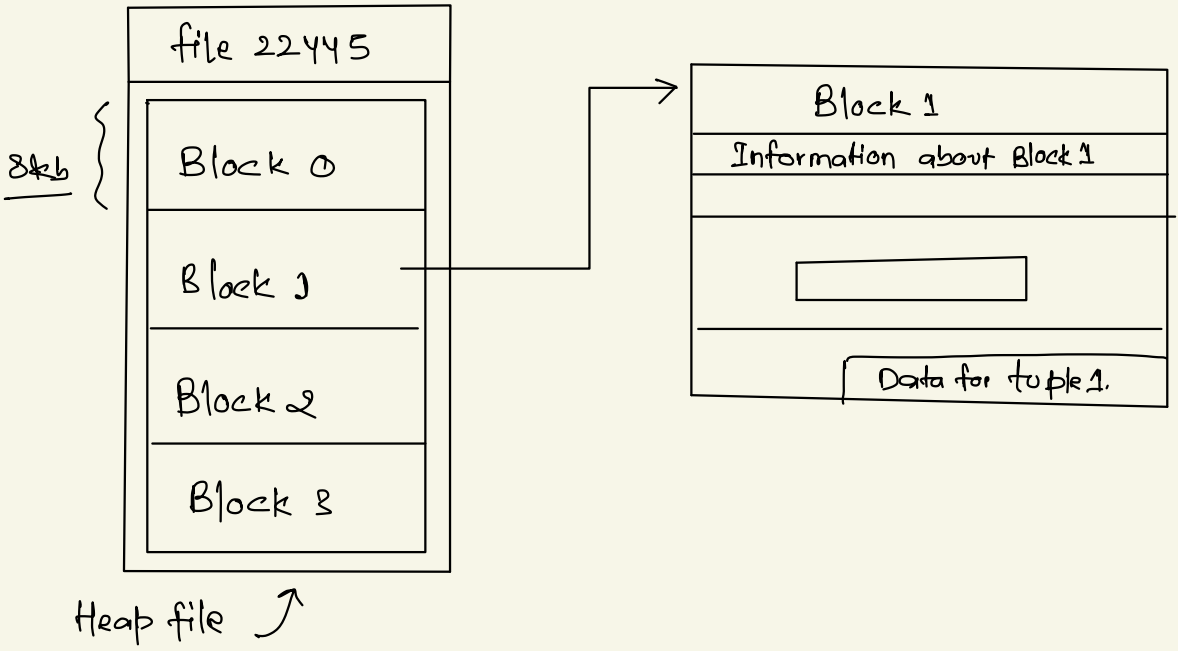
DataBase NotES

→ How Indexes increase Performance




```
Select * from users.users
where Username = 'AKASH';
```

Postgres can't just examine a file, to get this data, it first loads the file in main memory (RAM), then do some operation

***

Everytime a data loads from harddrive to main memory, it have some cost I/o cost (time taking). we need to reduce this i/o calls.

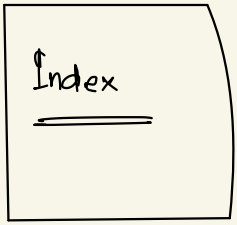
Once data loads up into the main memory, it will do a Seq. Search with condition usesname = 'AKASH';

Search one by one

↳ full table scan

Postgres has to load many (or all) rows from the heap file to memory, frequently (but not always) poor performance.

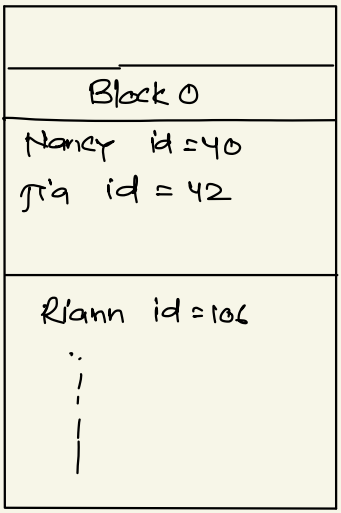
Select * from users
where usesname = 'Riann';



Data structure that efficiently tells us what block/index a record is stored at.

