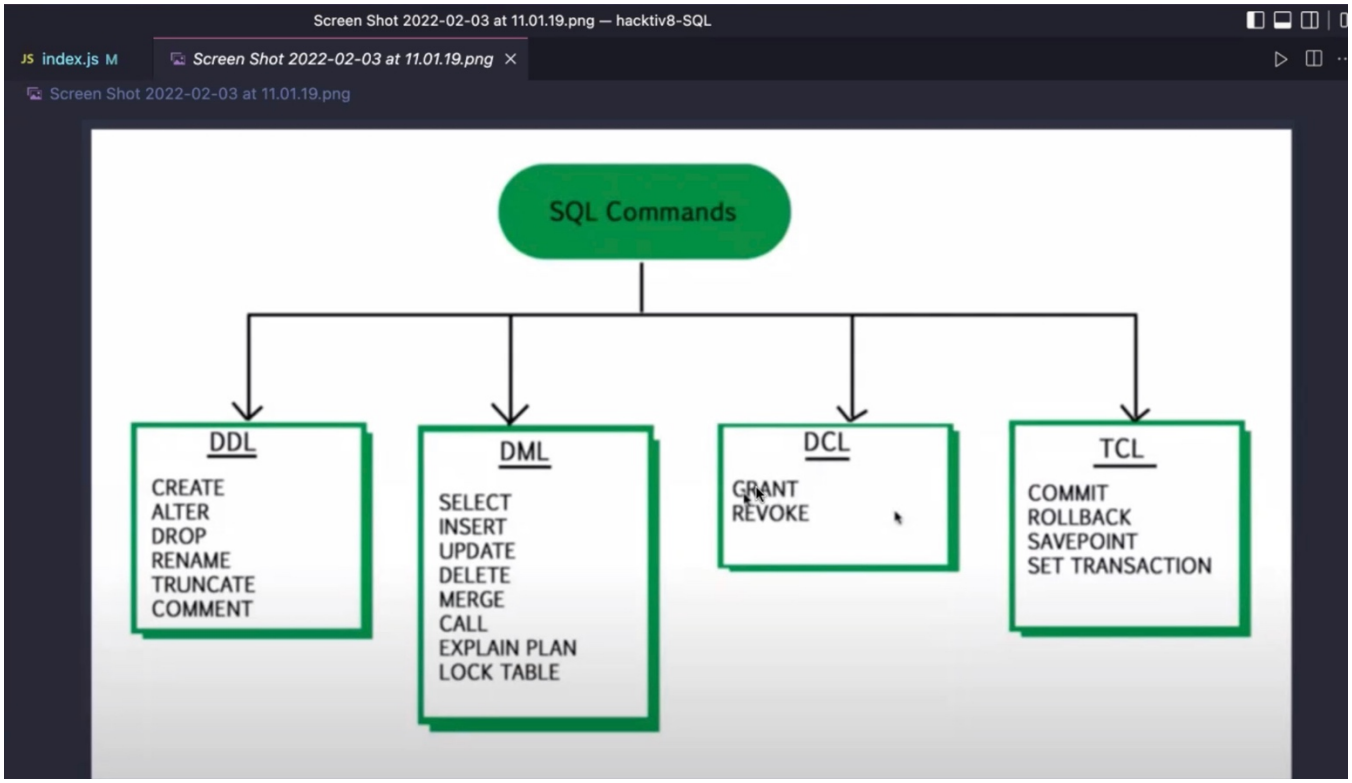
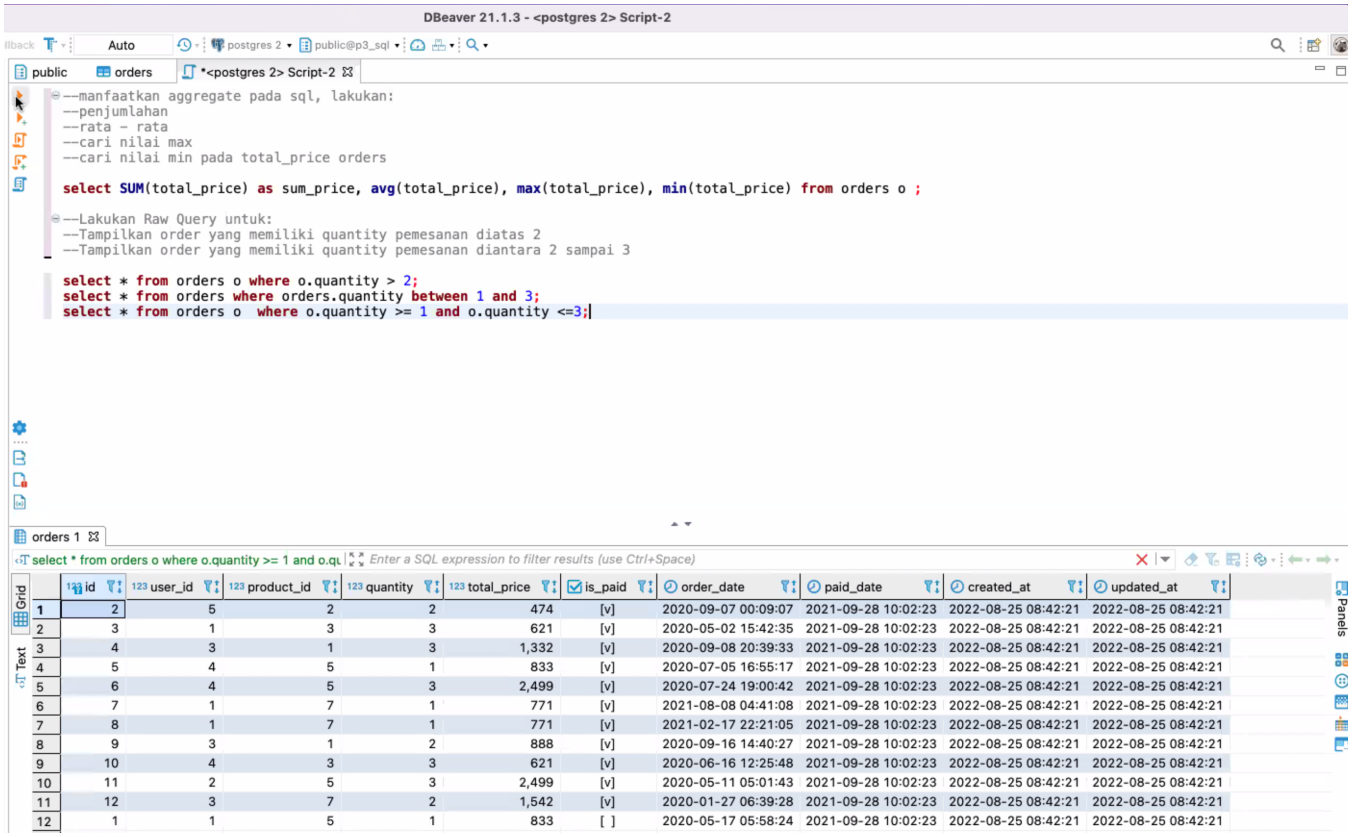
