    products p
    on p.id = orders.product_id
group by
    product_id, p.name ;
orders(+) 1

select SUM(quantity) as sum_quantity, sum(total_price) as sum_total_price Enter a SQL expression to filter results (use Ctrl+Space)
Grid 123 sum_quantity 123 sum_total_price 123 product_id abc name
1 5 2,220 1 Tasty Plastic Shoes
2 8 6,664 5 Awesome Granite Bike
3 6 1,242 3 Licensed Plastic Keyboard
4 4 3,084 7 Rustic Plastic Fish
5 2 474 2 Practical Cotton Salad
```

```
--Tampilkan order yang memiliki quantity pemesanan diatas 2
--Tampilkan order yang memiliki quantity pemesanan diantara 2 sampai 3

select * from orders o where o.quantity > 2;
select * from orders where orders.quantity between 1 and 3;
select * from orders o where o.quantity >= 1 and o.quantity <=3;

--ambilah data total_quantity dari penjumlahan 'quantity' order, data jumlah total_price dari penjumlahan 'total_price' order, serta product_id , kelompokkan berdasarkan product id nya
--data yang didapatkan sudah benar, namun kita butuh menampilkan product namanya juga !
--data yang didapatkan sudah benar, namun kita butuh menampilkan categorynya juga !

select
    SUM(quantity) as sum_quantity,
    sum(total_price) as sum_total_price,
    product_id,
    p.name,
    c.name
from
    orders
inner join
    products p
    on p.id = orders.product_id
inner join
    categories c
    on c.id = p.category_id
group by
    product_id, p.name, c.name ;
orders(+) 1

select SUM(quantity) as sum_quantity, sum(total_price) as sum_total_price Enter a SQL expression to filter results (use Ctrl+Space)
Grid 123 sum_quantity 123 sum_total_price 123 product_id abc name abc name
1 6 1,242 3 Licensed Plastic Keyboard Gorgeous
2 5 2,220 1 Tasty Plastic Shoes Incredible
3 8 6,664 5 Awesome Granite Bike Sleek
4 2 474 2 Practical Cotton Salad Sleek
5 4 3,084 7 Rustic Plastic Fish Sleek
```

-- cari order di tahun 2021

```
select * from orders o where date_part('year', o.order_date) = 2021;
```

orders 1

| id | user_id | product_id | quantity | total_price | is_paid | order_date | paid_date | created_at | updated_at |
|----|---------|------------|----------|-------------|---------|---------------------|---------------------|---------------------|---------------------|
| 1 | 7 | 1 | 7 | 771 | [v] | 2021-08-08 04:41:08 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 | 2022-08-25 08:42:21 |
| 2 | 8 | 1 | 7 | 771 | [v] | 2021-02-17 22:21:05 | 2021-09-28 10:02:23 | 2022-08-25 08:42:21 | 2022-08-25 08:42:21 |

SQL RAW QUERIES

<https://sequelize.org/docs/v6/core-concepts/raw-queries/>

```
controllers > index.js > [?]<unknonw> > ⚡ findOrders
const { Order, Product, Category } = require('../models/index');
const {
  Sequelize,
  Op
} = require('../models');

module.exports = {
  findOrders(req, res) {
    // sum, min, max
    Order.findAll({
      attributes: [
        [sequelize.fn('SUM', sequelize.col('total_price')), 'sum_total_price'],
        [sequelize.fn('MIN', sequelize.col('total_price')), 'min_total_price'],
        [sequelize.fn('MAX', sequelize.col('total_price')), 'max_total_price'],
      ],
    })
      .then((data) => {
        res.status(200).json(data);
      })
      .catch((err) => {
        res.status(500).json(err);
      });
  },
  async payOrder(req, res) {},
};


```

The screenshot shows a Node.js application running in a development environment. The top bar has tabs for `index.js`, `controllers`, `models`, `Preview README.md`, and `localhost:3000`. The main area displays the `index.js` file content:

```
JS index.js controllers M X JS index.js models Preview README.md localhost:3000
controllers > JS index.js > [unknown] > findOrders > then() callback > result
1 const { Order, Product, Category } = require('../models/index');
2 const {
3   sequelize,
4   Sequelize: { Op },
5 } = require('../models');
6
7 module.exports = {
8   findOrders(req, res) {
9     // sum, min, max
10    // select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o ;
11
12   sequelize
13     .query(
14       'select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o'
15     )
16     .then((data) => {
17       res.status(200).json({ Body.json(): Promise<any>
18         result: data[0][0],
19       });
20       console.log(data[0]);
21     })
22     .catch((err) => {
23       console.log(err);
24     });
25   // Order.findOne({
26   //   attributes: [
27   //     [sequelize.fn('SUM', sequelize.col('total_price')), 'sum_total_price'],
28   //     [sequelize.fn('MIN', sequelize.col('total_price')), 'min_total_price'],
29   //     [sequelize.fn('MAX', sequelize.col('total_price')), 'max_total_price'],
30   //   ],
31   // })
32 }
33 }
```

The terminal below shows the application output:

```
[nodemon] starting `node ./bin/www`
Executing (default): select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o
{
  sum_price: '13684',
  avg: '1140.3333333333333333',
  max: 2499,
  min: 474
}
```

A screenshot of a Node.js development environment within a code editor. The top navigation bar shows tabs for 'index.js controllers M' (active), 'index.js models', 'Preview README.md', and 'localhost:3000'. Below the tabs, a breadcrumb navigation path indicates the file structure: 'controllers > index.js > [] <unknown> > findOrders > then() callback'. The main code editor area displays a JavaScript file with Sequelize query logic. The terminal below shows the execution of the code, resulting in an array of objects representing product categories with their respective sum quantities and total prices. A successful HTTP request is shown at the bottom.

