**INVESTIGATION B-tree, Hash, GIN, GIST INDEXes:**

My investigation represents insert of big amount of rows into 3 different tables which have relations between each other.

I created 3 tables without indexes (exact script you can find in database\_tables.sql file):

Students (student\_id, name, surname, birth\_date, phone\_number, created\_datetime),

Subject (subject\_id, subject\_name, mentor\_name),

exam\_result (student\_id, subject\_id, mark );

**REQUIREMENTS:**

Analyze performance for each of the indexes (use ANALYZE and EXPLAIN) .Check the size of the index. Try to set index before inserting test data and after. What was the time? Test data:

a. 100K of users

b. 1K of subjects

c. 1 million of marks

**INVESTIGATION:**  
  
The most popular indexes are B-tree, Hash, GIN and Gist.

-B-tree is index by default when we want to create index. It uses operators such as <, <=, =, =>, >. Row should have single key. This type of index sorts data in db;

-Hash index can be used only for integer data type and uses only operator ‘=’;

-GIN index can be used only for data type tsvector (text). Row in db might be as array or presented as text with multiply words;

-GIST is used for geometric data types and full-text;

In my investigation I’ve used 3 tables and 3 types of indexes.

1. **Inserting 100K of students (B-tree index)**

For inserting big amount of rows I create functions:

*CREATE FUNCTION insert\_data\_into\_students\_table() RETURNS VOID LANGUAGE PLPGSQL AS $$*

*DECLARE name TEXT= INITCAP(generate\_student\_random\_name());*

*DECLARE surname TEXT= INITCAP(generate\_random\_surname());*

*DECLARE birth\_date DATE= CAST( NOW() - INTERVAL '100 year' \* RANDOM() AS DATE);*

*DECLARE phone\_number BIGINT=CAST(1000000000 + FLOOR(RANDOM() \* 9000000000) AS BIGINT);*

BEGIN

INSERT INTO students (name, *surname, birth\_date, phone\_number*) VALUES (name, *surname, birth\_date, phone\_number*);

END;

*CREATE FUNCTION generate\_student\_random\_name() RETURNS TEXT LANGUAGE SQL AS $$*

*select (array['Anastasiya', 'Volha', 'Siri','Jim','Peter','Tony','Rosa','Sam', 'Don','Ron','Fred', 'Sasha','Sara', 'Ron','Dima','Pavel'])[random()\*(16-1)+1];*

*$$;*

*CREATE FUNCTION generate\_random\_surname () RETURNS TEXT LANGUAGE SQL AS $$*

*Select (array['Marchuk', 'Petrov', 'Solviev','Pupkin','Zajcev','Shyshko','Osipov', 'Bobrov', 'Petrosiuk','Sorokin','Solovej', 'Michaluk','Moroz','Yard'])[random()\*(14-1)+1];*

*$$;*

*BEGIN*

*INSERT INTO students (name, surname, birth\_date , phone\_number ) VALUES (first\_name, last\_name, date\_of\_birth, mobile\_no);*

*END;*

On thirst step on investigation, I try to analyze insert into table without indexes.

**Insert 100000 rows without index:**

*EXPLAIN Select insert\_data\_into\_students\_table()*

*from generate\_series(1, 100000);*

EXPLAIN does not insert data into table. It just gives information about costs, rows and width of table*.*

Graphical user interface, text, application, email

Description automatically generated

While we use EXPLAIN ANALYZE and indeed insert data and can also monitor the time it was needed to execute operation.

Graphical user interface, text, application, email

Description automatically generated

As we can see planning time is much lower than execution. And process takes longer time than planned.

Let’s try to add index to the table. I chose B-tree index:

*CREATE INDEX ix\_students\_name\_surname*

*ON students (name, surname);*

After adding index, we can compare the result.

**Insert rows with B-tree index:**

Graphical user interface, text, application, email

Description automatically generated

Execution time is bigger. We can make conclusion that insert will takes more time if we have index to the table. But let’s try to make selection all students with name ‘Peter’ and Surname ‘Yard’

**Selection with index:**

*EXPLAIN (ANALYZE) Select \* from students where name='Peter' and surname='Yard';*

Graphical user interface, text, application, email

Description automatically generated

Excecution time is 0.240 ms.

Now I remove index *ix\_students\_name\_surname* and run the same script. And we can see advantage using index – higher productivity and less execution time. If we compare execution time of selection – without index it takes much more time.