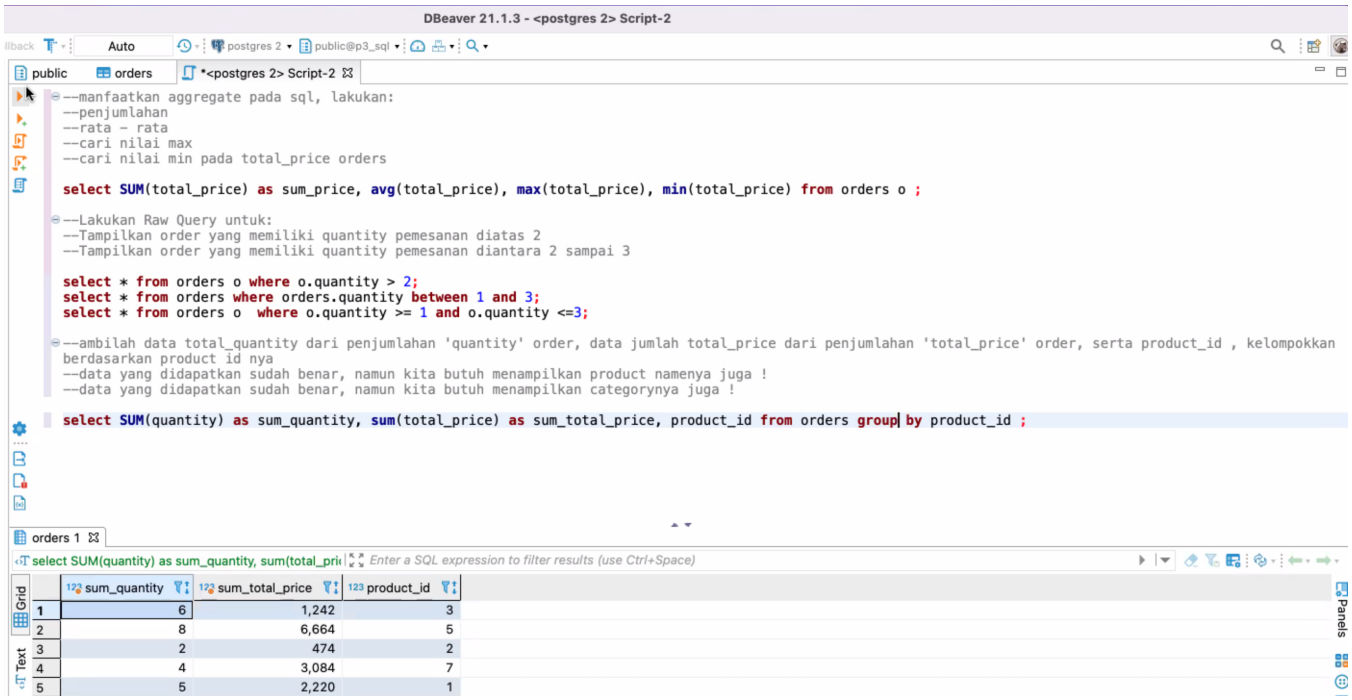
SQL