How Index is created

1. which column do we want to have very fast lookups.
2. Extract only the properties we want to do fast lookup by and the index/block for each.



Nancy
B = 0
I = 1

Tia
B = 0
I = 2

Riann
B = 1
I = 1

⟹ Put these data in B-tree

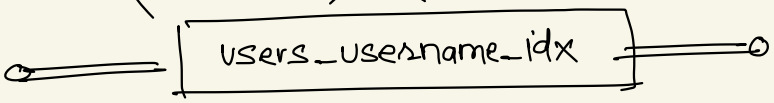
⟹ B-tree overview (Add it

Creating an Index

create index on users (username)

↳ Index got created bcz of this is

(table_name)_(col_name)_idx



If you want to apply manual_name, it would be

- create index manual_name on users (username)

To drop index

- Drop index users_username_idx ;

Bench Marking Queries

select * from users where username = 'Emil30';

Explain and Analize ⟹ Planning time : 0.119 ms

Execution time : 0.03 ms

Downsides of Indexes

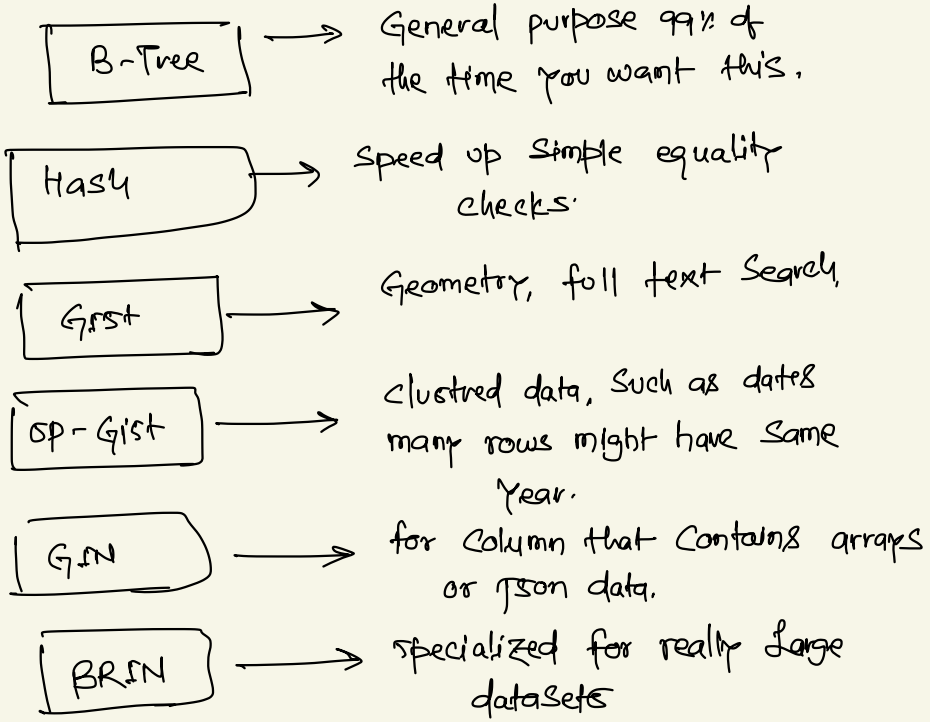
1. Storage Cost, A file gets created to store index data.

query to get the size of table.

```
Select pg_size_pretty (pg_relation_size('users'));
```

- Can be large! stores data from at least one column of real table.
- Slow down insert/update/delete – the index has to be updated.
- Index might not actually get used.