**Selection without index:**

Graphical user interface, text, application, email

Description automatically generated

1. **Inserting 1K of subjects (GIN index)**

For inserting 1000 rows into subject table I’ve created functions:

CREATE FUNCTION insert\_data\_into\_subject\_table() RETURNS VOID LANGUAGE PLPGSQL AS $$

DECLARE subject\_name TEXT= INITCAP(generate\_random\_subject());

DECLARE mentor\_name TEXT= INITCAP(generate\_mentor\_random\_name());

BEGIN

INSERT INTO subject (subject\_name, mentor\_name) VALUES (subject\_name, mentor\_name);

END;

CREATE FUNCTION generate\_random\_subject() RETURNS TEXT LANGUAGE SQL AS $$

select (array['economics', 'medecine', 'computer science','mathematics','algebra','biology', 'chemistry'])[ random()\*(7-1)+1];

$$;

CREATE FUNCTION generate\_mentor\_random\_name() RETURNS TEXT LANGUAGE SQL AS $$

select (array['Alex A.', 'Alex B.', 'Alex C.', 'John M.', 'Sara A.', 'Sara B.','Jimmy K.','Kate L.','Leyla N.'])[random()\*(9-1)+1];

$$;

**Insert rows without index:**

*EXPLAIN ANALYZE Select insert\_data\_into\_subject\_table()*

*from generate\_series(1, 1000);*

Graphical user interface, text, application, email

Description automatically generated

Then I added a column to the table (to use it for GIN index). And create GIN index:

*ALTER TABLE subject ADD mentor\_index tsvector;*

*Create index on subject using gin(*mentor\_index*);*

**And insert 1000 rows with index:**

Graphical user interface, text, application, email

Description automatically generated

**Select rows with index:**

I try to select subjects with mentor’s name Alex A. I use exact and partial match. Execution time is lower with index for exact match, but for partial it is faster without index. In this case chose of index depends on what type of match usually is used at work.

*EXPLAIN ANALYZE Select \* from subject where mentor\_name='Alex A.';*

Graphical user interface, text, application, email

Description automatically generated

*EXPLAIN ANALYZE Select \* from subject where mentor\_name='Alex %';*

Graphical user interface, text, application, email

Description automatically generated

**Select rows without index:**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. **Inserting 1 000 000 of mark (Hash index)**

For inserting 1 000 000 rows into exam\_result table I’ve created functions:

*CREATE FUNCTION insert\_data\_into\_exam\_result() RETURNS TABLE LANGUAGE SQL AS $$*

*DECLARE student\_id INT= INITCAP(generate\_random\_student\_id());*

*DECLARE subject\_id INT= INITCAP(generate\_random\_subject\_id());*

*DECLARE mark INT= INITCAP(generate\_random\_mark());*

*BEGIN*

*INSERT INTO subject (student\_id, subject\_id, mark ) VALUES (student\_id, subject\_id, mark);*

*END;*

*CREATE FUNCTION get\_random\_student\_id() RETURNS INT LANGUAGE SQL AS $$*

*Select student\_id from students*

*order by random() limit 1;*

*$$;*

*CREATE FUNCTION generate\_random\_subject\_id() RETURNS INT LANGUAGE SQL AS $$*

*Select subject\_id from subject*

*order by random() limit 1;*

*$$;*

CREATE FUNCTION generate\_mentor\_random\_name() RETURNS TEXT LANGUAGE SQL AS $$

Select floor (random()\*(10-1+1)+1);

$$;

**Insert rows without index:**

*EXPLAIN ANALYZE Select insert\_data\_into\_exam\_result ()*

*from generate\_series(1, 1000000);*

Graphical user interface, text, application, email

Description automatically generated

Then I added HASH index:

*Create index on exam\_result using hash(student\_id);*

**And insert 1000000 rows with index:**

Graphical user interface, text, application, email

Description automatically generated

**Select rows without index:**

*EXPLAIN (ANALYZE) Select \* from exam\_result where student\_id=* *302940;*

Graphical user interface, text, application, email

Description automatically generated

**Select rows with index:**

Graphical user interface, text, application, email

Description automatically generated

Hash index helps to speed up selection but also need more time for insertion. Constraint for this type of index is using only operator ’=’. And it is not suitable for each selection during work.

**Size of all indexes:**

To look at all indexes, tables and size run the script:

*SELECT*

*relname as name\_tables,*

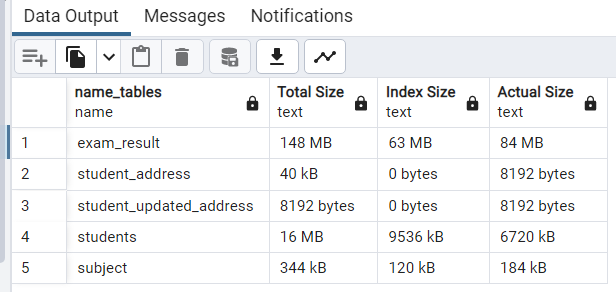
*pg\_size\_pretty(pg\_total\_relation\_size(relid)) As "Total Size",*

*pg\_size\_pretty(pg\_indexes\_size(relid)) as "Index Size",*

*pg\_size\_pretty(pg\_relation\_size(relid)) as "Actual Size"*

*FROM pg\_catalog.pg\_statio\_user\_tables*

*ORDER BY 1;*



**Conclusion.**

Using indexes has both advantages and disadvantages. From pluses I can notice: speeding up searching in db, sorting data (B-tree), possibility to group by data, using column with unique values (primary key). Also from disadvantages side: require additional disk space, slow inserting, updating and deleting data in db;