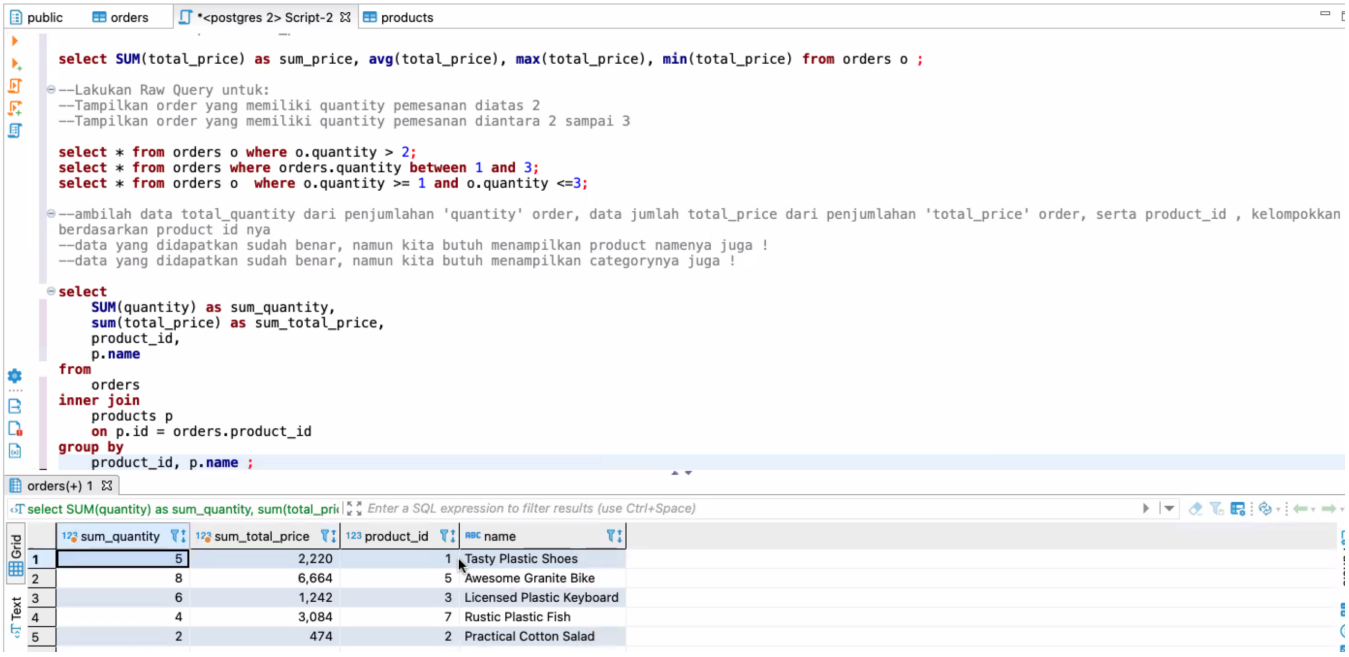


	id	user_id	product_id	quantity	total_price	is_paid	order_date	paid_date	created_at	updated_at
1	2	5	2	2	474	[v]	2020-09-07 00:09:07	2021-09-28 10:02:23	2022-08-25 08:42:21	2022-08-25 08:42:21
2	3	1	3	3	621	[v]	2020-05-02 15:42:35	2021-09-28 10:02:23	2022-08-25 08:42:21	2022-08-25 08:42:21
3	4	3	1	3	1,332	[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	[v]	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	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	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	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	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	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	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	