Date _____ 20 ____

PostgreSQL

- writing efficient queries to retrieve information
- designing schema
- understanding when to use advanced features
- managing database in a production environment

Date _____ 20 ____

Database design process

what kind of thing are we storing? table

what properties does this thing have? column

what type of data does each of these properties contain? data type

Table collection of records

row record

Keywords

identifiers

Date _____ 20 ____

create separate table for each feature

features that seem to indicate a relationship / ownership b/w 2 types of resources need to be reflected in table design

One to Many & Many to One relationships

One to One

Many to Many

Primary Key
Foreign Key

many side has foreign key column

Primary Key

1 prim key
unique
int / UUID
never changes

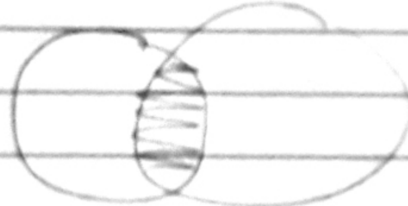
F Key

belong to another record

may rows same table can have same foreign key

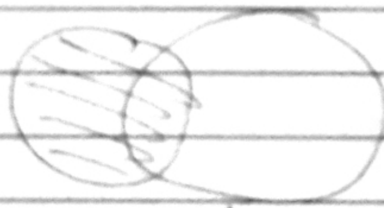
On delete Restrict
no action
cascade
Set Null
set default

Table order FROM & Join makes a difference



Join / Inner Join

Inner Join



Left Join

Left outer Join

Right Join

right outer join

Full Join

Date _____ 20 ____

grouping → many to few rows

aggregates - down to 1

Group By

↓

find all unique values

↓

take each row & assign to group based on unique values

↓

aggregate functions

sum, avg, count, min, max

Date _____ 20 ____

From → specifies starting set of rows to work with
join → merges in data from additional tables
where → filters set of rows
group by → groups rows by unique set of values
having → filters the set of groups

many to many relationships

table → join table ← table

sorting

order by ASC DESC

offset Limit

Limit Offset

Date _____ 20 ____

(Query) Union (Query)

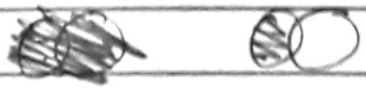
same columns for union w/ same/compatible data types

Union 

Union All

Intersect 

Intersect All

Except 

Except All

All keyword = keep duplicates

Date _____ 20 ____

Subqueries can be a src of a value (select/where)
src of rows (from/join)
src of a column (where)

scalar query 1 row 1 column

Subquery in from must use alias (AS)

