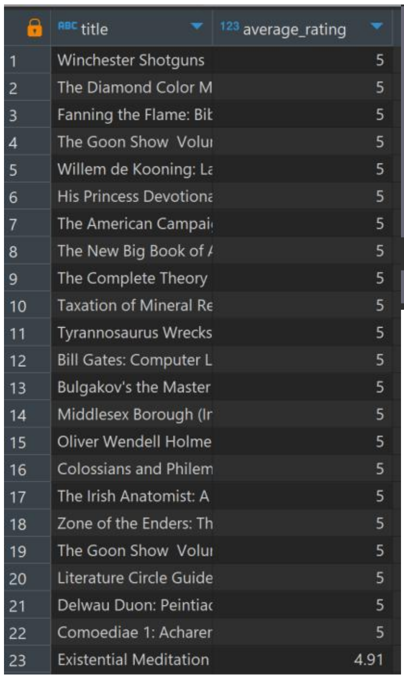
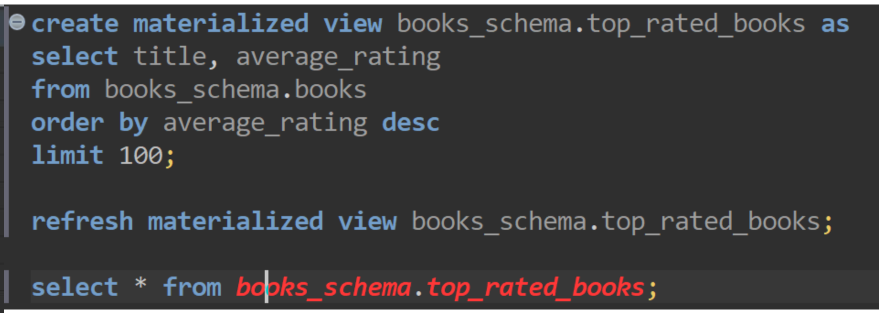


	title	average_rating
1	Winchester Shotguns	5
2	The Diamond Color M	5
3	Fanning the Flame: Bit	5
4	The Goon Show Volu	5
5	Willem de Kooning: La	5
6	His Princess Devotiona	5
7	The American Campai	5
8	The New Big Book of	5
9	The Complete Theory	5
10	Taxation of Mineral Re	5
11	Tyrannosaurus Wrecks	5
12	Bill Gates: Computer L	5
13	Bulgakov's the Master	5
14	Middlesex Borough (Ir	5
15	Oliver Wendell Holme	5
16	Colossians and Philem	5
17	The Irish Anatomist: A	5
18	Zone of the Enders: Th	5
19	The Goon Show Volu	5
20	Literature Circle Guide	5
21	Delwau Duon: Peintiac	5
22	Comoediae 1: Acharer	5
23	Existential Meditation	4.91

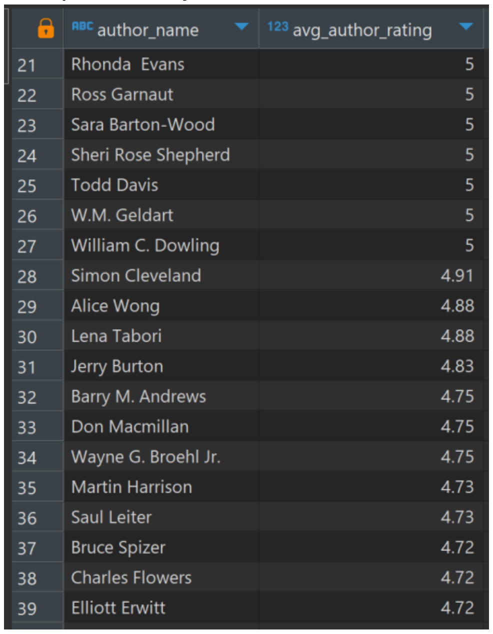


```sql
create materialized view books_schema.top_rated_books as
select title, average_rating
from books_schema.books
order by average_rating desc
limit 100;

refresh materialized view books_schema.top_rated_books;

select * from books_schema.top_rated_books;
```

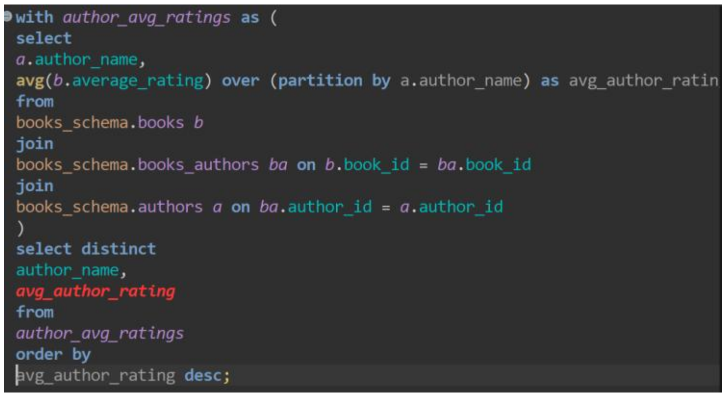
A materialized view is a way in which data can be stored physically from the query and helps improve query performance when certain data is being accessed a lot. However, they need to be updated from time to time to make them up to date. I chose to create this table since there are times where someone might be interested in seeing what the top 100 books are or even top 10 or so, we can modify the sql to do so but this would reduce the amount of time needed to write and execute the code.