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	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), 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;
--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 namenya 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
Enter a SQL expression to filter results (use Ctrl+Space)
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
Grid
Text
Panels

JOINS

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



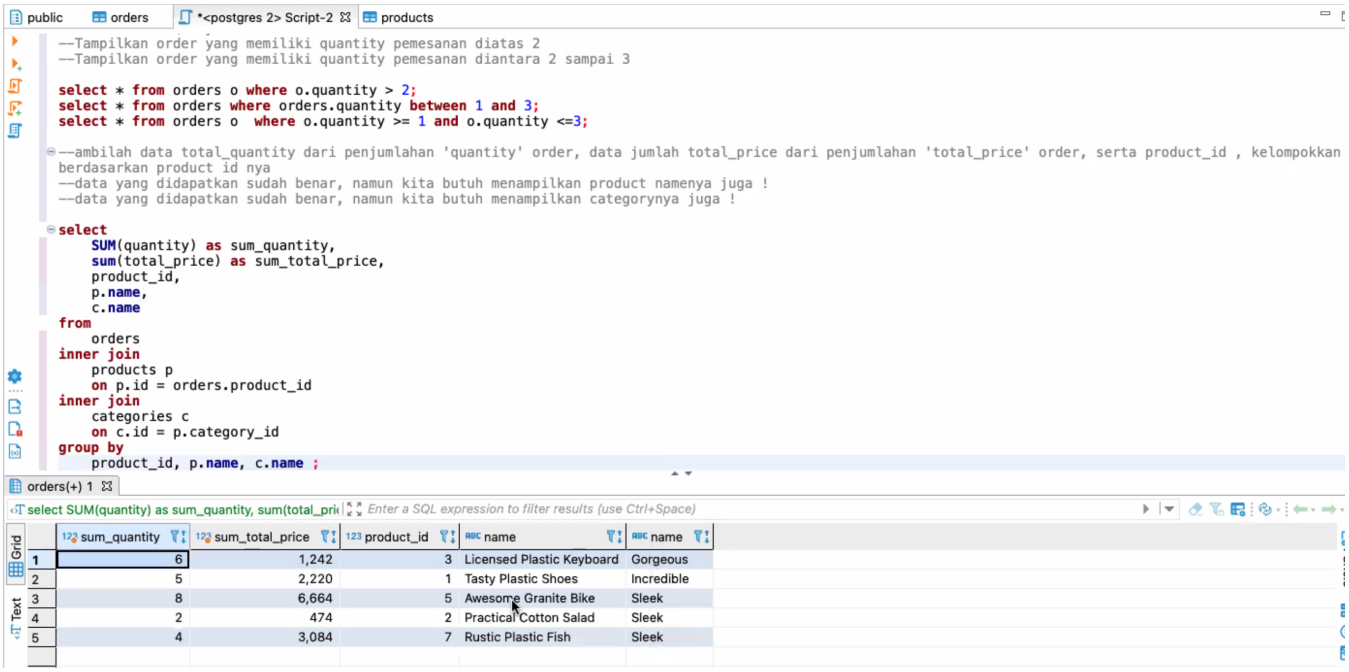
```
select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o ;

--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;

--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 namenya 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_pri
```