Types of Indexes

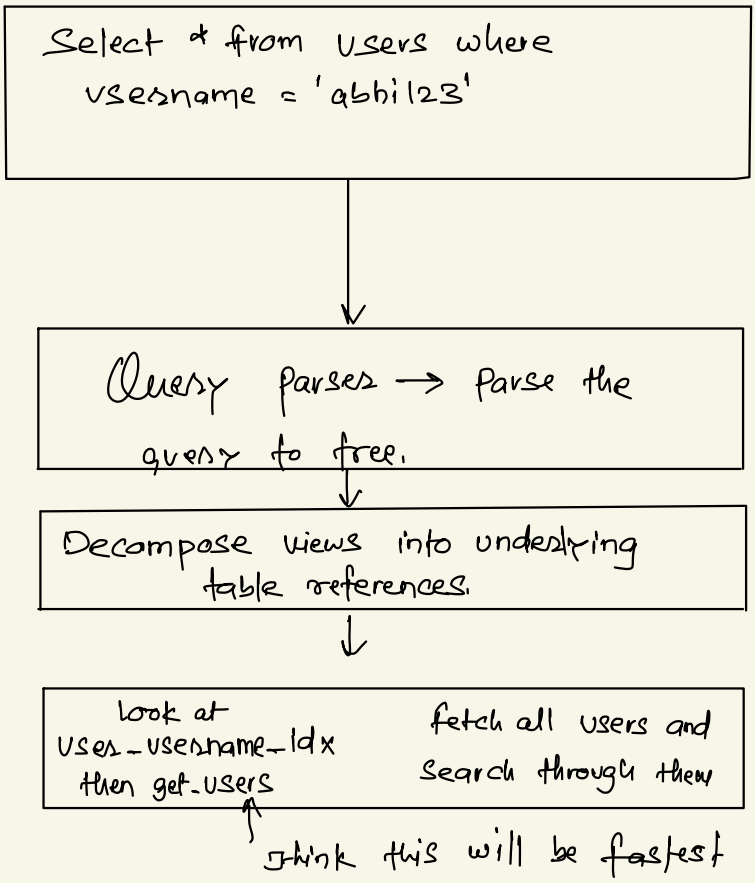


→ Automatically created indexes

- Postgres automatically creates an index for the primary key column of every table.
- for any unique constraint.
- These don't get listed in under indexes in PGAdmin

```
Select relname, relkind from pg_class
where relkind = 'i' ;
```

⟹ What happens when we feed a postgres query.



Query planner

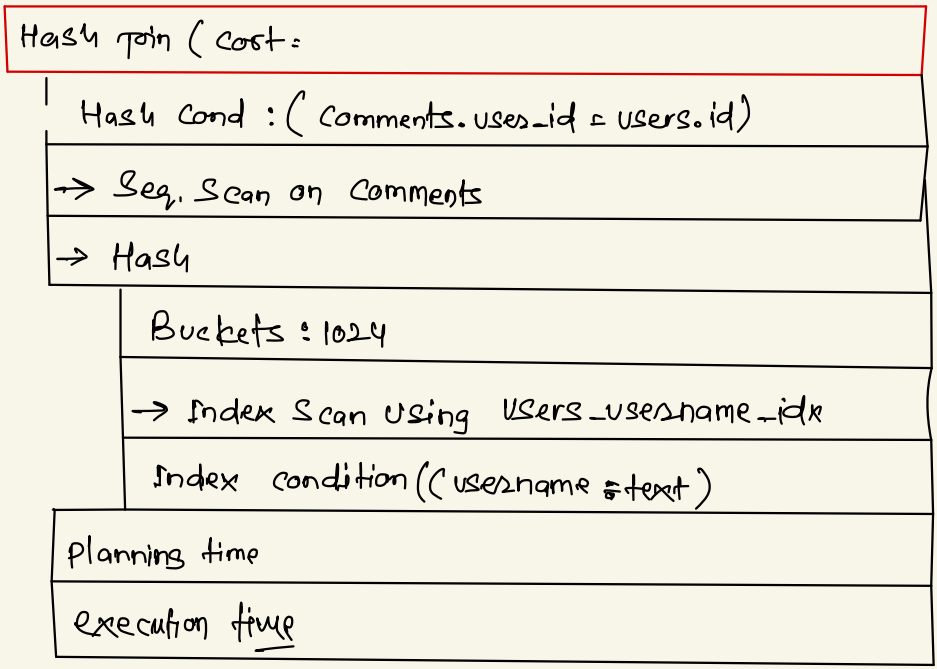
→ Explain ⟶ Build a query plan and display info about it.