Coreleted Subqueries

~~rename~~ add alias to outer query use in inner query

select w/o From if subquery returns single value

select Distinct
select Greatest
select least

Sharjeel

Date _____ 20 ____

CASE
WHEN THEN
ELSE
END

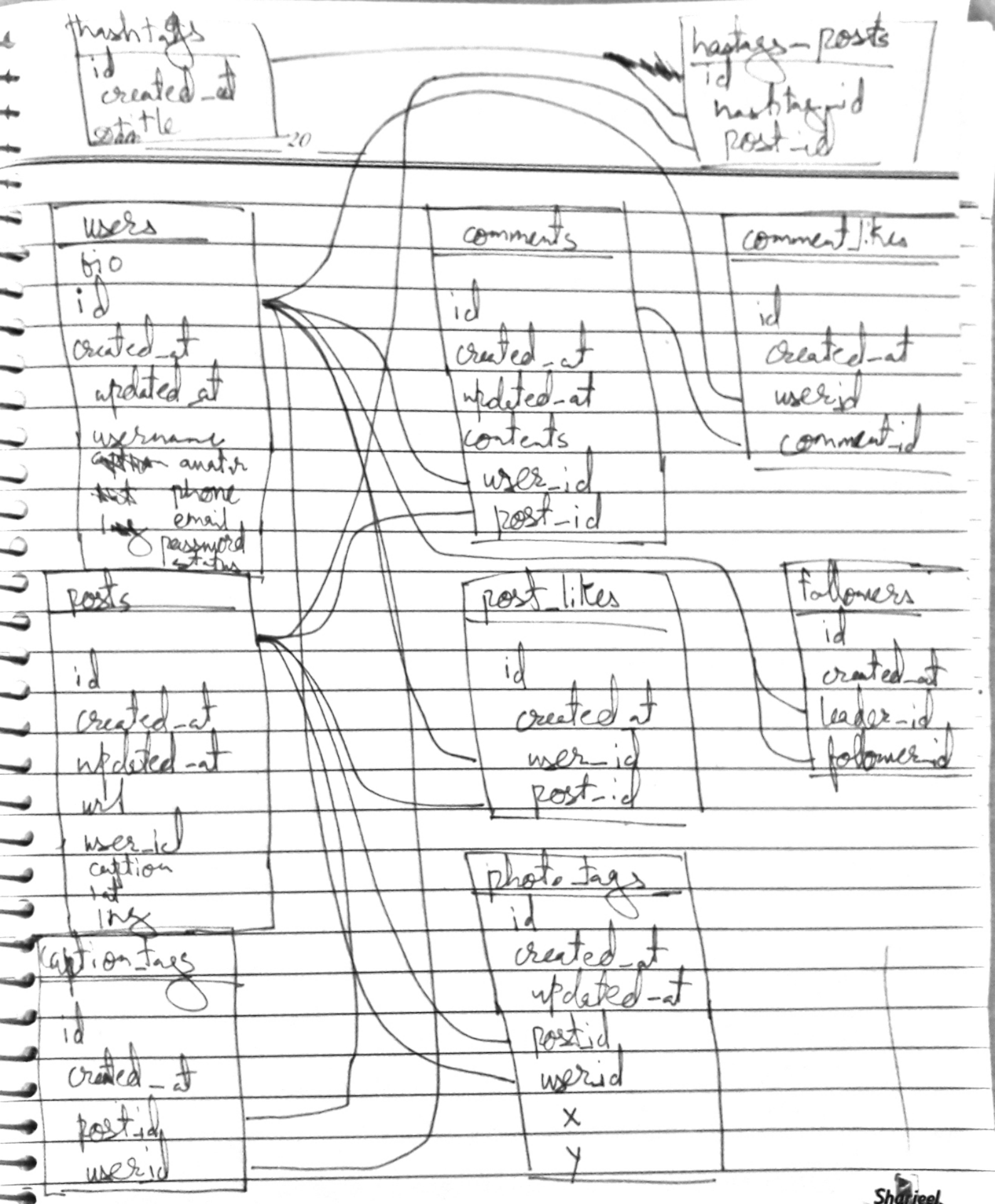
accurate number → numeric

double precision not as accurate

Row level Validation

NOT NULL
UNIQUE

CHECK


hashtags
id
created_at
title
hashtags_posts
id
hashtag_id
post_id
users
bio
id
created_at
updated_at
username
avatar
phone
email
password
comments
id
created_at
updated_at
contents
user_id
post_id
comment_likes
id
created_at
user_id
comment_id
posts
id
created_at
updated_at
url
user_id
caption
lat
lng
post_likes
id
created_at
user_id
post_id
followers
id
created_at
leader_id
follower_id
caption_tags
id
created_at
post_id
user_id
photo_tags
id
created_at
updated_at
post_id
user_id
x
y

Date _____ 20 ____

Likes

1 user 1 like a post
unlike a post
how many users like a post?
list which users like a post
other resources may be liked comments etc.
dislikes

Polymorphic association (not recommend)

id column & id type column
an id w/ a type column to point to table
does not check if id exists as ~~set~~ foreign key

loose out on data consistency

Date _____ 20 _____

generally don't want to store derived data.

Internals of PostgreSQL

SHOW data_directory;

/var/lib/postgresql/14/main/base

↓
database saved by oid

Select oid, datname From pg_database;

base/{oid num}

↑
lots of files

select * from pg_class;