Enter a SQL expression to filter results (use Ctrl+Space)

	sum_quantity	sum_total_price	product_id	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



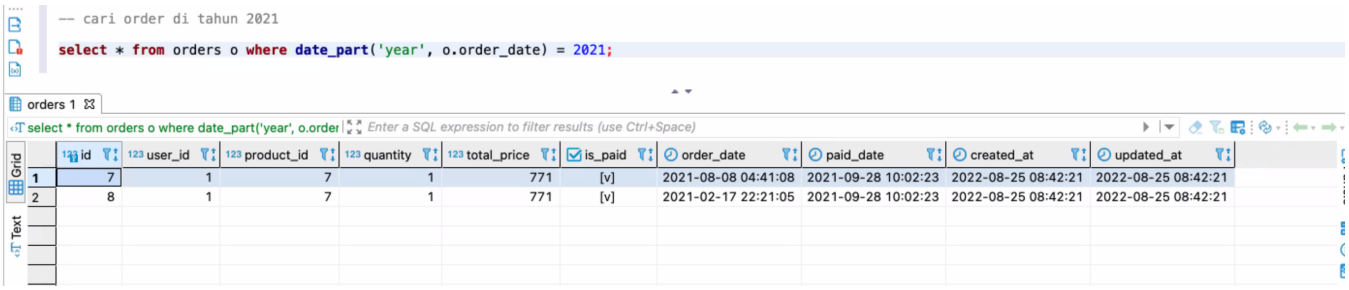
```
--Tampilkan order yang memiliki quantity pemesanan diatas 2
--Tampilkan order yang memiliki quantity pemesanan diantara 2 sampai 3

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 namenya 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_pri
```

Enter a SQL expression to filter results (use Ctrl+Space)

	sum_quantity	sum_total_price	product_id	name	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

