Practice on the Energy database

See `./sql/create_database_energydb.sql` to create :

- the database `energydb`
- the tables: `countries` , `energy_production_daily` , `energy_sources` , and `production_entities`
- the functions: `generate_daily_production` and `generate_production_entities`
- and generating thousands of rows in each of these tables

see also `./docs/06/S06.03.energy-db-a-new-database.md` for an explanation.

Indexing Exercises

The goal of this exercise is to illustrate the impact of B-tree and Hash indexes on a table attribute.

Start by EXPLAIN ANALYZE a simple select query and note the real performance of the query in terms of cost and total execution time.

We'll add indexes (B-tree and Hash) on that column to show that adding indexes can or sometimes cannot improve the query performance.

We'll look at different types of filtering :

- operator : =, range, < or >
- resulting volume of the filtering : when the query returns a unique record or a few, more than a few or many records

and compare the EXPLAIN ANALYZE plans.

We'll also look at INSERTS queries to understand the overhead brought by the presence of indexes.

Keep in mind

Here are some key points for you to keep in mind:

1. Hash indexes are generally best for equality conditions, while B-tree indexes are versatile and can handle equality, range, and sorting operations.
2. The effectiveness of an index depends on various factors, including the selectivity of the query (i.e., how many rows it's expected to return relative to the total number of rows in the table).
3. Sometimes, the query planner might choose not to use an index if it determines that a sequential scan would be faster (e.g., when returning a large portion of the table).
4. Multi-column indexes can be particularly useful for queries that filter on multiple columns simultaneously.
5. Function-based indexes can improve performance for queries that use functions or expressions in their WHERE clauses.

The database

The results will depends on the volume of data your version of the `energydb` has generated.

If you don't see major differences in the performance time of the queries with or without indexes, then don't hesitate to generate more data using the `generate_daily_production` and `generate_production_entities` functions.

The results presented in this document are based on 400k production entities.

Exercise 1: Equality Condition with Hash Index vs. B-tree Index

The candidate query is :

```sql
SELECT * FROM production_entities
WHERE energy_source_id = 1;
```

There are only 6 types of energy sources in the database. So this query will return quite a large amount of rows.

Our **baseline** consists of this query without any index on the `energy_source_id` column.

No Index

Run the following query and use EXPLAIN ANALYZE to evaluate its performance:

```SQL
EXPLAIN ANALYZE SELECT * FROM production_entities
WHERE energy_source_id = 1;
```

results : here's a possible query plan (yours may differ)

```SQL
 Seq Scan on production_entities  (cost=0.00..20187.25 rows=66643 width=
   Filter: (energy_source_id = 1)
   Rows Removed by Filter: 333408
 Planning Time: 0.055 ms
 Execution Time: 61.401 ms
 ```

Interpretation : as expected, the planner uses a Sequential Scan to

Add B-tree

Create a B-tree index with

```sql
create index idx_energy_source_id on  production_entities (energy_sourc
```

check that the index has been created. `\d production_entities` should inclued this line

```SQL
Indexes:
    "idx_energy_source_id" btree (energy_source_id)
```

Now repeat the EXPLAIN ANALYZE query.

results : A possible query plan (yours may differ)

```SQL
 Bitmap Heap Scan on production_entities  (cost=744.91..16763.94 rows=6
   Recheck Cond: (energy_source_id = 1)
   Heap Blocks: exact=5981
   ->  Bitmap Index Scan on idx_energy_source_id  (cost=0.00..728.25 ro
         Index Cond: (energy_source_id = 1)
 Planning Time: 0.373 ms
 Execution Time: 29.159 ms
```

Interpretation : as expected, the planner uses a Bitmap Index Scan + Bitmap Heap Scan

(since there are quite a lot of rows in the result) leveraging the index. The time has gone down to `29.159 ms` and the cost to `16763.94` .

adding the index reduces the query execution time by a rough factor of 2

Add Hash Index

Let's do the same but now with a Hash Index.