<u>Window Functions</u>

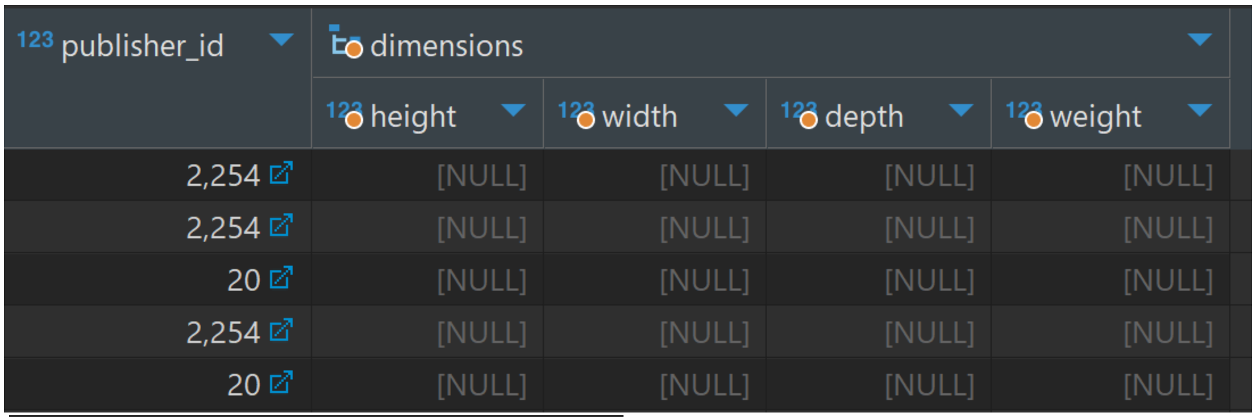
The window function here is used to help calculate the authors average rating for all their books which shows the use of being able to work across related rows. They are more helpful than having to use things like subqueries or joins.



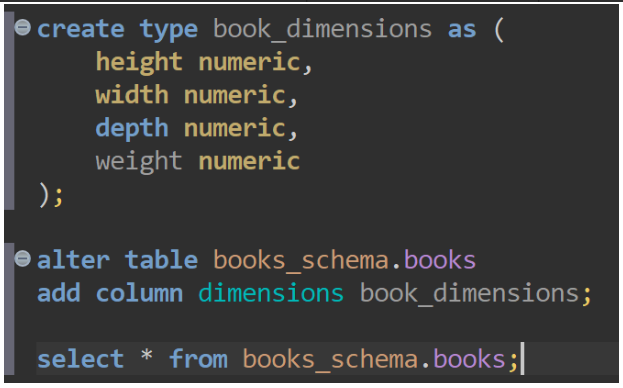
	author_name	avg_author_rating
21	Rhonda Evans	5
22	Ross Garnaut	5
23	Sara Barton-Wood	5
24	Sheri Rose Shepherd	5
25	Todd Davis	5
26	W.M. Geldart	5
27	William C. Dowling	5
28	Simon Cleveland	4.91
29	Alice Wong	4.88
30	Lena Tabori	4.88
31	Jerry Burton	4.83
32	Barry M. Andrews	4.75
33	Don Macmillan	4.75
34	Wayne G. Broehl Jr.	4.75
35	Martin Harrison	4.73
36	Saul Leiter	4.73
37	Bruce Spizer	4.72
38	Charles Flowers	4.72
39	Elliott Erwitt	4.72



```sql
with author_avg_ratings as (
select
a.author_name,
avg(b.average_rating) over (partition by a.author_name) as avg_author_ratin
from
books_schema.books b
join
books_schema.books_authors ba on b.book_id = ba.book_id
join
books_schema.authors a on ba.author_id = a.author_id
)
select distinct
author_name,
avg_author_rating
from
author_avg_ratings
order by
avg_author_rating desc;
```



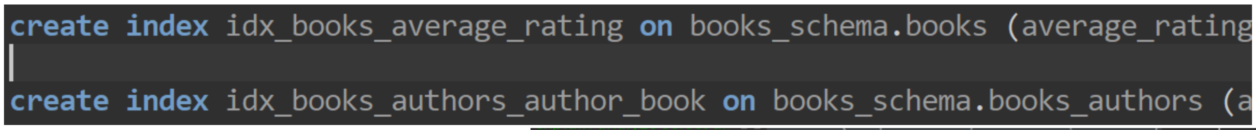
publisher_id	dimensions			
	height	width	depth	weight
2,254	[NULL]	[NULL]	[NULL]	[NULL]
2,254	[NULL]	[NULL]	[NULL]	[NULL]
20	[NULL]	[NULL]	[NULL]	[NULL]
2,254	[NULL]	[NULL]	[NULL]	[NULL]
20	[NULL]	[NULL]	[NULL]	[NULL]



```sql
create type book_dimensions as (
    height numeric,
    width numeric,
    depth numeric,
    weight numeric
);

alter table books_schema.books
add column dimensions book_dimensions;

select * from books_schema.books;
```