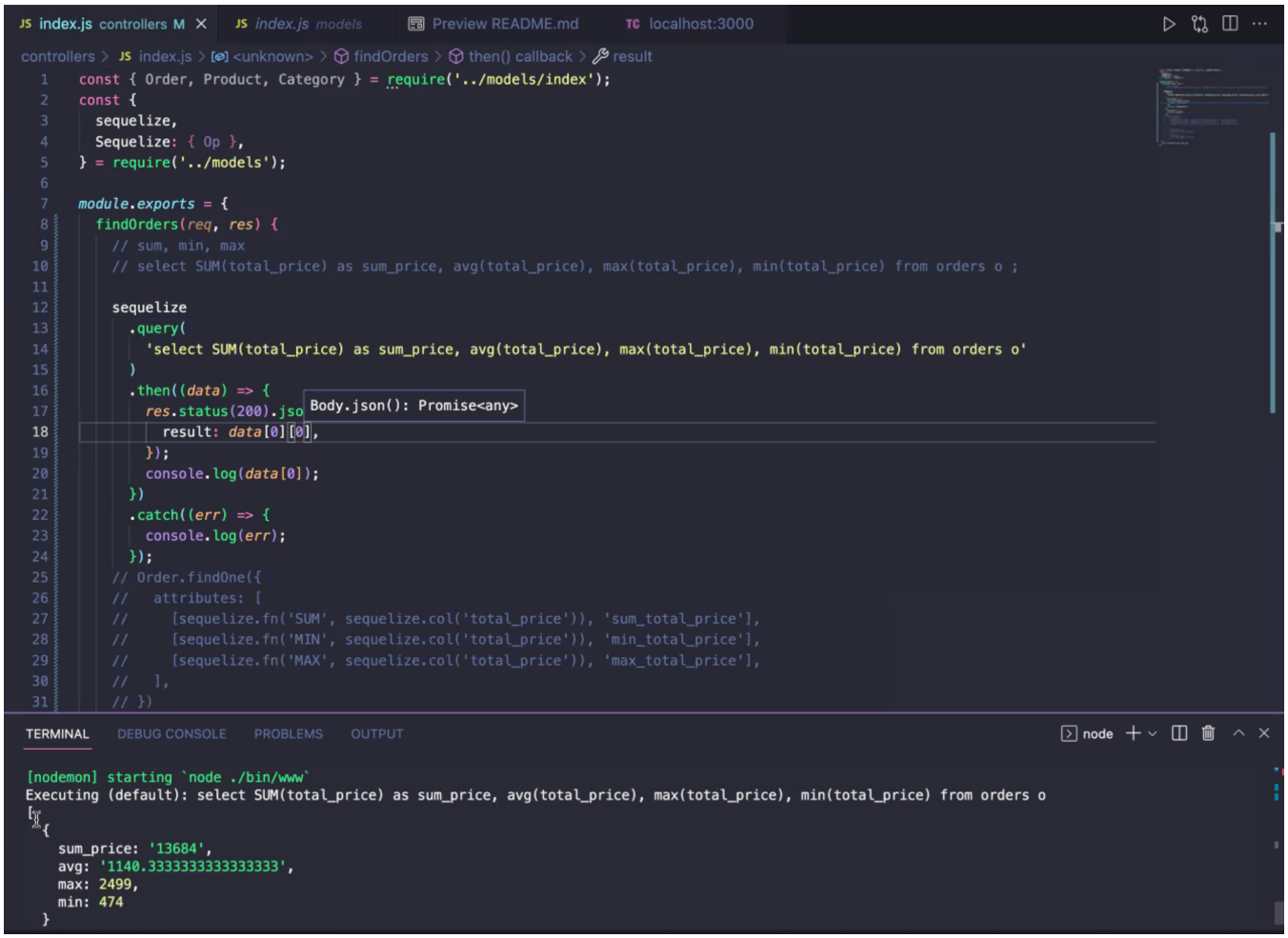


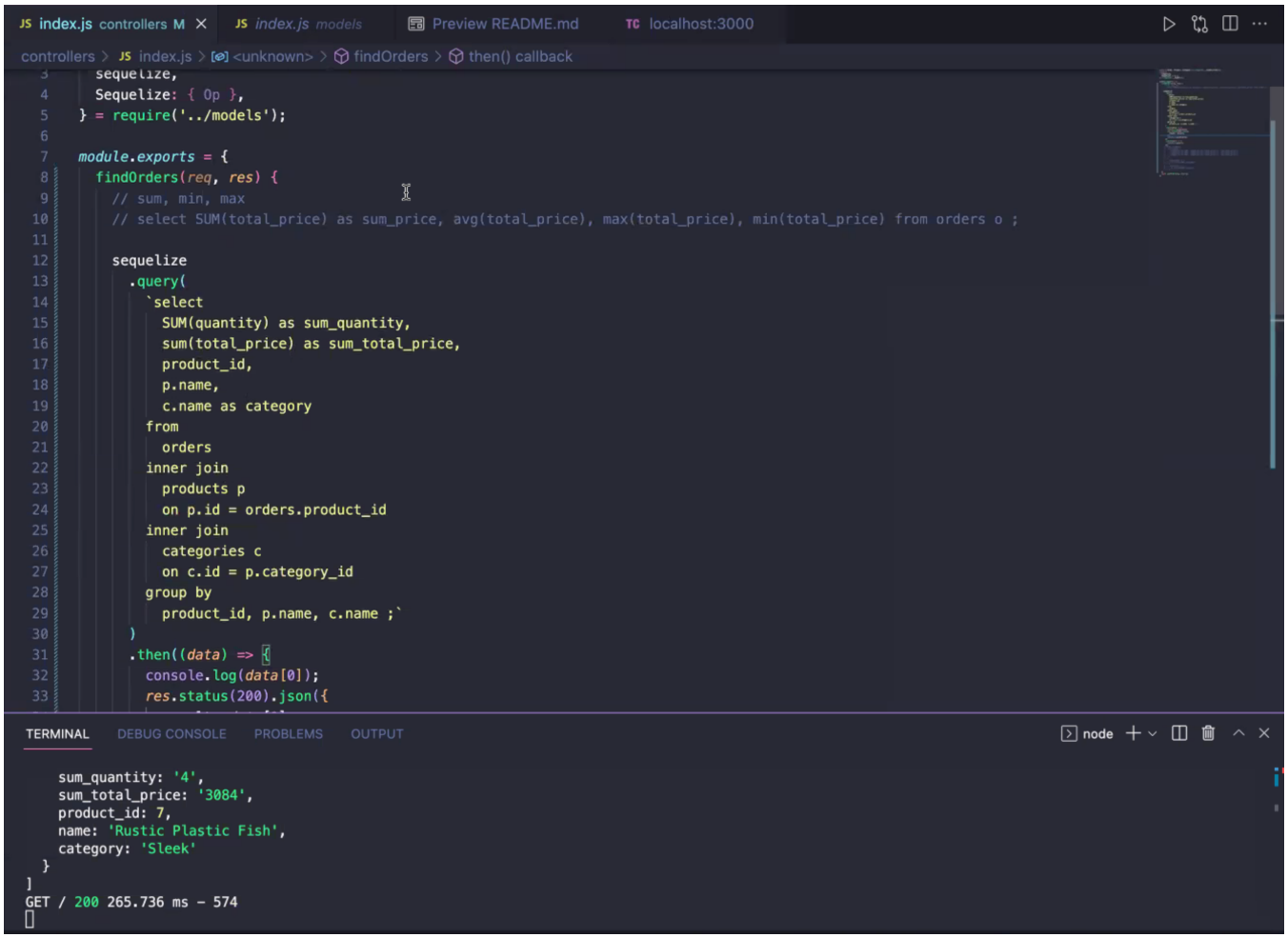
SQL RAW QUERIES
https://sequelize.org/docs/v6/core-concepts/raw-queries/

index.js M | Preview README.md | localhost:3000 | orders.js

controllers > index.js > <unknown> > findOrders

```
const { Order, Product, Category } = require('../models/index');
const {
  sequelize,
  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) {},
};
```