But first drop the b-tree inedx with

```SQL
drop index idx_energy_source_id;
```

Create the hash index with

```SQL
create index idx_energy_source_id  on  production_entities USING HASH (
```

and repeat the EXPLAIN ANALYZE of the query.

results : A possible query plan (yours may differ)

```SQL
Bitmap Heap Scan on production_entities  (cost=2196.48..18215.52 rows=(
  Recheck Cond: (energy_source_id = 1)
  Heap Blocks: exact=5981
  ->  Bitmap Index Scan on idx_energy_source_id  (cost=0.00..2179.82 r(
        Index Cond: (energy_source_id = 1)
Planning Time: 0.281 ms
Execution Time: 32.655 ms
```

Interpretation : the Hash Index brings a similar performance boost to a B-tree index with a cost of `18215` and an execution time of around 30ms.

Conclusion

When the number of records is significant there is not a lot of difference between Hash and B-tree indexes on a categorical column condition.

Exercise 2: Join query

Let's do the same exercise on a query that produces the same result but that uses the values of the `energy_sources` table.

The query is :

```sql
EXPLAIN ANALYZE SELECT *
FROM production_entities pe
join energy_sources es on es.id = pe.energy_source_id
WHERE es.source_name = 'Solar';
```

- EXPLAIN ANALYZE the query without any index on either `energy_source_id` or `source_name`
- create a B-tree index on energy*sources source*name and compare
- create a Hash index on energy*sources source*name and compare

Exercise 3: Range Condition with B-tree Index

Run the following query and use EXPLAIN ANALYZE to analyze its performance:

```sql
EXPLAIN ANALYZE
SELECT * FROM production_entities
WHERE capacity_mw BETWEEN 100 AND 500;
```

Now create a B-tree index on the `capacity_mw` column:

```sql
CREATE INDEX idx_btree_capacity_mw ON production_entities USING BTREE (
```

Run the EXPLAIN ANALYZE query again and compare the results. What is the index not put to use ?

Now reduce the range in the query until the the planner uses the index.

What kind of boost do you get in terms of execution time ?

Exercise 4: Multi-column Index

Run the following query and use EXPLAIN to analyze its performance:

```sql
EXPLAIN ANALYZE
SELECT * FROM production_entities
WHERE country_id = 1 AND built_date > '2010-01-01';
```

Create a multi-column B-tree index:

```sql
CREATE INDEX idx_btree_country_built_date ON production_entities USING
```

- Run the EXPLAIN ANALYZE query again and compare the results.
- What if the condition is just based on the country_id (`WHERE country_id = 1 ;`) ?
- What if the condition is just based on the built_date (`WHERE built_date > '2010-01-01';`) ?

Exercise 5: Function-based Index

Run the following query and use EXPLAIN to analyze its performance:

```sql
EXPLAIN ANALYZE
SELECT * FROM production_entities
WHERE EXTRACT(YEAR FROM built_date) = 2015;
```

Create a function-based index:

```sql
CREATE INDEX idx_btree_built_year ON production_entities USING BTREE (E
```

Run the EXPLAIN ANALYZE query again and compare the results.

Exercise 6: HASH Index on JOIN Condition

Run the following query and use EXPLAIN ANALYZE to analyze its performance:

```sql
EXPLAIN ANALYZE
SELECT pe.entity_name, epd.date, epd.energy_produced_mwh
FROM energy_production_daily epd
JOIN production_entities pe ON epd.production_entity_id = pe.id
WHERE epd.date BETWEEN '2023-01-01' AND '2023-12-31';
```

Create a HASH index on the `production_entity_id` column in the `energy_production_daily` table:

```sql
CREATE INDEX idx_hash_production_entity_id ON energy_production_daily U
```

Run the EXPLAIN ANALYZE query again and compare the results.

Optionally, create a B-tree index on the `date` column to further improve performance:

```sql
CREATE INDEX idx_btree_date ON energy_production_daily USING BTREE (dat
```

Run the EXPLAIN ANALYZE query a third time and compare the results.

For this exercise, students should:

Other experiments

You can experiment with other types of queries with different

- complexity : Joins, CTEs etc
- selectivity of the query: how many rows it returns
- conditions on different types of data. for instance dates columns.
- different operators: BETWEEN

and also increasing the volume of generated data in the tables `production_entities` and `energy_production_daily` ;