Week 01 Lecture

COMP9315 DBMS Implementation

(Data structures and algorithms inside relational DBMSs)

Lecturer: ***John Shepherd***

Web Site: http://www.cse.unsw.edu.au/~cs9315/

(If WebCMS unavailable, use http://www.cse.unsw.edu.au/~cs9315/16s1/)

Lecturer

2/71

Name:	John Shepherd
Office:	K17-410 (turn right from lift)
Phone:	9385 6494
Email:	jas@cse.unsw.edu.au
Consult:	Tue 3-4, Wed 2-3
Research:	Information Extraction/Integration Information Retrieval/Web Search e-Learning Technologies Multimedia Databases Query Processing

Course Goals

3/71

Introduce you to:

- architecture(s) of relational DBMSs (via PostgreSQL)
- algorithms/data-structures for data-intensive computing
- representation of relational database objects
- representation of relational operators (sel,proj,join)

- techniques for processing SQL queries
- techniques for managing concurrent transactions
- concepts in distributed and non-relational databases

Develop skills in:

- analysing the performance of data-intensive algorithms
- the use of C to implement data-intensive algorithms

Learning/Teaching

Stuff that's available for you:

- Textbooks: describe some syllabus topics in detail
- Notes: describe all syllabus topics in some detail
- Lecture slides: summarise Notes and contain exercises
- Lecture videos: for review or if you miss a lecture, or are in WEB stream
- Readings: research papers on selected topics

The onus is on *you* to use this material.

Note: Lecture slides, exercises and videos will be available only *after* the lecture.

Note: I will be away from March 2-4; Week 4 lecture is from 4-6.

... Learning/Teaching

Things that you need to **do**:

- Exercises: tutorial-like questions
- Prac work: lab-class-like exercises
- Assignments: large/important practical exercises
- On-line quizzes: for self-assessment

Dependencies:

- Exercises → Exam (theory part)
- Prac work → Assignments → Exam (prac part)

There are **no** tute/lab classes; use Forum, Email, Consultations

- debugging is best done in person (where full environment is visible)

Pre-requisites

We assume that you are already familiar with

- the C language and programming in C (or C++)
  (e.g. completed an intro programming course in C)
- developing applications on RDBMSs
  (SQL, [relational algebra] e.g. an intro DB course)
- basic ideas about file organisation and file manipulation
  (Unix `open, close, lseek, read, write, flock`)
- sorting algorithms, data structures for searching
  (sorting, trees, hashing e.g. a data structures course)

If you don't know this material, you will struggle to pass ...

Exercise 1: SQL (revision)

Given the following schema:

```
Students(sid, name, degree, ...)
e.g. Students(3322111, 'John Smith', 'MEngSc', ...)
Courses(cid, code, term, title, ...)
e.g. Courses(1732, 'COMP9311', '12s1', 'Databases', ...)
Enrolments(sid, cid, mark, grade)
e.g. Enrolments(3322111, 1732, 50, 'PS')
```

Write an SQL query to solve the problem

- find all students who passed COMP9315 in 16s1
- for each student, give (student ID, name, mark)

Exercise 2: Unix File I/O (revision)

Write a C program that reads a file, block-by-block.

Command-line parameters:

- block size in bytes
- name of input file

Use low-level C operations: `open, read`.

Count and display how many blocks/bytes read.

Prac Work

In this course, we use PostgreSQL v9.4.6 (compulsory)

Prac Work requires you to compile PostgreSQL from source code

- instructions explain how to do this on Linux at CSE
- also works easily on Linux and Mac OSX at home
- PostgreSQL docs describe how to compile for Windows

Make sure you do the first Prac Exercise when it becomes available.

Sort out any problems ASAP (preferably at a consultation).

... Prac Work

PostgreSQL is a *large* software system:

- > 1000 source code files in the core engine/clients
- > 500,000 lines of C code in the core

You won't be required to understand all of it :-)

You will need to learn to navigate this code effectively.

Will discuss relevant parts in lectures to help with this.

PostgreSQL books?

- tend to add little to the manual, and cost a lot

Assignments

Schedule of assignment work:

Ass	Description	Due	Marks
1	Storage Management	Week 5	11%
2	Query Processing	Week 11	14%

Assignments will be carried out in pairs (see WebCMS).

Choose own online tools to share code (e.g. git, DropBox).