↓

individual object table
↳ columns, constraint,

Sharjeel

Date _____ 20 ____

Heap / Heap file

↓

all data (rows) of table

Tuple / Item

↓

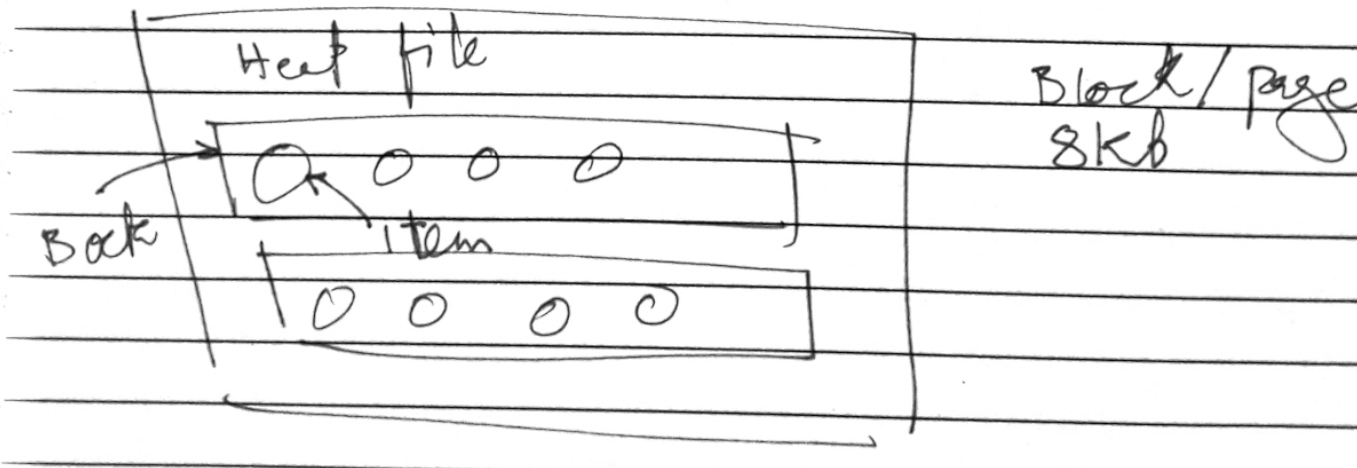
Individual row from table

Block / page

↓

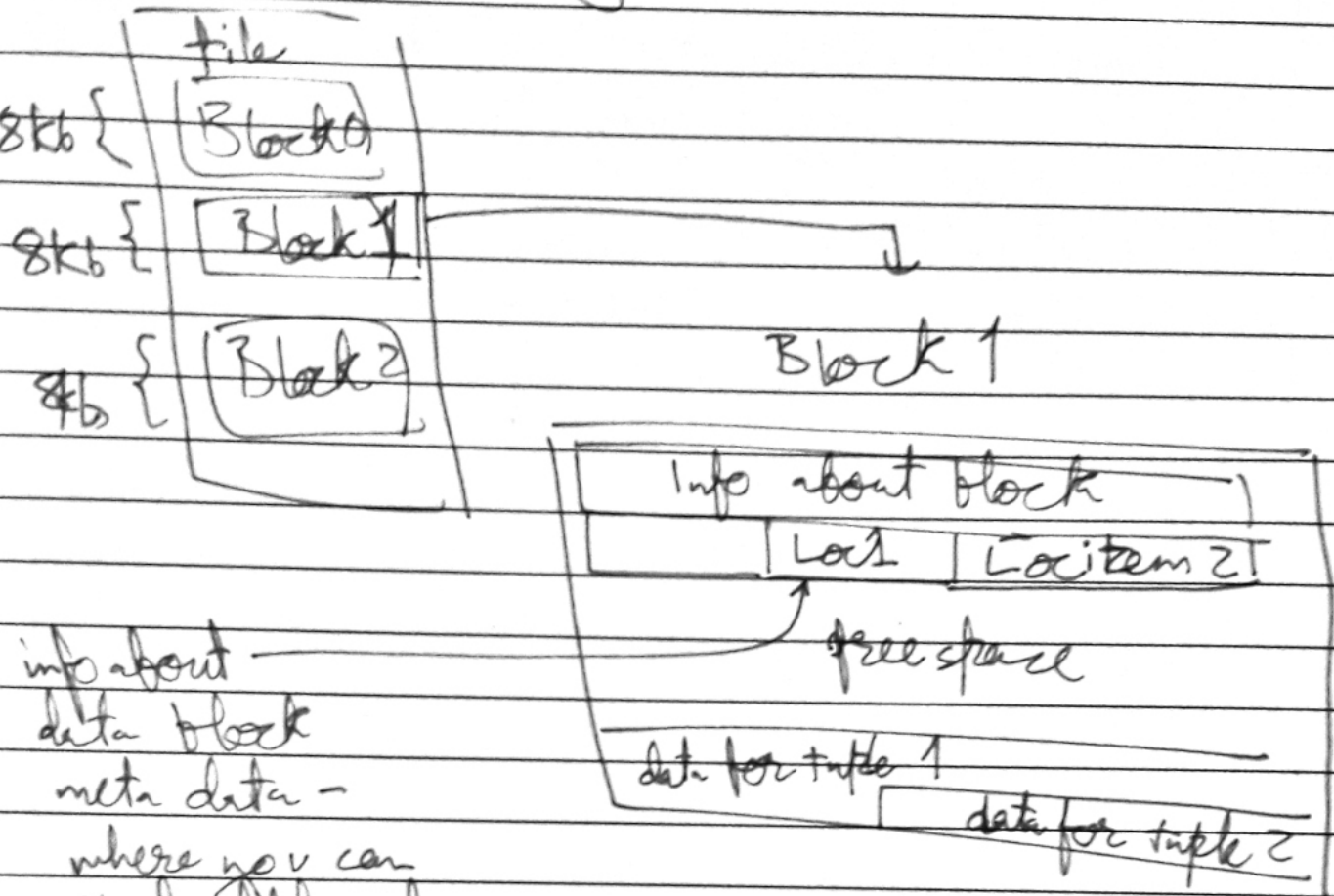
Heap file ÷ into many pages

each page ~~has~~ stores some number of rows



Date _____ 20 ____

Block data layout



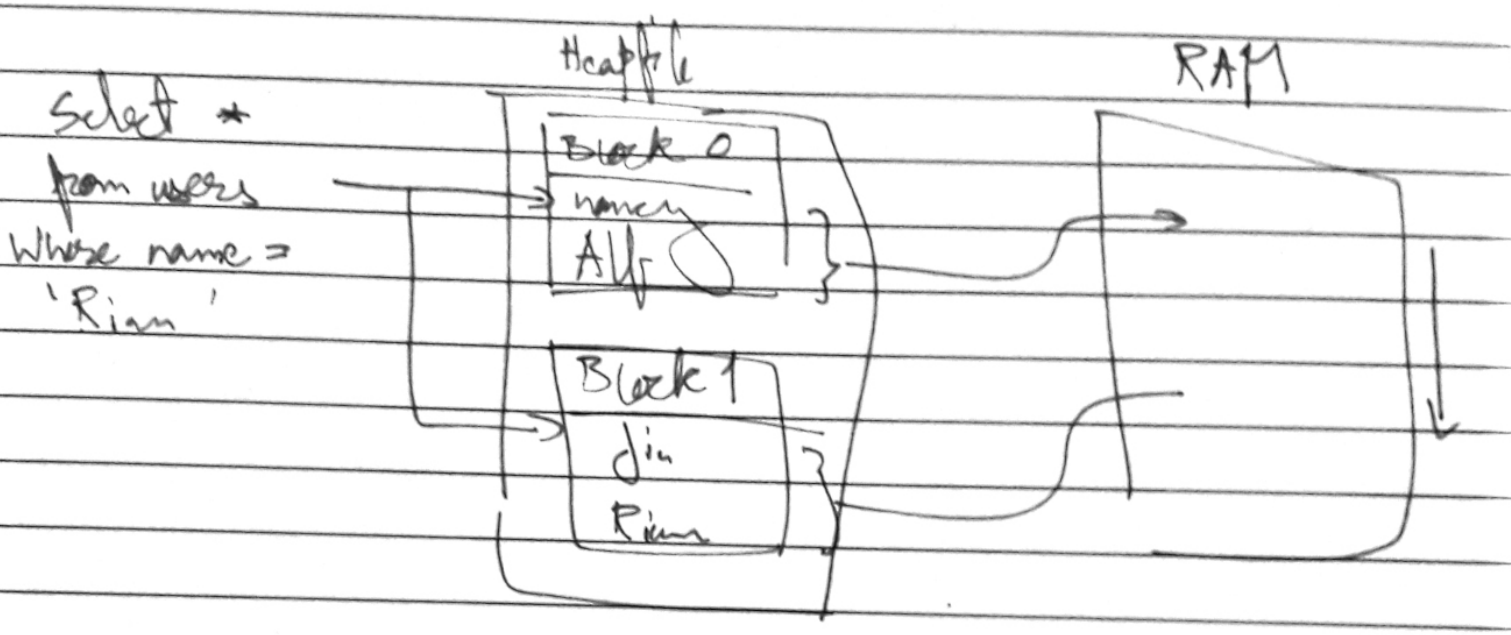
Date _____ 20 ____

Overall Page layout

- page header data → 24 bytes long info, free space ptrs
- Item Id data → array of ptrs to items
