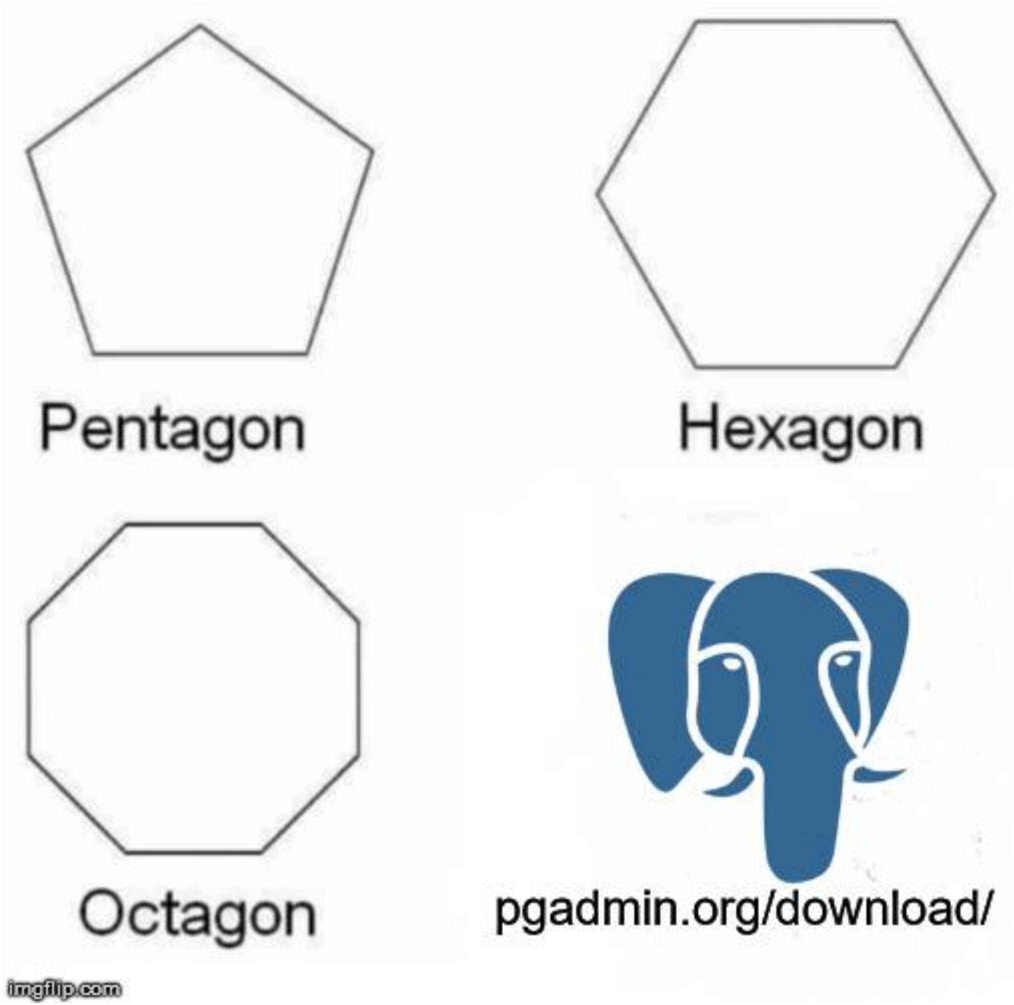

Pentagon
Hexagon
Octagon
pgadmin.org/download/
imgflip.com

Databases - Tutorial 08
Query optimization & Indexes

Hamza Salem - Innopolis University


TA
Did you install PostgreSQL?
STUDENT
I dont know
what is PostgreSQL
imgflip.com