Ultimately, submission is via CSE's `give` system.

Will spend some time in lectures reviewing assignments.

Assignments will require up-front code-reading (see Pracs).

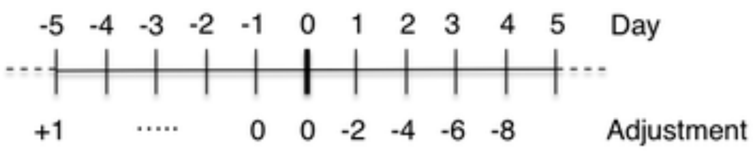
... Assignments

Don't leave assignments to the last minute

- they require significant code reading
- as well as code writing and testing
- and, you *can* submit early.

"Carrot": bonus marks are available for early submissions.

"Stick": marks deducted (from max) for late submissions.



Quizzes

Over the course of the semester ...

- six online quizzes
- taken in your own time (but there are deadlines)
- each quiz is worth a small number of marks

Quizzes are primarily a review tool to check progress.

But they contribute 10% of your overall mark for the course.

Exam

Three-hour exam in the June exam period.

Held in the CSE Labs, but mainly a written (typed) Exam.

The Course Notes (only) will be available in the exam.

Things that we can't reasonably test in the exam:

- writing *large* programs, running *major* experiments, drawing diagrams

Everything else is potentially examinable.

Contains: descriptive questions, analysis, small programming exercises.

Exam contributes 65% of the overall mark for this course.

Supp Exams: you get *one chance* at passing the exam.

Assessment Summary

Your final mark/grade is computed according to the following:

```
ass1   = mark for assignment 1      (out of 11)
ass2   = mark for assignment 2      (out of 14)
quiz   = mark for on-line quizzes   (out of 10)
exam   = mark for final exam        (out of 65)
okExam = exam > 26/65           (after scaling)

mark   = ass1 + ass2 + quiz + exam
grade  = HD|DN|CR|PS,  if mark ≥ 50 && okExam
       = FL,           if mark < 50 && okExam
       = UF,           if !okExam
```

Relational Database Revision

Relational DBMS Functionality

Relational DBMSs provide a variety of functionalities:

- storing/modifying *data* and *meta-data* (data defintions)
- *constraint* definition/storage/maintenance/checking
- declarative manipulation of data (via *SQL*)
- extensibility via *views, triggers, procedures*
- query re-writing (*rules*), optimisation (*indexes*)
- *transaction* processing, concurrency/recovery
- etc. etc. etc.

Common feature of all relational DBMSs: relational model, SQL.

Data Definition

Relational data: relations/tables, tuples, values, types, e.g.

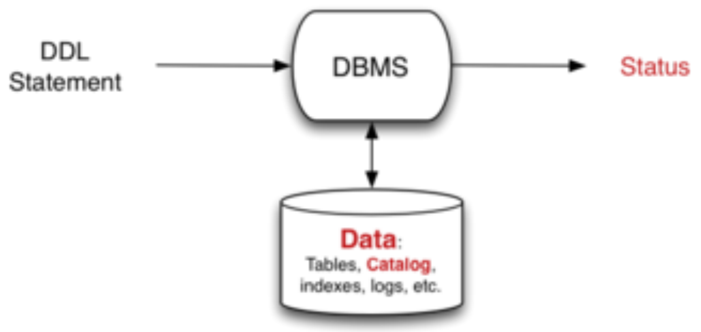
```
create domain WAMvalue float
   check (value between 0.0 and 100.0);

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)
);
```

The above adds *meta-data* to the database.

DBMSs typically store meta-data as special tables (catalog).

... Data Definition

Input: DDL statement (e.g. `create table`)



Result: meta-data in catalog is modified

... Data Definition

Constraints are an important aspect of data definition:

- attribute (column) constraints
- tuple constraints
- relation (table) constraints
- referential integrity constraints

Examples:

```
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)
);
```

On each attempt to change data, DBMS checks constraints.