- free space → new ids allocated from start, new items end
- Items → the actual items
- special space → index access method specific data
  different methods store different data.
  empty in ordinary tables

Date _____ 20 ____

Indexes



load data from blocks into mem
look at each item 1 by 1

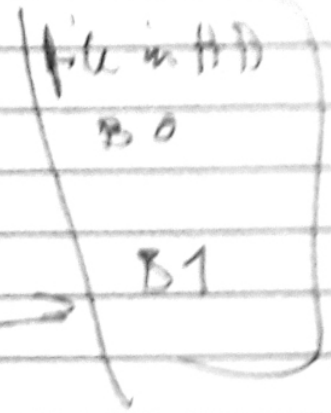
Full table scan

PG has to load many or all rows from the heap file to memory

Poor performance

Date _____ 20 ____

select → Index



Index

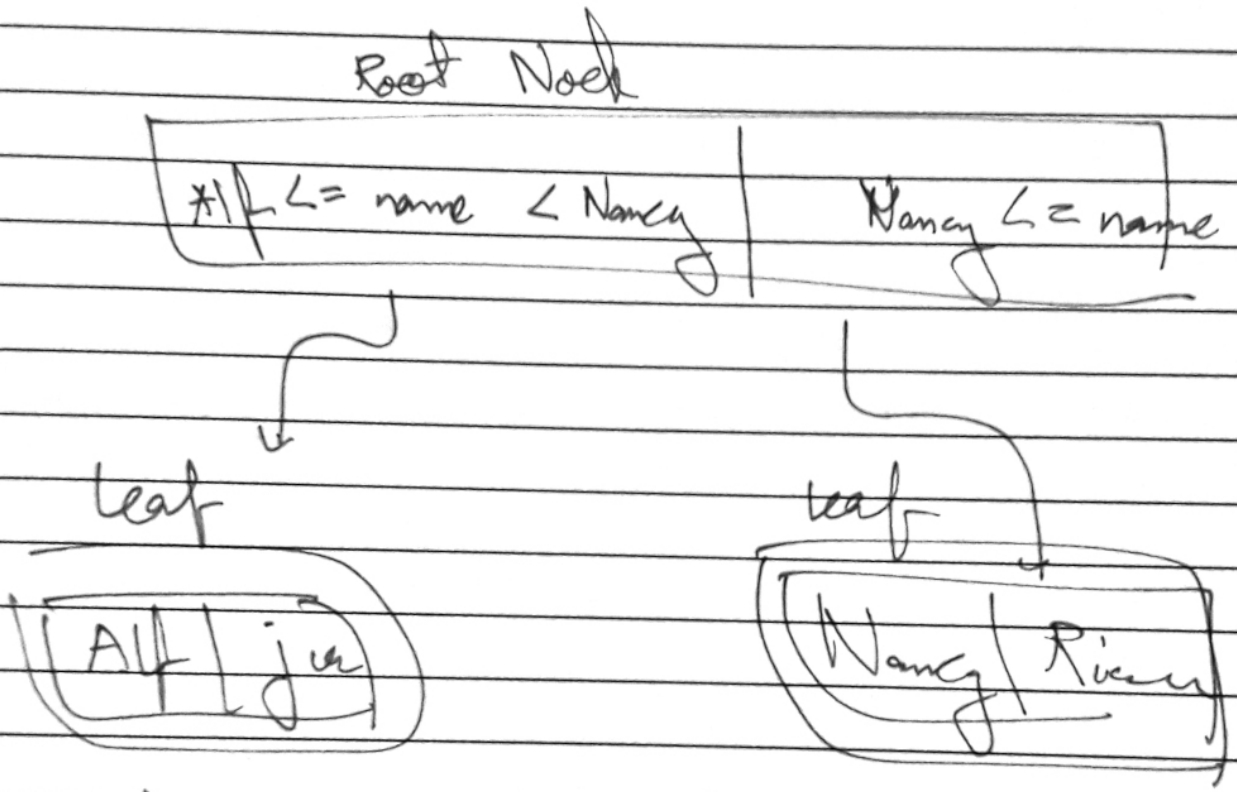
data structure that efficient tells us what block / index a record is stored at

How an index works?