Contents

- Last normalization types
- Query optimization
- Indexes

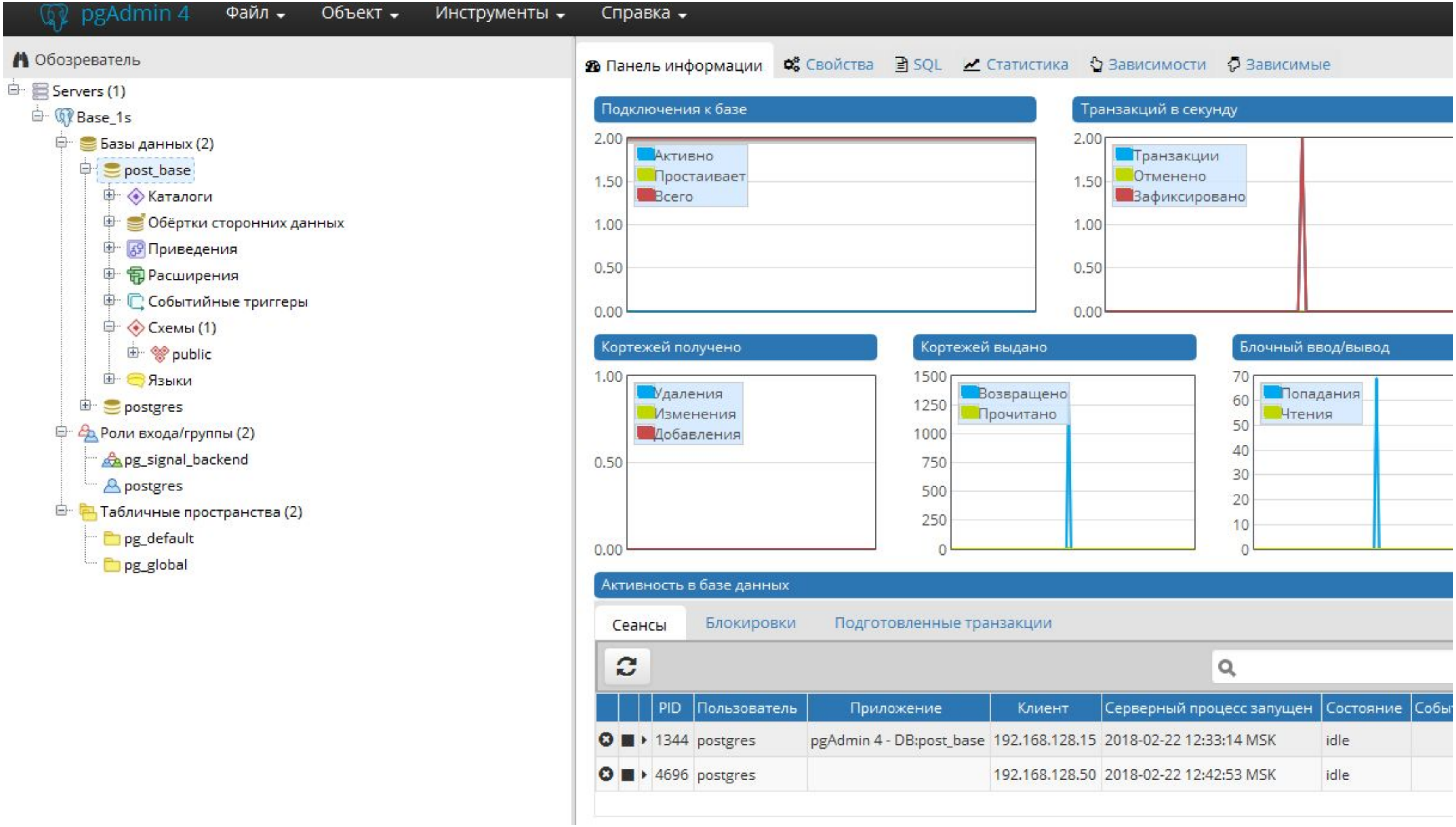
Assignment 1 is very hard

Other courses in Innopolis:

Friend: How was the exam?

Me: I did it well...