Data Modification

Critical function of DBMS: changing data

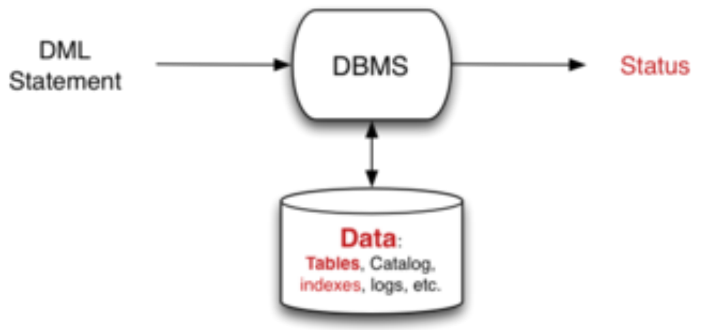
- `insert` new tuples into tables
- `delete` existing tuples from tables
- `update` values within existing tuples

E.g.

```
insert into Enrolments(student,course,mark)
values (3312345, 5542, 75);

update Enrolments set mark = 77
where  student = 3354321 and course = 5542;

delete Enrolments where student = 331122333;
```

... Data Modification

Input: DML statements



Result: tuples are added, removed or modified

Query Evaluator

Most common function of relational DBMSs

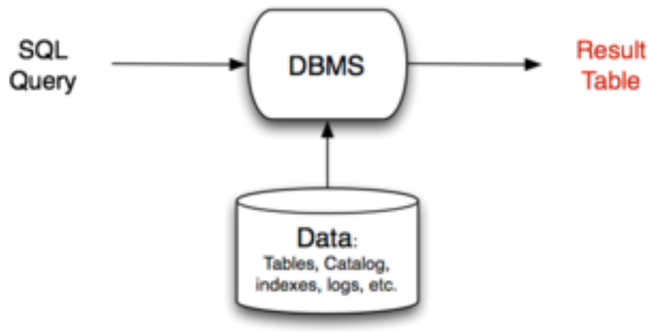
- read an SQL query
- return a table giving result of query

E.g.

```
select s.id, c.code, e.mark
from   Students s, Courses c, Enrolments e
where  s.id = e.student and e.course = c.id;
```

... Query Evaluator

Input: SQL query



Output: table (displayed as text)

DBMS Architecture

25/71

The aim of this course is to

- look inside the DBMS box
- discover the various mechanisms it uses
- understand and analyse their performance

Why should we care? (apart from passing the exam)

Practical reason:

- if we understand how query processor works,
  we can do a better job of writing efficient queries

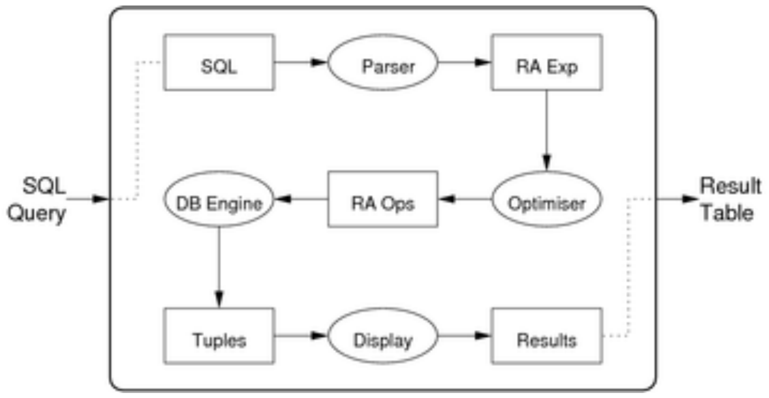
Educational reason:

- DBMSs contain interesting data structures + algorithms
- these may be useful outside the (relational) DBMS context

... DBMS Architecture

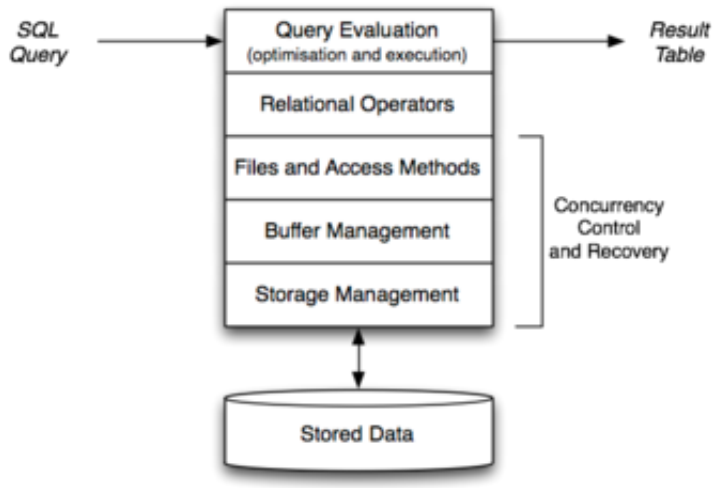
26/71

Path of a query through a typical DBMS:



... DBMS Architecture

27/71



Database Engine Operations

DB engine = "relational algebra virtual machine":

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

For each of these operations:

- various data structures and algorithms are available
- DBMSs may provide only one, or may provide a choice

Relational Algebra

Relational algebra (RA) can be viewed as ...