→ Explain analyze

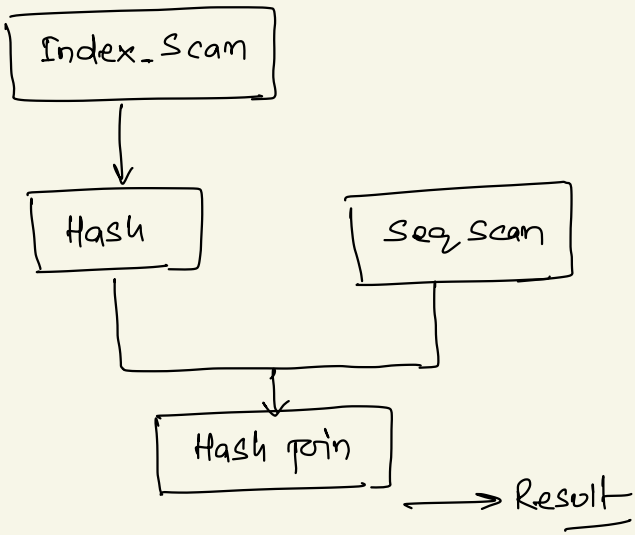
↳ ⟶ Build a query plan, run it and info about it.

These are for benchmarking + evaluating queries, not for use in real data fetching.

```
select username, contents from users
join comments on comments.user_id = users.id
  where username = 'Alyson14';
```



→ All rows with → are nodes of a query planner



Hash join (cost = 8.31 .... 1756.11) (rows = 11) (width = 81)
1. 2 3 4.

1. How this node is generating data
2. Amount of processing power required for this step.
3. A guess at how many rows this step will produce
4. A guess at avg number of bytes of each row.

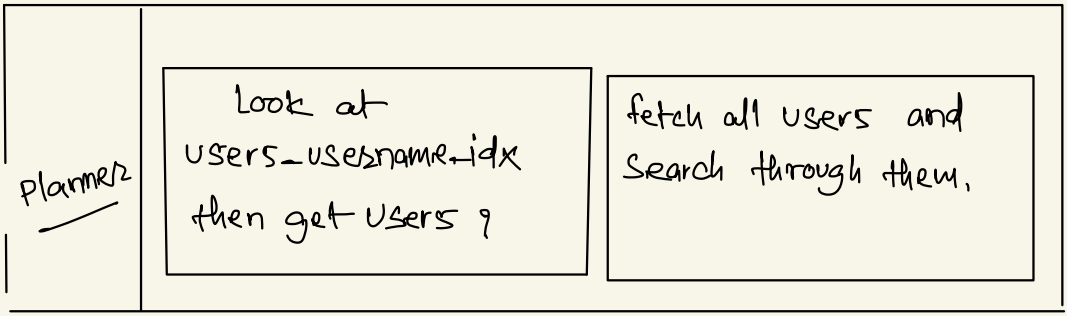
How postgres make a guess on number of rows and width ??

→ It stores the stats of the table to access it.

Select * from pg_stats where tablename = 'users';

Working definition of cost

→ Amount of time (seconds ? or milliseconds ?) to execute some part of our query plan.



How planner chooses which path to choose ??

Using index

→ open username index
→ access the root node
→ jump to the required child node
→ get the pointer to the data Block and index
→ opens up users heap
→ go to exact Block and index return the data

Using user heap file

→ open the heap file
→ open up the first block
→ go through all user rows if found, return the value
→ else repeat 2-3

→ fetching random pages in heap.

Loading data from random spots off a hard drive usually takes more time than loading data sequentially.