- Index on column(s)
  - fast lookup on column
  - extract only property we want to do fast lookups by & the block / index for each
  - sort in some meaningful way

Date _____ 20 ____

organize into a tree data structure



skip huge portion of search

Date _____ 20 ____

Creating an index

```
CREAT INDEX ON users (username);
```

downsides of Indexes

- can be large, stores data from at least one column of real table
- slows down insert / update / delete index has to be updated
- index might not actually get used

Types of indexes

- B tree 99% of the time mean this

Date _____ 20 ____

Hash → speeds up simple equality checks

GiST → Geometry full text search

SP-GiST → clustered data such as dates many rows might have same year

GIN → arrays or JSON data

BRIN → specialized for really large datasets

---

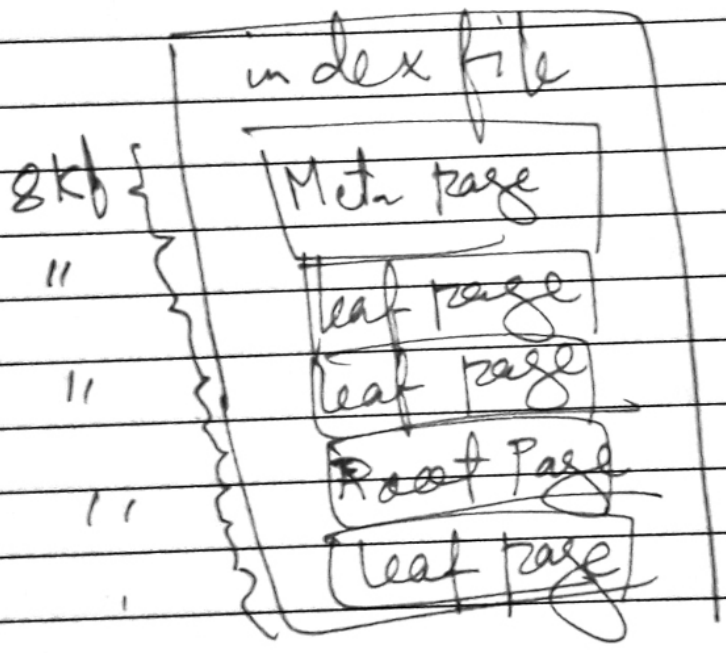
*tomatically generated indexes

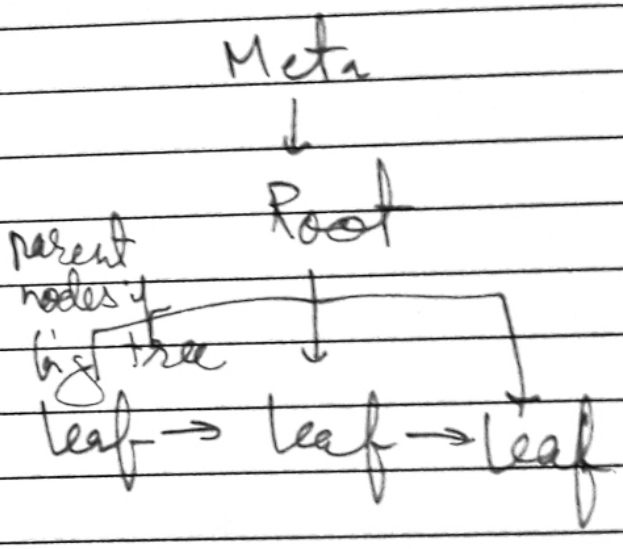
on Primary Key, Unique constraints

Date _____ 20 ____

Indexes behind the scenes

On disk index file





leaf node 1st entry pointer to next leaf node * 1st record

Date _____ 20 ____

Query tuning

Query

↓

parser → check if valid sequel outputs query tree

↓

Rewriter → decompose views into underlying table

↓

Planner → figure out most efficient way to do something

↓

Executer

Date _____ 20 ____

EXPLAIN → build a query plan & display info about

EXPLAIN ANALYZE → build a query plan run it & info about it

Query PLAN
↓
Query Node → some step, accessing data or doing some processing

Index scan
↓
Hash

Seq Scan
↓

Hash Join
↓
res

Date _____ 20 ____

Hash Join

cost → amount of processing power required for this step

rows → estimate of how many rows step will produce

width → guess at avg # of bytes of each row

cost
= amount of time to execute one part of our query plan

How can planner decide most efficient strategy w/o executing query?

Date _____ 20 ____

loading data from random spots off a HDD usually takes more than loading data sequentially.