index.js controllers M
index.js models
Preview README.md
localhost:3000
controllers > index.js > <unknown> > findOrders > then() callback > result
const { Order, Product, Category } = require('../models/index');
const {
sequelize,
Sequelize: { Op },
} = require('../models');
module.exports = {
findOrders(req, res) {
// sum, min, max
// select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o ;
sequelize
.query(
'select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o'
)
.then((data) => {
res.status(200).jso
Body.json(): Promise<any>
result: data[0][0],
});
console.log(data[0]);
})
.catch((err) => {
console.log(err);
});
// Order.findOne({
// 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'],
//],
// })
TERMINAL
DEBUG CONSOLE
PROBLEMS
OUTPUT
node
[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
}


index.js controllers M
index.js models
Preview README.md
localhost:3000
controllers > index.js > <unknown> > findOrders > then() callback
sequelize,
Sequelize: { Op },
} = require('../models');
module.exports = {
findOrders(req, res) {
// sum, min, max
// select SUM(total_price) as sum_price, avg(total_price), max(total_price), min(total_price) from orders o ;
sequelize
.query(
`select
SUM(quantity) as sum_quantity,
sum(total_price) as sum_total_price,
product_id,
p.name,
c.name as category
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 ;`
)
.then((data) => {
console.log(data[0]);
res.status(200).json({
TERMINAL
DEBUG CONSOLE
PROBLEMS
OUTPUT
node
sum_quantity: '4',
sum_total_price: '3084',
product_id: 7,
name: 'Rustic Plastic Fish',
category: 'Sleek'
}
]
GET / 200 265.736 ms - 574

PostgreSQL inner join

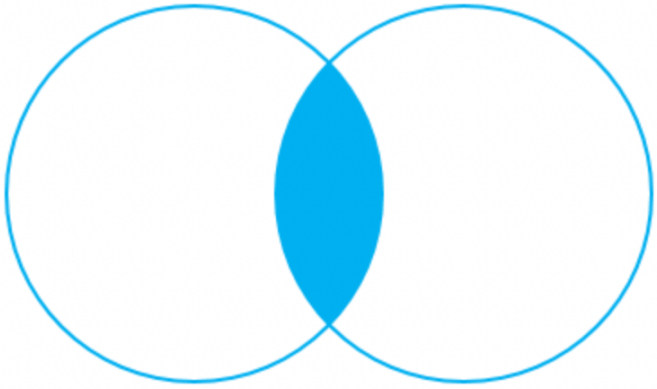
The following statement joins the first table (`basket_a`) with the second table (`basket_b`) by matching the values in the `fruit_a` and `fruit_b` columns:

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

	a integer	fruit_a character varying (100)	b integer	fruit_b 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 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;
```

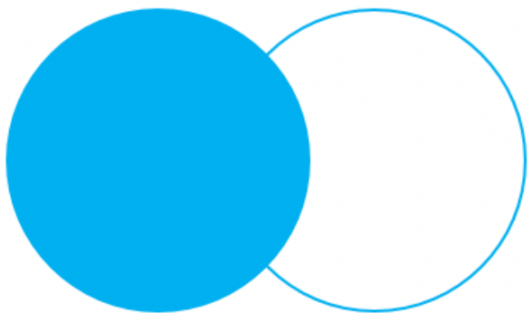
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]

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:

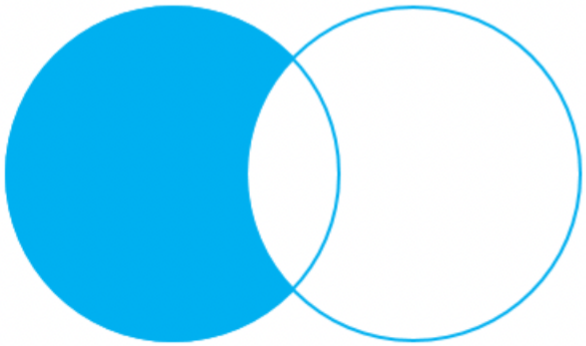
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
LEFT JOIN basket_b
    ON fruit_a = fruit_b
WHERE b IS NULL;
```

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