```
3  sequelize,
4  Sequelize: { Op },
5 } = require('../models');
6
7 module.exports = {
8   findOrders(req, res) {
9     // sum, min, max
10    // select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o ;
11
12   sequelize
13     .query(
14       `select
15        SUM(quantity) as sum_quantity,
16        sum(total_price) as sum_total_price,
17        product_id,
18        p.name,
19        c.name as category
20      from
21      orders
22      inner join
23      products p
24      on p.id = orders.product_id
25      inner join
26      categories c
27      on c.id = p.category_id
28      group by
29        product_id, p.name, c.name ;`
30     )
31     .then((data) => [
32       console.log(data[0]);
33       res.status(200).json({
34         sum_quantity: '4',
35         sum_total_price: '3084',
36         product_id: 7,
37         name: 'Rustic Plastic Fish',
38         category: 'Sleek'
39       }
40     ]
41     GET / 200 265.736 ms - 574
42   
```

PostgreSQL inner join

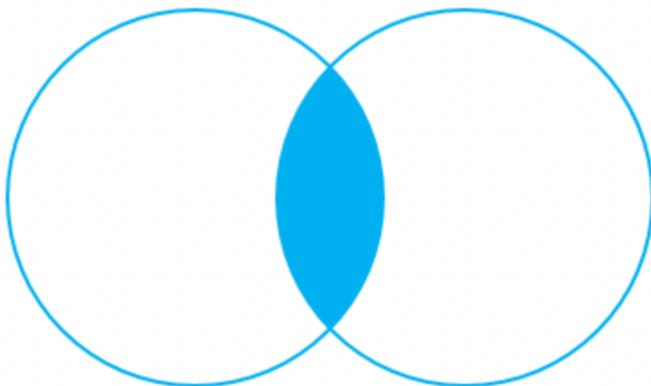
The following statement joins the first table (`basket_a`) with the second table (`basket_b`) by matching the values in the `fruit_a` and `fruit_b` columns:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
INNER JOIN basket_b
    ON fruit_a = fruit_b;
```

| | a | fruit_a | b | fruit_b |
|---|---------|-------------------------|---------|-------------------------|
| | integer | character varying (100) | integer | character varying (100) |
| 1 | 1 | Apple | 2 | Apple |
| 2 | 2 | Orange | 1 | Orange |

The inner join examines each row in the first table (`basket_a`). It compares the value in the `fruit_a` column with the value in the `fruit_b` column of each row in the second table (`basket_b`). If these values are equal, the inner join creates a new row that contains columns from both tables and adds this new row to the result set.

The following Venn diagram illustrates the inner join:



PostgreSQL left join

The following statement uses the left join clause to join the `basket_a` table with the `basket_b` table. In the left join context, the first table is called the left table and the second table is called the right table.

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
LEFT JOIN basket_b
    ON fruit_a = fruit_b;
```

| | a integer | fruit_a character varying (100) | b integer | fruit_b character varying (100) |
|---|--------------|------------------------------------|--------------|------------------------------------|
| 1 | 1 | Apple | 2 | Apple |
| 2 | 2 | Orange | 1 | Orange |
| 3 | 3 | Banana | [null] | [null] |
| 4 | 4 | Cucumber | [null] | [null] |

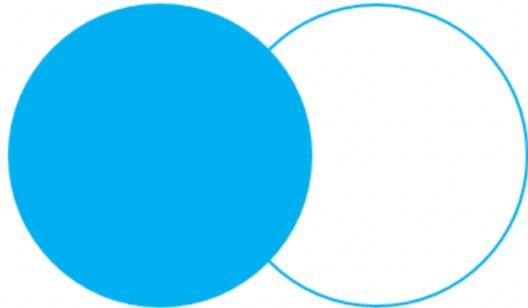
The left join starts selecting data from the left table. It compares values in the `fruit_a` column with the values in the `fruit_b` column in the `basket_b` table.

If these values are equal, the left join creates a new row that contains columns of both tables and adds this new row to the result set. (see the row #1 and #2 in the result set).

In case the values do not equal, the left join also creates a new row that contains columns from both tables and adds it to the result set. However, it fills the columns of the right table

(`basket_b`) with null. (see the row #3 and #4 in the result set).

The following Venn diagram illustrates the left join:



LEFT OUTER JOIN

To select rows from the left table that do not have matching rows in the right table, you use the left join with a `WHERE` clause. For example:

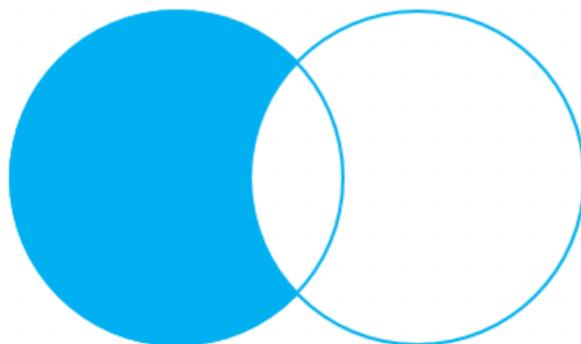
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
LEFT JOIN basket_b
    ON fruit_a = fruit_b
WHERE b IS NULL;
```

The output is:

| | a integer | fruit_a character varying (100) | b integer | fruit_b character varying (100) |
|---|--------------|------------------------------------|--------------|------------------------------------|
| 1 | 3 | Banana | [null] | [null] |
| 2 | 4 | Cucumber | [null] | [null] |

Note that the `LEFT JOIN` is the same as the `LEFT OUTER JOIN` so you can use them interchangeably.

The following Venn diagram illustrates the left join that returns rows from the left table that do not have matching rows from the right table:



**LEFT OUTER JOIN – only
rows from the left table**