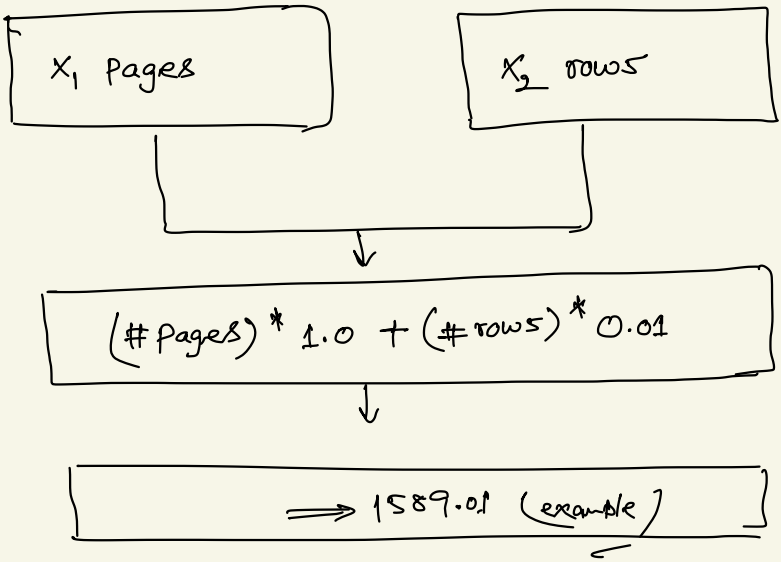
k — cost to get the data from random address from hard disc.

L → cost to get data from sequential address from H.D

k * Numbers of Blocks opened

L * Numbers of Blocks loaded

Seq. Scan Cost estimation



Cost = (# Pages read sequentially) * seq_page_cost
+ (# pages read at random) * random page cost
+ (# rows_scanned) * cpu tuple_cost
+ (# index entries scanned) * cpu_index_tuple_cost
+ (# time function/operator evaluated) *
cpu_operator_cost.

→ Approaching database design

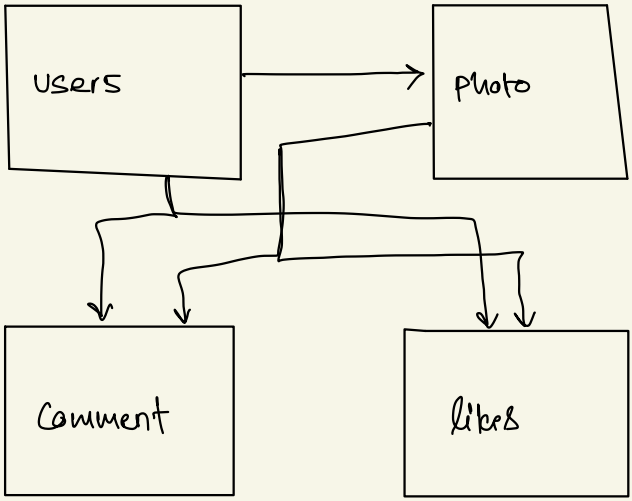
What table we should make. ?

- Common features (like authentication, comments etc) are frequently build with conventional table names and columns.

Building an upvote system ? you are not the first.

- what type of resources exist in your app.? create a separate table for each of these features.
- features that seems to indicate a relationship or ownership between two types of resources need to be reflected in our table design.

Database for photo-sharing App



→ Relationships in SQL

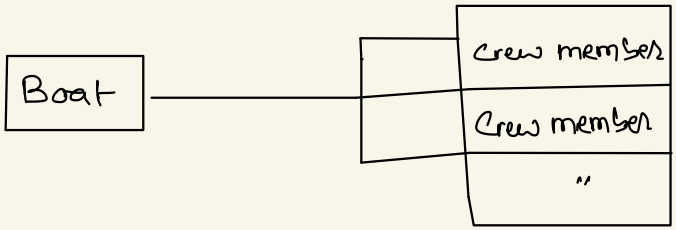
Users ⟶ Photos (one to many)

A user can have many photos, its one to many

Photos ⟶ Users (many to one)

Photo ⟶ Comment (one to many)

Comment ⟶ Photo (one to one)



A boat has many crew members

one crew member have one boat

Many to many

students ⟷ classess

Tasks ⟷ Engineers

one to one

Company ⟷ CEO

Boat ⟷ Captain

student ⟷ Desk.