- mathematical system for manipulating relations, or
- data manipulation language (DML) for the relational model

Core relational algebra operations:

- *selection*: choosing a subset of rows
- *projection*: choosing a subset of columns
- *product*, *join*: combining relations
- *union*, *intersection*, *difference*: combining relations
- *rename*: change names of relations/attributes

Common extensions include:

- *sorting* (**`order by`**), *partition* (**`group by`**), *aggregation*

... Relational Algebra

All RA operators return a result of type *relation*.

For convenience, we can name a result and use it later.

E.g. database *R1(x,y)*, *R2(y,z)*,

```
Tmp1(x,y)   = Sel[x>5]R1
Tmp2(y,z)   = Sel[z=3]R2
Tmp3(x,y,z) = Tmp1 Join Tmp2
Res(x,z)    = Proj[x,z] Tmp3
-- which is equivalent to
Res(x,z)    = Proj[x,z]((Sel[x>5]R1) Join (Sel[z=3]R2))
-- which is equivalent to
Tmp1(x,y,z) = R1 Join R2
Tmp2(x,y,z) = Sel[x>5 & z=3] Tmp1
Res(x,z)    = Proj[x,z]Tmp2
```

Each "intermediate result" has a well-defined schema.

Exercise 3: Relational Algebra

31/71

Using the same student/course/enrolment schema as above:

```
Students(sid, name, degree, ...)
Courses(cid, code, term, title, ...)
Enrolments(sid, cid, mark, grade)
```

Write relational algebra expressions to solve the problem

- find all students who passed COMP9315 in 16s1
- for each student, give (student ID, name, mark)

Describing Relational Algebra Operations

32/71

We define the semantics of RA operations using

- "conditional set" expressions e.g. *{ x | condition }*
- tuple notations:
  - *t[ab]* (extracts attributes *a* and *b* from tuple *t*)
  - *(x,y,z)* (enumerated tuples; specify attribute values)
- quantifiers, set operations, boolean operators

Notation: *r(R)* means relation instance *r* based on schema *R*

Relational Algebra Operations

33/71

Selection

- $\sigma_C(r) \;=\; Sel[C](r) \;=\; \{\, t \mid t \in r \wedge C(t) \,\}$
- *C* is a boolean function that tests selection condition

Computational view:

```
result = {}
for each tuple t in relation r
    if (C(t)) { result = result ∪ {t} }
```

... Relational Algebra Operations

34/71

Projection

- $\pi_X(r)$ = $Proj[X](r)$ = $\{ t[X] \mid t \in r \}$
- $X \subseteq R$; result schema is given by attributes in X

Computational view:

```
result = {}
for each tuple t in relation r
    result = result ∪ {t[X]}
```

... Relational Algebra Operations

35/71

Set operations involve two relations $r(R)$, $s(R)$ (union-compatible)

Union

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

Computational view:

```
result = $r_1$
for each tuple t in relation $r_2$
    result = result ∪ {t}
```

... Relational Algebra Operations

36/71

Intersection

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

Computational view:

```
result = {}
for each tuple t in relation $r_1$
    if (t ∈ $r_2$) { result = result ∪ {t} }
```

... Relational Algebra Operations

37/71

Theta Join

- $r \bowtie_C s$ = $Join[C](r,s)$ =
  $\{ (t_1 : t_2) \mid t_1 \in r \wedge t_2 \in s \wedge C(t_1 : t_2) \}$, where $r(R)$,$s(S)$
- C is the join condition (involving attributes from both relations)

Computational view:

```
result = {}
for each tuple $t_1$ in relation r
   for each tuple $t_2$ in relation s
      if (matches($t_1$,$t_2$,C))
         result = result ∪ {concat($t_1$,$t_2$)}
```

... Relational Algebra Operations

Left Outer Join

- $Join_{LO}[C](R,S)$ includes entries for all *R* tuples
- even if they have no matches with tuples in *S* under *C*

Computational description of *r(R) LeftOuterJoin s(S)*:

```
result = {}
for each tuple $t_1$ in relation r
    nmatches = 0
    for each tuple $t_2$ in relation s
        if (matches($t_1$,$t_2$,C))
            result = result ∪ {combine($t_1$,$t_2$)}
            nmatches++
    if (nmatches == 0)
        result = result ∪ {combine($t_1$,$S_{null}$)}
```

where S_{null} is a tuple with schema *S* and all atributes set to NULL.