My answers:


pgAdmin 4
Файл
Объект
Инструменты
Справка
Обозреватель
Servers (1)
Base_1s
Базы данных (2)
post_base
Каталоги
Обёртки сторонних данных
Приведения
Расширения
Событийные триггеры
Схемы (1)
public
Языки
postgres
Роли входа/группы (2)
pg_signal_backend
postgres
Табличные пространства (2)
pg_default
pg_global
Панель информации
Свойства
SQL
Статистика
Зависимости
Зависимые
Подключения к базе
Активно
Простаивает
Всего
Транзакций в секунду
Транзакции
Отменено
Зафиксировано
Кортежей получено
Удаления
Изменения
Добавления
Кортежей выдано
Возвращено
Прочитано
Блочный ввод/вывод
Попадания
Чтения
Активность в базе данных
Сеансы
Блокировки
Подготовленные транзакции
PID
Пользователь
Приложение
Клиент
Серверный процесс запущен
Состояние
1344 postgres pgAdmin 4 - DB:post_base 192.168.128.15 2018-02-22 12:33:14 MSK idle
4696 postgres 192.168.128.50 2018-02-22 12:42:53 MSK idle

Boyce-Codd Normal Form (BCNF)

For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:

1. It should be in the Third Normal Form.
2. And, for any dependency A → B, **A should be a super key.**

Super key vs primary key

Primary key

Customer ID	Forename	Surname
1	Simon	Jones
2	Emma	Price
3	Laura	Jones
4	Jonathan	Hale
5	Emma	Smith

Simple primary key

Super key

Super Key 1

Super Key 2

Roll No.	Name	Mobile No	Age
1	Anoop	9873xxxxx	21
2	Anurag	9874xxxxx	22
3	Ganesh	9560xxxxx	23

Dependency A ➡ B, A should be a super key

This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the 2nd Normal Form as there is no Partial Dependency.

And, there is no Transitive Dependency, hence the table also satisfies the 3rd Normal Form.

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

Dependency A ➜ B, A should be a super key

student_id, subject form primary key

But, there is one more dependency, professor → subject.

And while subject is a prime attribute, professor is a non-prime attribute, which is not allowed by BCNF

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++

student_id	p_id
101	1
101	2

Fourth Normal Form (4NF)

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the Boyce-Codd Normal Form.
2. And, the table should not have any **Multi-valued Dependency**.

Multi-valued Dependency

1. **For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.**
2. **Also, a table should have at-least 3 columns for it to have a multivalued dependency.**
3. **And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.**

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

s_id	course
1	Science
1	Maths
2	C#
2	Php

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Tips design your query

- SELECT fields instead of using SELECT *
  - Write the query as

SELECT id, first_name, last_name, age, subject FROM student_details;

  - Instead of:

SELECT * FROM student_details;

Tips design your query

- Use operator **EXISTS, IN** and table joins appropriately in your query.
    - Usually IN has the slowest performance.
    - IN is efficient when most of the filter criteria is in the sub-query.
    - EXISTS is efficient when most of the filter criteria is in the main query.

Tips design your query

- Use **WHERE** instead of **HAVING** to define filters
  - Write the query as

```
SELECT subject, count(subject)
FROM student_details
WHERE subject != 'Science'
AND subject != 'Maths'
GROUP BY subject;
```

  - Instead of:

```
SELECT subject, count(subject)
FROM student_details
GROUP BY subject
HAVING subject!= 'Vancouver' AND subject!= 'Toronto';
```

Tips design your query

- Minimize the number of subqueries on your main query
  - Write the query as

```
SELECT name
FROM employee
WHERE (salary, age ) = (SELECT MAX (salary), MAX (age)
FROM employee_details)
AND dept = 'Electronics';
```

  - Instead of:

```
SELECT name
FROM employee
WHERE salary = (SELECT MAX(salary) FROM employee_details)
AND age = (SELECT MAX(age) FROM employee_details)
AND emp_dept = 'Electronics';
```

Tips design your query

- Use **wildcards at the end of a phrase only**
  - Write the query as

  SELECT City FROM Customers

  WHERE City LIKE ‘Char%’

  - Instead of

  SELECT City FROM Customers

  WHERE City LIKE ‘%Char%’

Query optimization

- It refers to the process by which the best execution strategy for a given query is found from a set of alternatives. Query optimization is a part of query processing.
- The main aims of query optimization are to choose a transformation that minimizes resource usage, reduce total execution time of query and also reduce response time of query.

