1. Inserting test data without indexes and constraints
    - `students` - 100 000 rows

	QUERY PLAN
1	Insert on students (cost=0.00..13000.00 rows=0 width=0) (actual time=1179.865..1179.865 rows=0 loops=1)
2	-> Subquery Scan on "*SELECT*" (cost=0.00..13000.00 rows=100000 width=526) (actual time=15.627..698.997 rows=100000 loops=1)
3	-> Function Scan on generate_series number (cost=0.00..9500.00 rows=100000 width=124) (actual time=15.559..545.245 rows=100000 loops=1)
4	Planning Time: 0.158 ms
5	Execution Time: 1186.503 ms

    - `subjects`

	QUERY PLAN
1	Insert on subjects (cost=0.00..47.50 rows=0 width=0) (actual time=5.756..5.757 rows=0 loops=1)
2	-> Subquery Scan on "*SELECT*" (cost=0.00..47.50 rows=1000 width=240) (actual time=0.176..2.647 rows=1000 loops=1)
3	-> Function Scan on generate_series (cost=0.00..30.00 rows=1000 width=64) (actual time=0.127..2.104 rows=1000 loops=1)
4	Planning Time: 0.040 ms
5	Execution Time: 5.787 ms

    - `results`

	QUERY PLAN
1	Result (cost=0.00..0.26 rows=1 width=4) (actual time=5558.537..5558.538 rows=1 loops=1)
2	Planning Time: 0.013 ms
3	Execution Time: 5558.546 ms

2. Inserting test data with indexes
    - `students`

	QUERY PLAN
1	Insert on students (cost=0.00..16750.00 rows=0 width=0) (actual time=21024.053..21024.054 rows=0 loops=1)
2	-> Subquery Scan on "*SELECT*" (cost=0.00..16750.00 rows=100000 width=534) (actual time=14.963..1122.955 rows=100000 loops=1)
3	-> Function Scan on generate_series n (cost=0.00..13750.00 rows=100000 width=152) (actual time=14.890..896.733 rows=100000 loops=1)
4	SubPlan 1
5	-> Result (cost=0.00..0.04 rows=1 width=32) (actual time=0.001..0.001 rows=1 loops=100000)
6	Planning Time: 0.114 ms
7	Execution Time: 21025.736 ms

    - `subjects`

	QUERY PLAN
1	Insert on subjects (cost=0.00..47.50 rows=0 width=0) (actual time=23.936..23.937 rows=0 loops=1)
2	-> Subquery Scan on "*SELECT*" (cost=0.00..47.50 rows=1000 width=240) (actual time=0.156..3.692 rows=1000 loops=1)
3	-> Function Scan on generate_series (cost=0.00..30.00 rows=1000 width=64) (actual time=0.115..2.944 rows=1000 loops=1)
4	Planning Time: 0.037 ms
5	Execution Time: 23.965 ms

- `results`

	QUERY PLAN
1	Result (cost=0.00..0.26 rows=1 width=4) (actual time=33058.566..33058.567 rows=1 loops=1)
2	Planning Time: 0.015 ms
3	Execution Time: 33058.577 ms

3. Size of indexes
    - `students`

	pg_size_pretty
1	41 MB

    - `subjects`

	pg_size_pretty
1	456 kB

    - `results`

	pg_size_pretty
1	36 MB

4. Test queries without indexes
    - Find user by name (exact match)

	QUERY PLAN
1	Seq Scan on students (cost=0.00..3473.00 rows=1 width=143) (actual time=6.209..12.452 rows=1 loops=1)
2	Filter: ((name)::text = '3ad2c7e5bb959f5b6fd092a4b31cbfef'::text)
3	Rows Removed by Filter: 99999
4	Planning Time: 1.512 ms
5	Execution Time: 12.467 ms

    - Find user by surname (partial match)

	QUERY PLAN
1	Seq Scan on students (cost=0.00..3473.00 rows=10 width=143) (actual time=1.631..247.216 rows=39 loops=1)
2	Filter: ((surname)::text ~~* '%e76a%'::text)
3	Rows Removed by Filter: 99961
4	Planning Time: 0.297 ms
5	Execution Time: 247.245 ms

- Find user by phone number (partial match)