PostgreSQL

PostgreSQL

PostgreSQL is a full-featured open-source (O)RDBMS.

- provides a relational engine with:
  - efficient implementation of relational operations
  - very good transaction processing (concurrent access)
  - good backup/recovery (from application/system failure)
  - novel query optimisation (genetic algorithm-based)
  - replication, JSON, extensible indexing, etc. etc.
- already supports several non-standard data types
- allows users to define their own data types
- supports most of the SQL3 standard

PostgreSQL Online

Web site: www.postgresql.org

Key developers: Bruce Momjian, Tom Lane, Marc Fournier, ...

Full list of developers: www.postgresql.org/developer/bios

Local copy of source code:

http://www.cse.unsw.edu.au/~cs9315/16s1/postgresql/src.tar.bz2

Documentation is available via WebCMS menu.

User View of PostgreSQL

Users interact via SQL in a *client* process, e.g.

```
$ psql webcms;
psql (9.4.6)
Type "help" for help.
webcms2=# select * from calendar;
 id | course |   evdate   |       event
----+--------+------------+---------------------------
  1 |      4 | 2001-08-09 | Project Proposals due
 10 |      3 | 2001-08-01 | Tute/Lab Enrolments Close
 12 |      3 | 2001-09-07 | Assignment #1 Due (10pm)
 ...
```

or

```
$dbconn = pg_connect("dbname=webcms");
$result = pg_query($dbconn,"select * from calendar");
while ($tuple = pg_fetch_array($result))
   { ... $tuple["event"] ... }
```

PostgreSQL Functionality

PostgreSQL systems deal with various kinds of entities:

- *users* ... who can use the system, what they can do
- *groups* ... groups of users, for role-based privileges
- *databases* ... collections of schemas/tables/views/...
- *namespaces* ... to uniquely identify objects (schema.table.attr)
- *tables* ... collection of tuples (standard relational notion)
- *views* ... "virtual" tables (can be made updatable)
- *functions* ... operations on values from/in tables
- *triggers* ... operations invoked in response to events
- *operators* ... functions with infix syntax
- *aggregates* ... operations over whole table columns
- *types* ... user-defined data types (with own operations)
- *rules* ... for query rewriting (used e.g. to implement views)
- *access methods* ... efficient access to tuples in tables

... PostgreSQL Functionality

PostgreSQL's dialect of SQL is mostly standard (but with extensions).

Differences visible at the user-level:

- attributes containing arrays of atomic values
- table type inheritance, table-valued functions, ...

Differences at the implementation level:

- referential integrity checking is accomplished via triggers
- views are implemented via query re-writing *rules*

Example:

```
create view myview as select * from mytab;
-- is implemented as
create type as myview (same fields as mytab);
create rule myview as on select to myview
```

```
            do instead select * from mytab;
```

... PostgreSQL Functionality 45/71

PostgreSQL stored procedures differ from SQL standard:

- only provides functions, not procedures (but functions can return `void`)
- allows function overloading (same function name, diff argument types)
- defined at different "lexical level" to SQL
- provides own PL/SQL-like language for functions

... PostgreSQL Functionality 46/71

Example:

```
create or replace function
    barsIn(suburb text) returns setof Bars
as $$
declare
    r record;
begin
    for r in
        select * from Bars where location = suburb
    loop
       return next r;
    end loop;
end;
$$ language plpgsql;
used as e.g.
select * from barsIn('Randwick');
```

... PostgreSQL Functionality 47/71

Uses *multi-version concurrency control* (MVCC)

- multiple "versions" of the database exist together
- a transaction sees the version that was valid at its start-time
- readers don't block writers; writers don't block readers
- this significantly reduces the need for locking

Disadvantages of this approach:

- extra storage for old versions of tuples (`vacuum` fixes this)

PostgreSQL also provides locking to enforce critical concurrency.

