course 1 introduction

course goals:

- learn architecture of relatinoal database(like postgre sql)
- familiar with algorithm/data-structure about data-intensive computing
- implementation of relational operators (like sel, proj, join)
- familiar with relational database objects…..for what, i don’t know
- learning technique for managing concurrent trasactions
- learning concepts in distributed and no-sql database

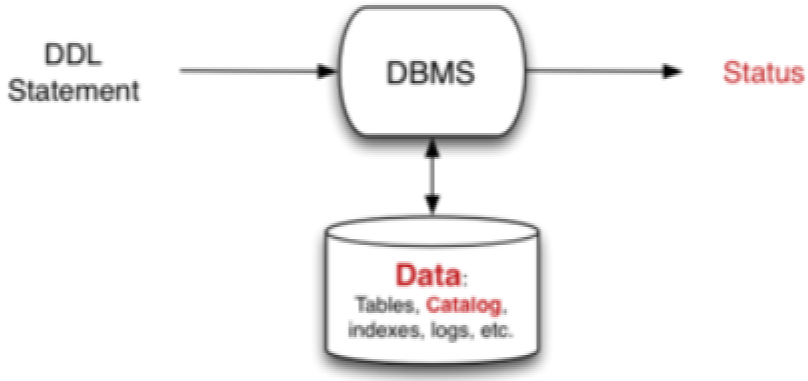
functions of DBMS:

- extensibilities via views, triggers and procedures
- defining relational data like (relations/tables, tuples, values, types, constraints)
- etc

Data Defination:

- how to define, examples underneath

```
create domain WAMvalue float
  check (value between 0.0 and 100.0);
create table Students (
  id integer, -- e.g. 3123456
  familyName text, -- e.g. 'Smith'
  givenName text, -- e.g. 'John'
  birthDate date, -- e.g. '1-Mar-1984'
  wam WAMvalue, -- e.g. 85.4
  primary key (id)
);
```



if you input DDL(data description languages) into DBMS, the meta-data in catalog will be modified.

- Constraints

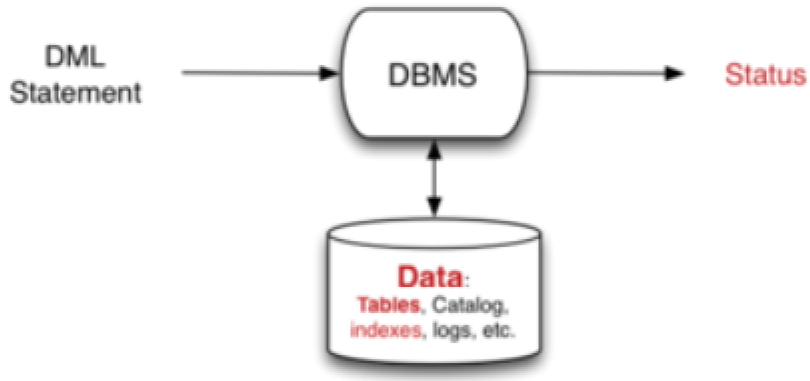
constraints aims to specify rules for the data in tables

```
create table Employee (
   id      integer primary key,
   name    varchar(40),
   salary  real,
   age     integer check (age > 15),
   worksIn integer
              references Department(id),
   constraint PayOk check (salary > age*1000)
);
```

Data Modification:

manipulating data is an important function of DBMS, like:

- insert new tuples into tables
- delete existing tuples from tables
- updating values within existing tuples



tip: most DBMS also provide **bulk** download/upload too

DBMS architecture:

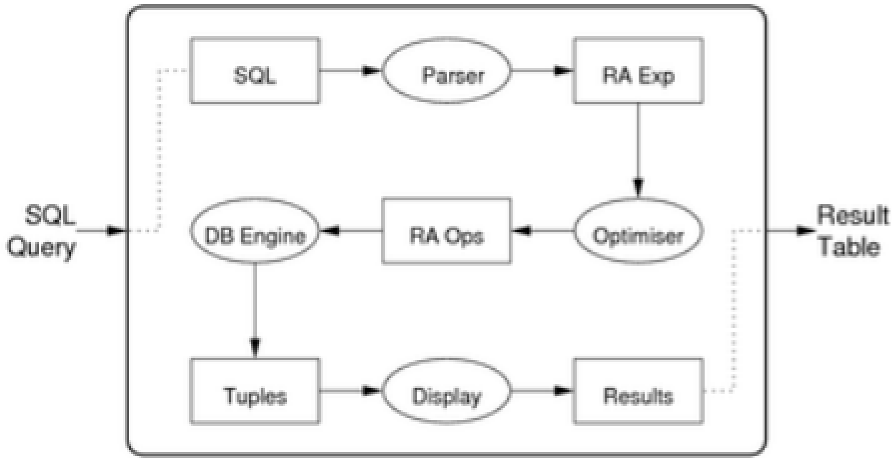
fundamental tenets of DBMS architecture:

- data is stored permanently in large but slow device
- data is processed in fast but small memory

which means that:

- data structure should minimize storage utilisation
- algorithm should minimize the mem/disk transfer

path of a query through a typical DBMS:



There are several DBMS architectures, but **there are some section they must contains:**

- query optimiser
- query executor
- access methods
- buffer manager
- storage manager
- concurrency manager
- recovery manager
- integrity manager: *verifies integrity constraints and user privileges*

DB engine operations:

there are several implementation of selection, for examples:

- a hash-structured file is good for queries like:

```
select * from Students where id = 3312345;
```

- a B-tree file is good for queries like:

```
select * from Employees where age > 55;
```

Relational Algerba

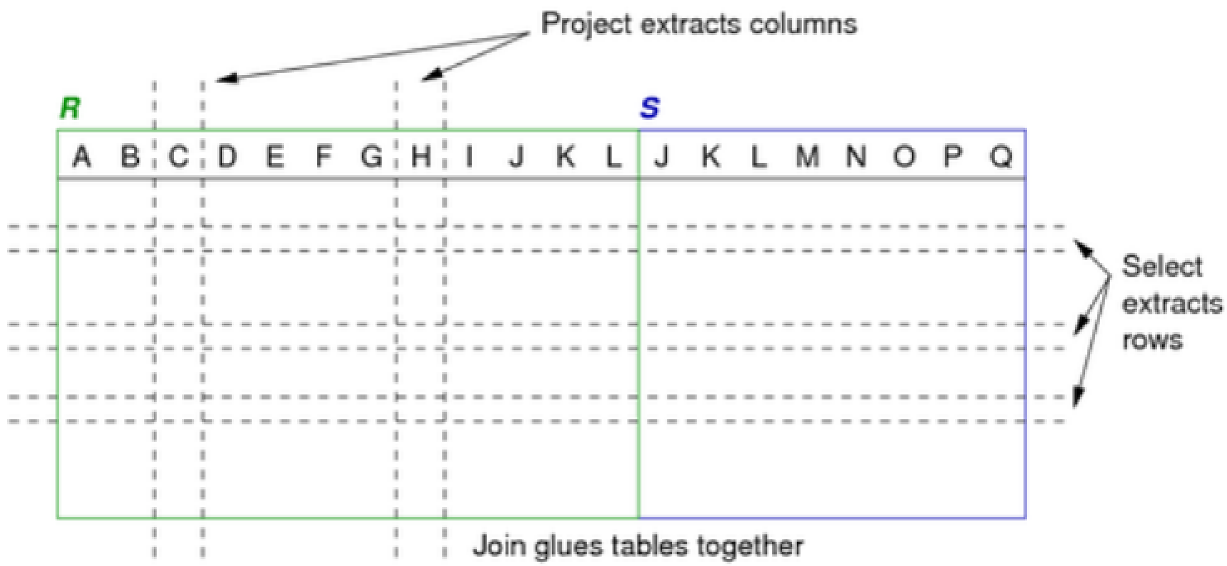
DB engine = relational algebra virtual machine

selection (σ)	projection (π)	join ($\bowtie$)
union ($\cup$)	intersection ($\cap$)	difference ($-$)
sort	group	aggregate

RA consists of:

- operands : relations, or variables representing relations
- operators : **map relations to relations**
- rules : **combining operands/operators into expressions**

illustrating how RA manipulate tables:



manipulation

examples database to demostrate RA opeators is shown below

R

A	B	C
1	a	2
2	b	2
3	c	3
4	a	3
5	b	4

S

C	D
2	x
3	y
5	z

examples database to demostrate RA operators is shown below

Account

branchName	accountNo	balance
Downtown	A–101	500
Mianus	A–215	700
Perryridge	A–102	400
Round Hill	A–305	350
Brighton	A–201	900
Redwood	A–222	700