Use of cost-based optimization

- What is cost?
    - Access cost to secondary storage: disk I/O cost. It depends on the type of access structures (ordering, hashing, primary, secondary indexes)
        - Focused in LARGE databases
    - Cost of storing on disk intermediate files (contrast with pipeline)
    - Computation cost
        - Focused in small databases
    - Number of memory buffers and block transfers between disk and main memory

How to estimate query cost on PostgreSQL http://morningcoffee.io/the-postgresql-query-cost-model.html

What is explain?

- EXPLAIN is a keyword that gets prepended to a query to show a user how the query planner plans to execute the given query.
- Depending on the complexity of the query, it will show the join strategy, method of extracting data from tables, estimated rows involved in executing the query, and a number of other bits of useful information.

```
EXPLAIN
select *
from actor a
where a.first_name like
'J%';
```

	QUERY PLAN text
1	Seq Scan on actor a (cost=0.00..4.50 rows=23 width=25)
2	Filter: ((first_name)::text ~~ 'J%'::text)

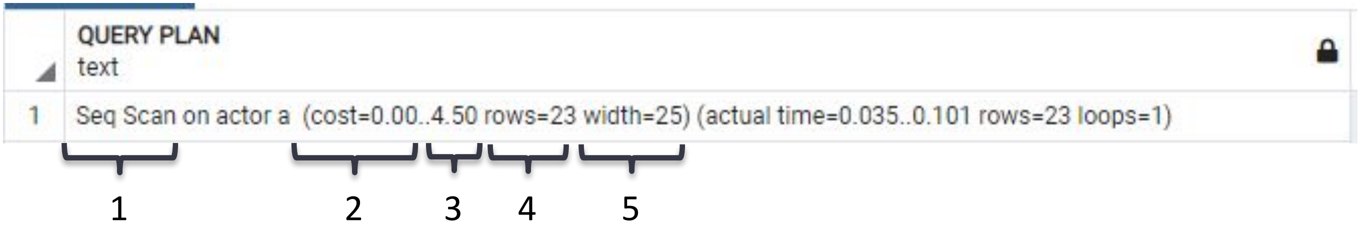
What is explain?

- Used with ANALYZE, EXPLAIN will also show the time spent on executing the query, sorts, and merges that couldn't be done in-memory, and more.

```
EXPLAIN ANALYZE
select *
from actor a
where a.first_name like
'J%';
```

	QUERY PLAN text
1	Seq Scan on actor a (cost=0.00..4.50 rows=23 width=25) (actual time=0.035..0.101 rows=23 loops=1)
2	Filter: ((first_name)::text ~~ 'J%'::text)
3	Rows Removed by Filter: 177
4	Planning Time: 0.217 ms
5	Execution Time: 0.121 ms

Explain result



1. Types of scan nodes: sequential scans, index scans, and bitmap index scans (depends on the table access methods)
2. Estimated start-up cost (time expended before the output scan can start, e.g., time to do the sorting in a sort node)
3. **Estimated total cost (if all rows are retrieved, though they might not be; e.g., a query with a LIMIT clause will stop short of paying the total cost of the Limit plan node's input node)**
4. Estimated number of rows output by this plan node (again, only if executed to completion)
5. Estimated average width (in bytes) of rows output by this plan node

Index creation

- Syntax

```
CREATE INDEX index_name ON table_name [USING method]
(column_name [ASC | DESC] [NULLS {FIRST | LAST }]);
```

- Example

```
CREATE INDEX idx_address_phone ON address(phone);
```

Impact of indexing

```
EXPLAIN SELECT *
FROM address
WHERE phone =
'223664661973';
```

