Modern Database Concepts
Prof. Dr. Florian Heinz
florian.heinz@oth-regensburg.de
Exercise Sheet 6

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Exercise 1: Full Text Search with PostgreSQL

Open DBeaver and connect to the OTH PostgreSQL database instance (or any other PostgreSQL-Server with version 13+)

Create a database schema **ex6** to work with. You can do that either in the context menu of the database node (your OTH-Username) at the left side or by opening an SQL console and using the `CREATE SCHEMA ...` SQL command.

1. Load and import the file `pap.sql`, which you can download from GRIPS. This file contains a table of rows containing a chapter number and the corresponding text of the book "Pride and Prejudice" from Jane Austin.
2. Check, how to perform a full-text search on the text columns to find out, which chapters contain the word 'prejudice'. Do not use the `LIKE` operator, but use the methods described in the PostgreSQL 14 manual Section 12.2.1 (the ts* function family). Check, how long the search takes with `EXPLAIN ANALYZE`
3. Now create a gin index to speed up the full text search as described in Section 12.2.2. Repeat the query, make sure the index is indeed used and compare the execution times.
4. Examine the total size of this GIN index (consult the PostgreSQL documentation 9.27) and compare it to the size of the data.
5. Using this full text search, try to find out which chapters contain the word **and**. Explain what you observe.

Modern Database Concepts
Prof. Dr. Florian Heinz
florian.heinz@oth-regensburg.de
Exercise Sheet 6

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Exercise 2: JSON and GIN Index

Load and import the file `people_array.sql`, which you can download from GRIPS. The dataset contains the SQL array version of the people and visited cities example.

1. Convert this data to JSON starting with this query:

```
CREATE TABLE jeople AS
SELECT name,id,JSONB_BUILD_OBJECT(
   'name', name,
   'id', id,
   'cities', JSONB_AGG(city),
   'info', JSONB_BUILD_OBJECT(
      'first', cities[1],
      'last', cities[ARRAY_LENGTH(cities, 1)]
   )
) AS data
   FROM people, UNNEST(cities) c(city) GROUP BY id;
```

2. Improve the query above to add a new property **nr** inside the **info** object, which should contain the total number of visited cities. Use a suitable SQL aggregation function for this
3. Using the containment operator @>, construct a query to determine the last visited city of all people, that first went to **Zaragoza**. Check the query plan and execution time with `EXPLAIN ANALYZE`
4. Create a GIN index to improve the performance of the query. Validate that the index is used and the execution time is improved.
5. Examine the total size of this GIN index (consult the PostgreSQL documentation 9.27).
6. Drop this index again and create a functional btree index which supports the query in 2 using the equality operator now.
7. Again, check that the index is used and the execution times. Compare the size of this index to the size of the GIN index.

Modern Database Concepts
Prof. Dr. Florian Heinz
florian.heinz@oth-regensburg.de
Exercise Sheet 6

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Exercise 3: JSON Path

1. Use the ?| operator to find out the names of all people, that visited the cities Berlin and Wien.
2. Try to accelerate this query with a GIN index (verify that it is used).
3. From the table **jeople**, use a single **json_path_query** to extract the name and the cities into a result table with two columns "name" and "city". This means, if the person visited 6 cities, there will be 6 entries for her in the result table. Now, refine the JSONPath expression to only get people which visited more than 5 cities.
4. Now, aggregate the result by name into an SQL array, so that each person is only listed once together with an SQL array of the cities he visited.