Branch

branchName	address	assets
Downtown	Brooklyn	9000000
Redwood	Palo Alto	2100000
Perryridge	Horseneck	1700000
Mianus	Horseneck	400000
Round Hill	Horseneck	8000000
North Town	Rye	3700000
Brighton	Brooklyn	7100000

Customer

name	address	customerNo	homeBranch
Smith	Rye	1234567	Mianus
Jones	Palo Alto	9876543	Redwood
Smith	Brooklyn	1313131	Downtown
Curry	Rye	1111111	Mianus

Depositor

account	customer
A–101	1313131
A–215	1111111
A–102	1313131
A–305	1234567
A–201	9876543
A–222	1111111
A–102	1234567

selection

Sel [B=a] R

A	B	C
1	a	2
4	a	3

Sel [C>2] S

C	D
3	y
5	z

Sel [A>=C] R

A	B	C
2	b	2
3	c	3
5	b	4

Sel [C=2 ll D=y] S

C	D
2	x
3	y

projection

R

A	B	C
1	a	2
2	b	2
3	c	3
4	a	3
5	b	4

Proj [B] R

B
a
b
c

Proj [A,C] R

A	C
1	2
2	2
3	3
4	3
5	4

Union

Union combines two compatible relatinos into a single relation via set union of sets of tuples

$$r_1 \cup r_2 \;=\; \{\, t \mid t \in r_1 \vee t \in r_2 \,\}, \quad \text{where } r_1(R),\ r_2(R)$$

union examples:

- which suburbs have either customers or branches?
  - Proj[address](Customers) ∪ Proj[address](Branch)
- which branches have either customers or accounts?
  - Proj[homeBranch](customers) ∪ Proj[branchNames](accounts)

intersection

intersection combines two compatible relations into a single relation via set intersection of sets of tuples

$$r_1 \cap r_2 \;=\; \{\, t \mid t \in r_1 \wedge t \in r_2 \,\}, \quad \text{where } r_1(R),\ r_2(R)$$

intersection examples:

- which suburbs have both customers or branches?
    - Proj[address](Customers) ∩ Proj[address](Branch)
- which branches have both customers or accounts?
    - Proj[homeBranch](customers) ∩ Proj[branchNames](accounts)

difference

difference finds the set of tuples that exist in one relation but do not occur in a second compatible relation

$$r_1 - r_2 \ = \ \{ \ t \ | \ t \in r_1 \wedge \neg \ t \in r_2 \}, \quad \text{where } r_1(R), r_2(R)$$

difference examples:

Sel [B=a] R

s1

A	B	C
1	a	2
4	a	3

Sel [C=2] R

s2

A	B	C
1	a	2
2	b	2

s1-s2

A	B	C
4	a	3

s2-s1

A	B	C
2	b	2

Natural Join

natural join is special that:

- containning only pairs that match on their both attributes
- with one of each pair of common attributes eliminiated(in other word, only keep one attributes if there are two same attributes)

natural join examples:

R Join S

A	B	C	D
1	a	2	x
2	b	2	x
3	c	3	y
4	a	3	y

Theta Join

theta join is a special product containing only pairs that match on supplied condition C

examples:

R Join [R.A>S.C] S

A	B	R.C	S.C	D
3	c	3	2	x
4	a	3	2	x
4	a	3	3	y
5	b	4	2	x
5	b	4	3	y

Outer Join

R join S eliminates all S tuples that do not match some R tuples.

R LeftOuterJoin [R.A>S.C] S

A	B	R.C	S.C	D
1	a	2	null	null
2	b	2	null	null
3	c	3	2	x
4	a	3	2	x
4	a	3	3	y
5	b	4	2	x
5	b	4	3	y

Aggregation

there are two types of aggregation are common in database queries:

- accumulating summary values for data in tables:
    - typical operation like: Sum, Average, Count
- grouping set of tuples with common values:

tips: both two typical operation usually **work on single column**

Generalised Projection

in generalised projection, we perform some computation on the attribute value before placing it in the result tuples.

examples:

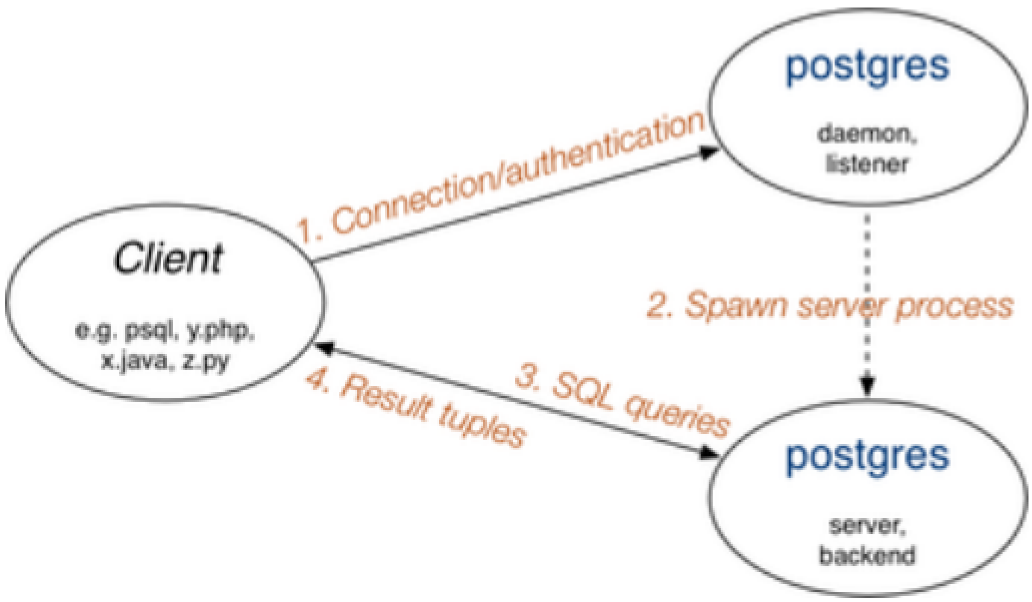
- Display branch assets in Aus$ rather than US$.
    - *Proj [branchname,address,assets*0.75] (Branch)*
- Display employee records using age rather than birthday.
    - *Proj [id,name,(today–birthdate)/365,salary] (Employee)*

PostgreSQL

PostgreSQL architecture

client/server architecture

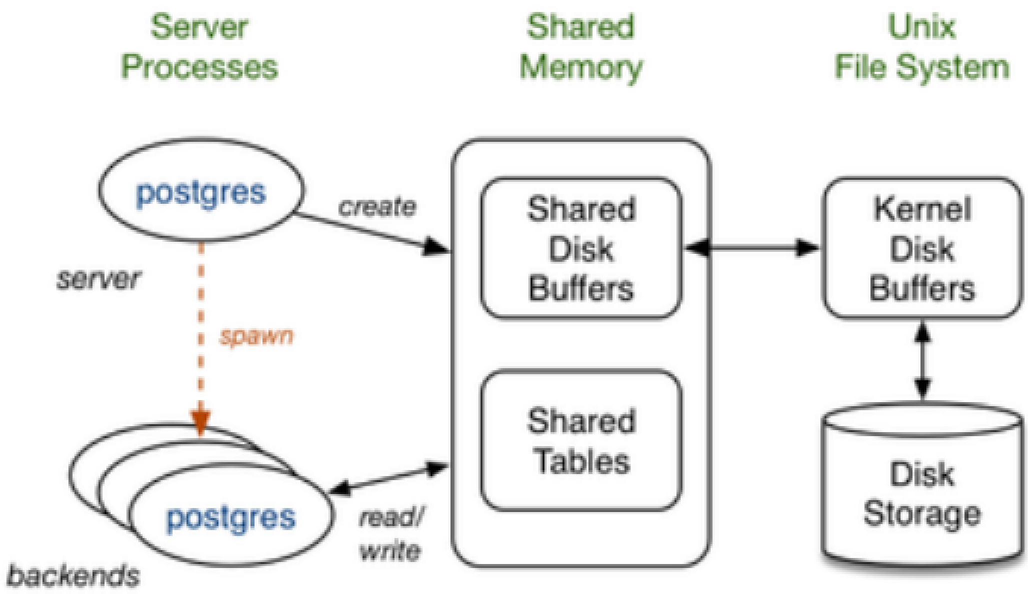
- client: like psql, y.php, x.java, z.py
- postgre: including daemon, listener
- postgres (postmaster): including server, backend



supplementary instructions:

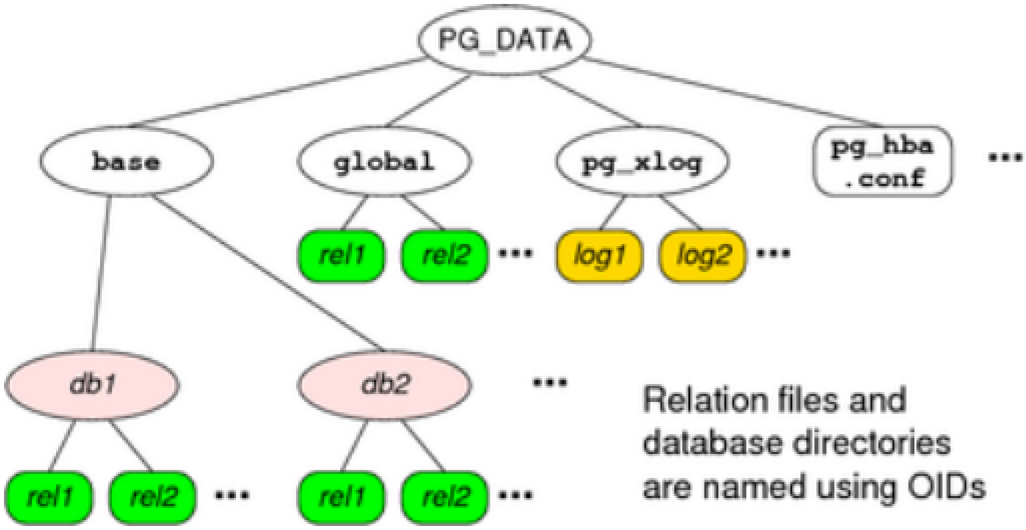
- exactly one postmaster, many client, many server
- exactly client has its own server process
- client/server communication via TCP/IP or Unix socket

Memory/storage architecture



- all servers access database via buffer pool
    - thus, all servers can get consistent view of data, which is essential
- use of shared memory limits distribution/scalability
    - all server process have to run on the same machine
- shared tables are “global” system catalog tables

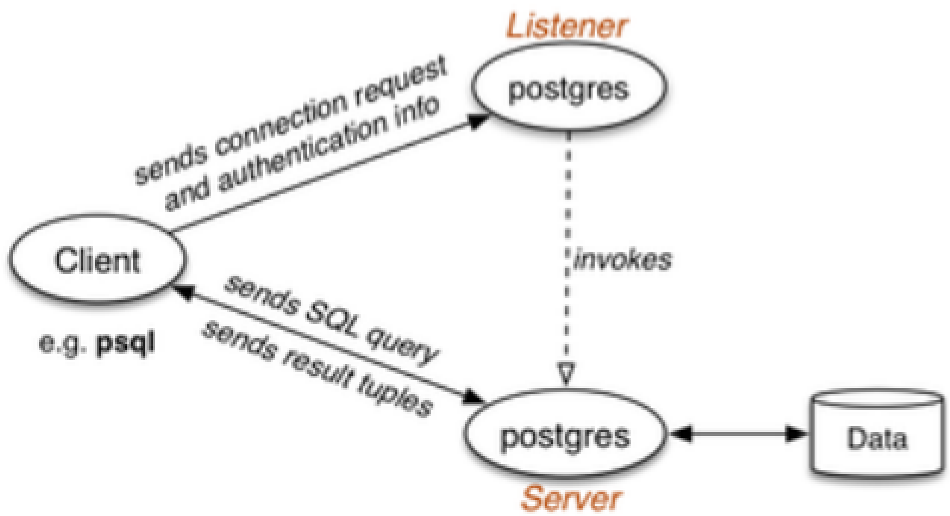
File-system architecture



there are severl interesting file in PG_DATA:

- PG_VERSION
- pg_hba.conf
- postgresql.conf
- postmaster.opts
- postmaster.pid

Life-cycle of a POstgreSQL query



how a postgreSQL query is executed:

- SQL string is produced in Client
- client established a connnection with postgreSQL
- dedicated server process attached to client
- SQL string sent to server process
- server pareses/plan/optimises query
- server executes query to produce result tuples
- tuples are transmitted back to client
- client disconnects from server

PostgreSQL Data Types

there are two most important data types in postgresql

- Node provides generic structure for nodes
  - Node types: parese trees, plan trees, execution trees, ….
- List provides generic singly-linked list

PostgreSQL query evaluation

Each query is represented by a Query structure, which holds all components of the SQL query, including:

- required columns as list of TargetEntrys
- referenced tables as list of RangeTblEntrys
- where clause as node in FromExpr struct
- sorting requirements as list of SortGroupClauses