	QUERY PLAN text
1	Seq Scan on address (cost=0.00..15.54 rows=1 width=61)
2	Filter: ((phone)::text = '223664661973'::text)

After creating the index

```
EXPLAIN SELECT *
FROM address
WHERE phone =
'223664661973';
```

	QUERY PLAN text
1	Index Scan using idx_address_phone on address (cost=0.28..8.29 rows=1 width=61)
2	Index Cond: ((phone)::text = '223664661973'::text)


B
Gin
100ml EAU DE PARFUM
#
GIST develop in the exterior
areas of the stomach wall

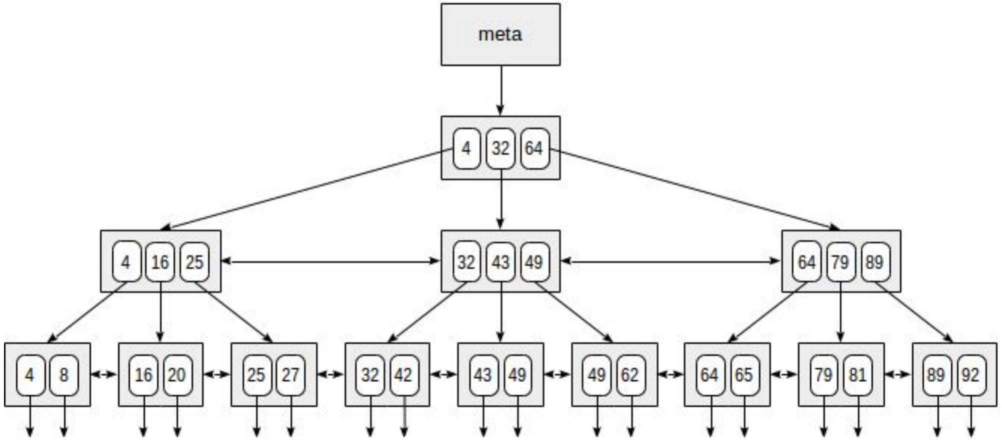
Generalized Inverted Indexes

Generalized Inverted Indexes

Generalized Search Tree indexes

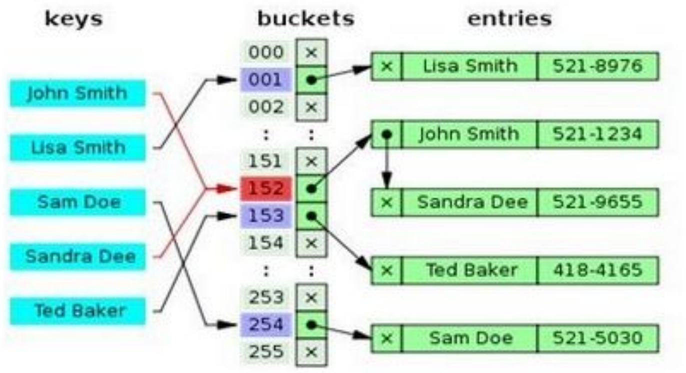
B tree Index

- B-tree index type, implemented as «btree» access method, is suitable for data that can be sorted. In other words, **«greater», «greater or equal», «less», «less or equal», and «equal» operators** must be defined for the data type. Note that the same data can sometimes be sorted differently.
- <   <=   =   >=   >