Custom Data Types

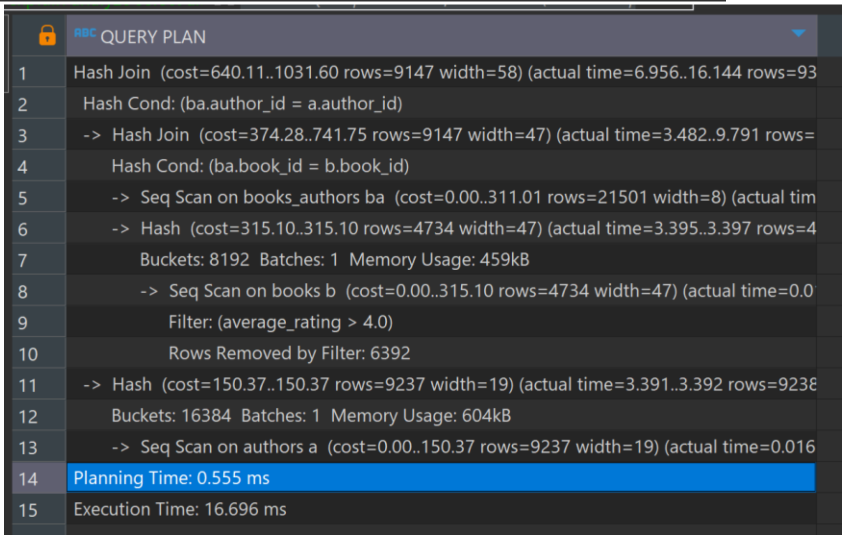
Custom data types are a way in which we can organize complex data structures into the schema and make things a bit more simplified. However it can sometimes make it harder to retrieve data at the same time. In this case I used dimensions since sometimes people might want to know how big or heavy the book might to know if they can carry it with them at all times or if it is a book that they can only read at home.



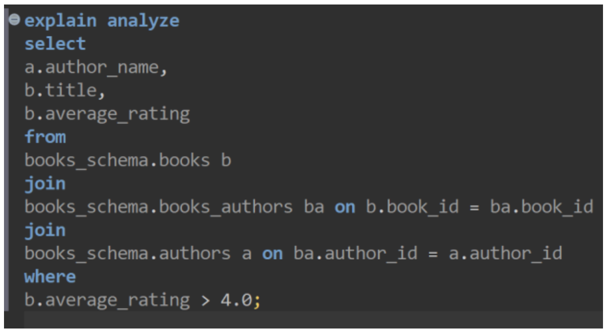
```sql
create index idx_books_average_rating on books_schema.books (average_rating

create index idx_books_authors_author_book on books_schema.books_authors (a
```

Performance Tuning

Indexes help in being able to optimize the data to be able to retrieved faster to enhance performance on frequently queried columns. Following this was



```
QUERY PLAN
Hash Join  (cost=640.11..1031.60 rows=9147 width=58) (actual time=6.956..16.144 rows=93
  Hash Cond: (ba.author_id = a.author_id)
  ->  Hash Join  (cost=374.28..741.75 rows=9147 width=47) (actual time=3.482..9.791 rows=
        Hash Cond: (ba.book_id = b.book_id)
        ->  Seq Scan on books_authors ba  (cost=0.00..311.01 rows=21501 width=8) (actual tim
        ->  Hash  (cost=315.10..315.10 rows=4734 width=47) (actual time=3.395..3.397 rows=4
              Buckets: 8192  Batches: 1  Memory Usage: 459kB
              ->  Seq Scan on books b  (cost=0.00..315.10 rows=4734 width=47) (actual time=0.0
                    Filter: (average_rating > 4.0)
                    Rows Removed by Filter: 6392
  ->  Hash  (cost=150.37..150.37 rows=9237 width=19) (actual time=3.391..3.392 rows=923
        Buckets: 16384  Batches: 1  Memory Usage: 604kB
        ->  Seq Scan on authors a  (cost=0.00..150.37 rows=9237 width=19) (actual time=0.016
Planning Time: 0.555 ms
Execution Time: 16.696 ms
```



```sql
explain analyze
select
a.author_name,
b.title,
b.average_rating
from
books_schema.books b
join
books_schema.books_authors ba on b.book_id = ba.book_id
join
books_schema.authors a on ba.author_id = a.author_id
where
b.average_rating > 4.0;
```