Sequential load faster than random load

cost 2 values = startup cost .. overall cost

1 for this step (row)
1 for this step to produce all steps

as soon as row process emited

Date ____ 20 ____

Common table expressions

```
WITH name AS (
        subquery
)
Select main query
```

produces table we can use anywhere else

Recursive common table expressions

- very different from simple CTE
- useful anytime using tree/graph data structure
- must use union keyword
- very advanced

Date _____ 20 ____

~~steps~~

table
columns
With Recursive countdown (_____) AS (

initial query
UNION
recursive query
)
select * from countdown;

- Define results & working table (temp tables)
- run the initial statement puts res in results & working table
- run the recursive statement replace final table name w/ ref to working table
- if recursive statement returns rows append them to results table & run recursion again
- if recursion returns no rows stop
- rename results table to final table

Date _____ 20 ____

Views

Like CTE but not attached to a query

virtual table

Views

↳ query gets executed every time referred to.

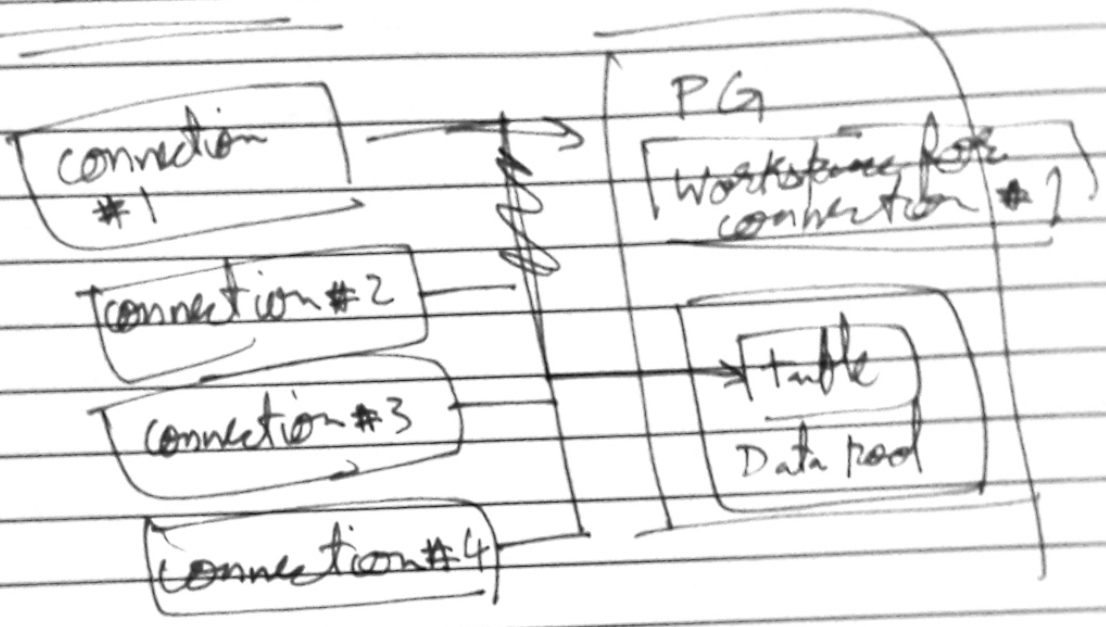
Materialized views

gets executed only at very specific times
but results are saved & can be referenced
w/o rerunning the query.

use when doing
a very expensive queries

Date _____ 20 ____

Transactions



Date _____ 20 ____

Schema Migrations

↓

making changes to your DB structure

- Changes to DB structure & changes to clients need to be made at precisely the same time
- When working w/ other engineers we really need a easy way to tie the structure of our database to our code

Schema migration file

code that describes a change

UP	DOWN
↓	↓
upgrade	undo UP
Apply	Revert

Date _____ 20 ___

Issues solved by migration files

- run all migrations in CI
- add to code review ⇒ migration file

Data Migrations

Add column → schema migration
copy / move data b/w columns } → data migrations

Run migrations in transactions

Split data & schema migrations

Date _____ 20 ____

Ex = lat & lng column add new loc column of type point delete lat & lng

1, add column loc (migration)

↓

2, deploy new version of API write values to both lat/lng & loc

↓

3, copy lat/lng to loc (not necessarily migration)