	QUERY PLAN
1	Seq Scan on students (cost=0.00..3473.00 rows=10 width=143) (actual time=0.351..177.123 rows=271 loops=1)
2	Filter: ((phone_numbers)::text ~~* '%1%67'::text)
3	Rows Removed by Filter: 99729
4	Planning Time: 0.331 ms
5	Execution Time: 177.168 ms

- Find user with mark by user surname (partial match)

	QUERY PLAN
1	Nested Loop (cost=4.50..3907.37 rows=100 width=147) (actual time=1.893..251.617 rows=390 loops=1)
2	-> Seq Scan on students s (cost=0.00..3473.00 rows=10 width=143) (actual time=1.758..247.727 rows=39 loops=1)
3	Filter: ((surname)::text ~~* '%e76a%'::text)
4	Rows Removed by Filter: 99961
5	-> Bitmap Heap Scan on results r (cost=4.50..43.34 rows=10 width=8) (actual time=0.057..0.073 rows=10 loops=39)
6	Recheck Cond: (student_id = s.id)
7	Heap Blocks: exact=390
8	-> Bitmap Index Scan on pk_results (cost=0.00..4.50 rows=10 width=0) (actual time=0.051..0.051 rows=10 loops=39)
9	Index Cond: (student_id = s.id)
10	Planning Time: 3.877 ms
11	Execution Time: 251.684 ms

5. Test queries with indexes
   - Find user by name (exact match)

	QUERY PLAN
1	Index Scan using ix_students_name on students (cost=0.00..8.02 rows=1 width=143) (actual time=0.009..0.010 rows=1 loops=1)
2	Index Cond: ((name)::text = '3ad2c7e5bb959f5b6fd092a4b31cbfef'::text)
3	Planning Time: 0.178 ms
4	Execution Time: 0.023 ms

   - Find user by surname (partial match)

	QUERY PLAN
1	Seq Scan on students (cost=0.00..3473.00 rows=10 width=143) (actual time=1.673..245.729 rows=39 loops=1)
2	Filter: ((surname)::text ~~* '%e76a%'::text)
3	Rows Removed by Filter: 99961
4	Planning Time: 0.394 ms
5	Execution Time: 245.759 ms

   - Find user by phone number (partial match)

```
QUERY PLAN
Seq Scan on students  (cost=0.00..3473.00 rows=10 width=143) (actual time=0.290..163.258 rows=271 loops=1)
  Filter: ((phone_numbers)::text ~~* '%1%67'::text)
  Rows Removed by Filter: 99729
Planning Time: 0.328 ms
Execution Time: 163.300 ms
```

- Find user with marks by user surname (partial match)

```
QUERY PLAN
Nested Loop  (cost=4.50..3907.37 rows=100 width=147) (actual time=1.649..242.143 rows=390 loops=1)
  ->  Seq Scan on students s  (cost=0.00..3473.00 rows=10 width=143) (actual time=1.594..239.956 rows=39 loops=1)
        Filter: ((surname)::text ~~* '%e76a%'::text)
        Rows Removed by Filter: 99961
  ->  Bitmap Heap Scan on results r  (cost=4.50..43.34 rows=10 width=8) (actual time=0.017..0.026 rows=10 loops=39)
        Recheck Cond: (student_id = s.id)
        Heap Blocks: exact=390
        ->  Bitmap Index Scan on pk_results  (cost=0.00..4.50 rows=10 width=0) (actual time=0.011..0.011 rows=10 loops=39)
              Index Cond: (student_id = s.id)
Planning Time: 0.930 ms
Execution Time: 242.227 ms
```

As a result of my investigation I can say that the indexes are useful and help to speed up SELECT queries and WHERE clauses. PostgreSQL has a lot of indexes such as B-tree, HASH, GIN, GIST. Each of them is useful in special cases.

B-tree indexes are default and best used for specific value searches, scanning ranges, data sorting or pattern matching.

HASH indexes are best suited to work with equality operators, especially for exact matches.

GIN indexes are best suited for data types such as JSONB, Array, Range types and full-text search.

GIST indexes are best for complex data.

These rules were tested on the real data. It works. The test queries work faster than before. But as a result, the data is inserted slowly.