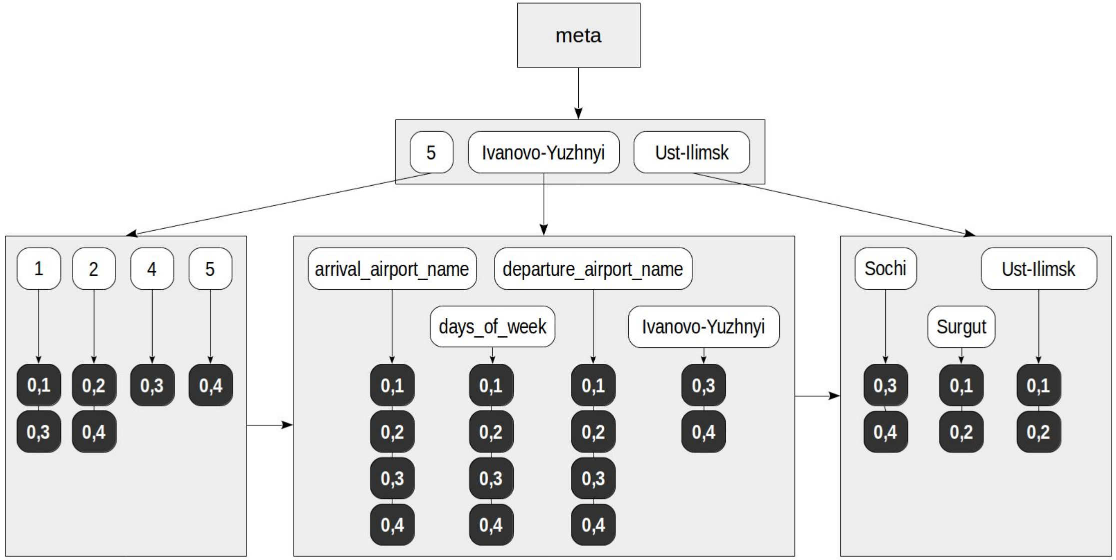
Hash Index

- The idea of hashing is to associate a small number (from 0 to *N*–1, *N* values in total) with a value of any data type. Association like this is called *a hash function*. The number obtained can be used as an index of a regular array where references to table rows (TIDs) will be stored. Elements of this array are called *hash table buckets* — one bucket can store several TIDs if the same indexed value appears in different rows.
- =



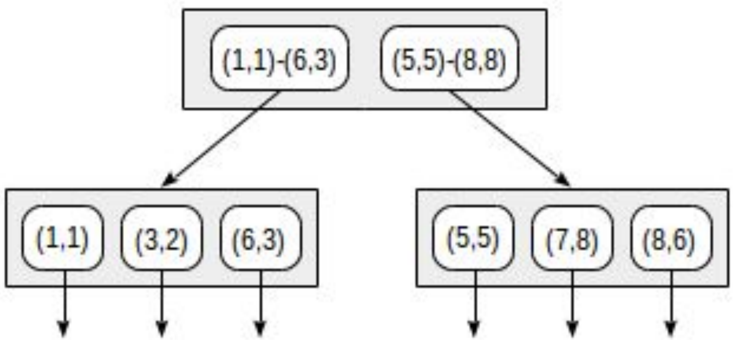
GIN index

- GIN (or Generalized Inverted Indexes) are useful when an index must map many values to one row, whereas B-Tree indexes are optimized for when a row has a single key value. GINs are good for indexing array values **as well as for implementing full-text search**.
- More about GIN index here.
- `<@    @>    =    &&`



GIST Index

- GiST (or Generalized Search Tree) indexes allow you to build general balanced tree structures, and can be used for **operations beyond equality and range comparisons.** They are used to index the **geometric data types, as well as full-text search**.
- More on GiST here.
- `<<   &<   &>   >>   <<|   &<|   |&>   |>>   @>   <@   ~=   &&`



Demo time

```
explain analyze SELECT * FROM customer2 where id =1 -- 8.31

explain analyze SELECT * FROM customer2 where id > 1 -- 1621.74

explain analyze SELECT * FROM customer2 where id > 1 and id<100 --14.33

explain analyze SELECT * FROM customer2 where id > 1 and id<1400 -- 117.45


CREATE INDEX id_idx1 ON customer2  USING hash (id)


CREATE INDEX id_idx3 ON customer2(name)


explain analyze SELECT * FROM customer2 where id =1 --8.02
```

Useful Links

- https://www.sisense.com/blog/sql-symbol-cheatsheet/