↓

4, update code to only write to loc column

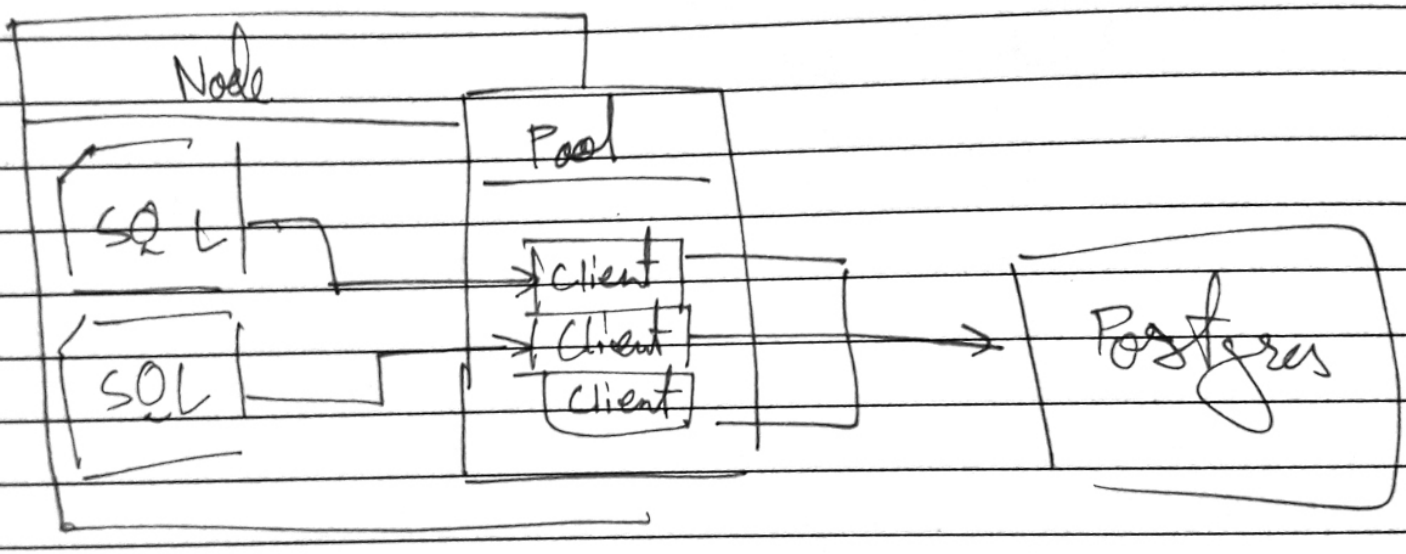
↓

5, drop columns lat/lng (migration)

Date _____ 20 ____

Connection pools

Client can only be used for one query at a time



pool maintains several different clients that can be reused

Client only used to run transactions

Date _____ 20 __

one central point for accessing kind of resource

Repository Pattern can be plain object, instance of class, static class

Functions

find → /resource GET

findById → /resource/:id GET

insert → /resource POST

update → /resource/:id PUT

delete → /resource/:id DELETE

findOne → filtering criteria

count → # of rows

validate

Date

Prepared Statements

Prepare identifier (_) AS

Query WHERE = $1 ;

Execute identifier

prepared statement can't put in tables or columns only values

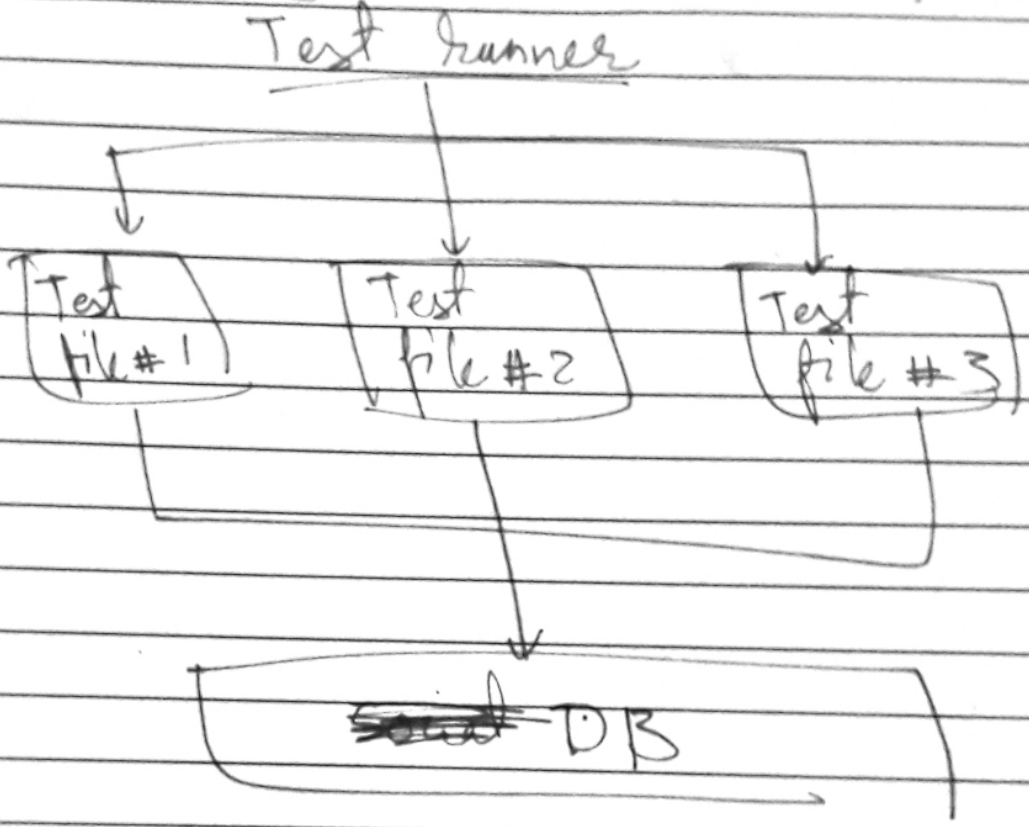
insert/ Query RETURNING * ;

↓

return all columns after insert

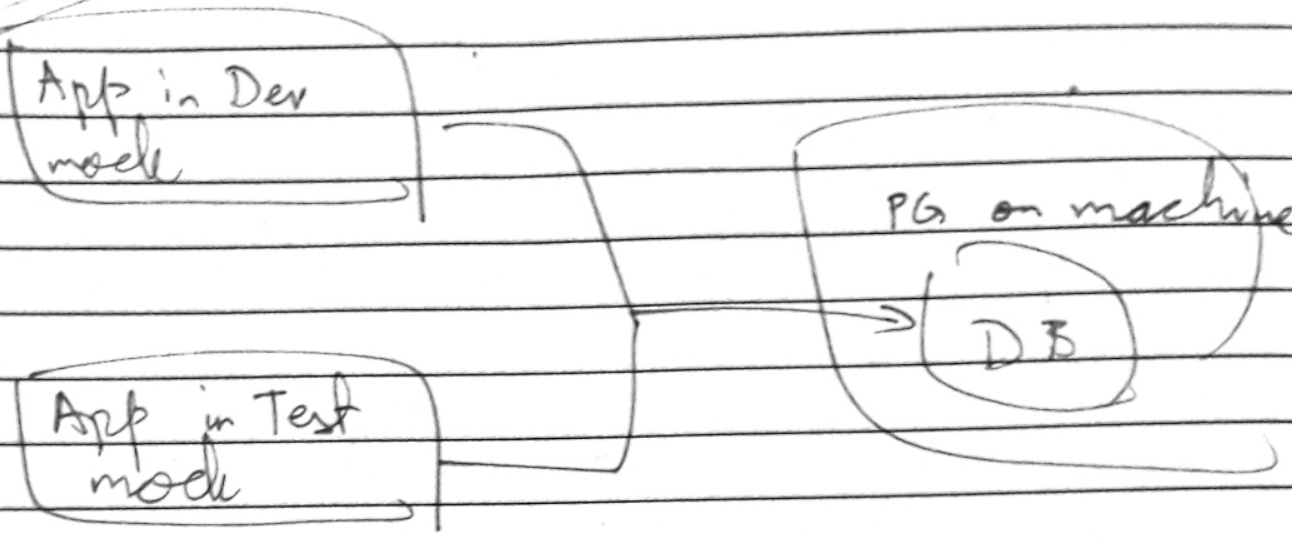
Date 20

Parallel Testing

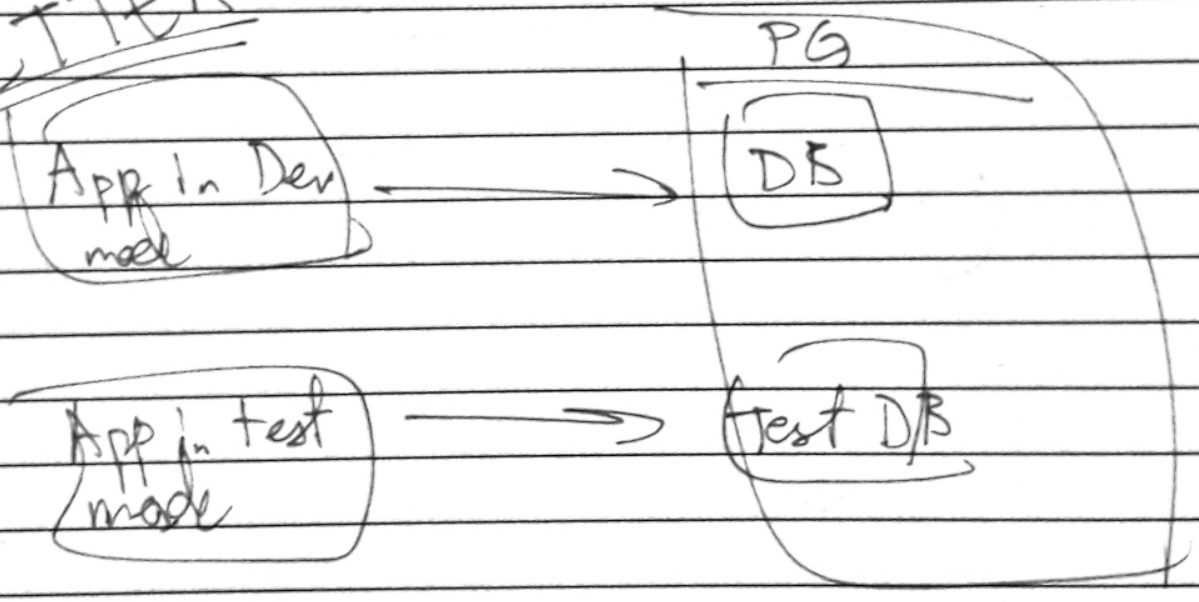


Date 20

WRONG



BETTER



Sharjeel

Date _____ 20 ____

Isolation

Isolation required to run tests in parallel

Isolation w/ schemas

each test file gets its own schema

- like a folder
- default schema is 'public'
- each schema has its own copy of tables

CREATE SCHEMA schema_name;

→ try schema w/ name same as user

search_path = "$user", public

↳ which schema PG picks first

Date _____ 20 ____

For Each Test file

1, connect to PG as normal

2, generate a random string of chars

3, create a new user (role) w/ that name

4, create a new schema w/ that name

5, Tell our test file to connect to the DB w/ that name