... PostgreSQL Functionality 48/71

PostgreSQL has a well-defined and open extensibility model:

- stored procedures are held in database as strings
    - allows a variety of languages to be used
    - language interpreters can be integrated into PostgreSQL engine
- new data types, operators, aggregates, indexes can be added
    - typically requires code written in C, following defined API
    - for new data types, need to write input/output functions, ...

- for new indexes, need to implement file structures

Installing PostgreSQL

PostgreSQL is available via the COMP9315 web site.

Provided as tarball and zip in `~cs9315/web/16s1/postgresql/`

Brief summary of installation:

```
$ tar xfj ..../postgresql/src.tar.bz2
# creates a directory postgresql-9.4.6
$ configure --prefix=~/your/pgsql/directory
$ make
$ make install
$ source ~/your/environment/file
   # set up environment variables
$ initdb
   # set up postgresql configuration ... done once?
$ pg_ctl start
   # do some work with PostgreSQL databases
$ pg_ctl stop
```

PostgreSQL Configuration

A typical environment setup for COMP9315:

```
# Set up environment for running PostgreSQL
# Must be "source"d from sh, bash, ksh, ...

# can be any directory
PGHOME=/home/jas/srvr/pgsql
# data does not need to be under $PGHOME
export PGDATA=$PGHOME/data
export PGHOST=$PGDATA
export PGPORT=5432
export PATH=$PGHOME/bin:$PATH

alias p0="$D/bin/pg_ctl stop"
alias p1="$D/bin/pg_ctl -l $PGDATA/log start"
```

Using PostgreSQL for Assignments

You will need to modify then re-start the server:

```
# edit source code to make changes
$ pg_ctl stop
$ make
$ make install
# restore postgresql configuration
$ pg_ctl start
# run tests and analyse results
```

Assumes no changes that affect storage structures.

In this case, existing databases will continue to work ok.

... Using PostgreSQL for Assignments

If you change storage structures ...

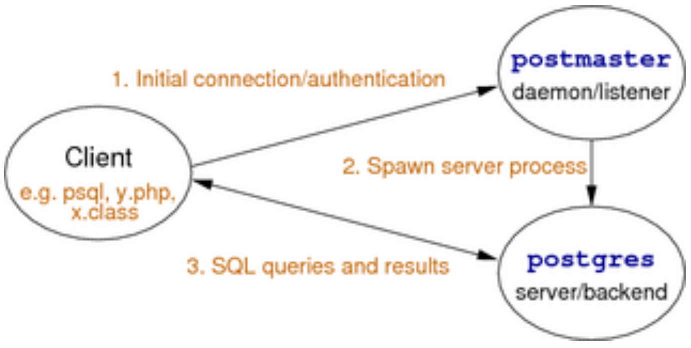
- old database will not work with the new server
- need to dump, re-run `initdb`, then restore

```
# edit source code to make changes
$ pg_dump testdb > testdb.dump
$ make
$ pg_ctl stop
$ rm -fr /your/pgsql/directory/data
$ make install
$ initdb
# restore postgresql configuration
$ pg_ctl start
$ createdb testdb
$ psql testdb -f testdb.dump
# run tests and analyse results
```

Need to save a copy of `postgresql.conf` before re-installing.

PostgreSQL Architecture

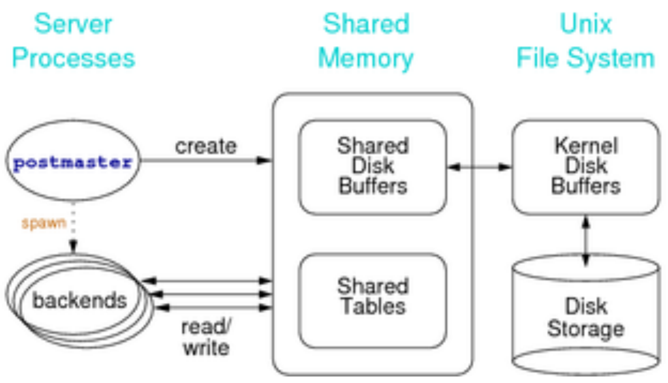
Client/server architecture:


1. Initial connection/authentication
postmaster
daemon/listener
Client
e.g. psql, y.php, x.class
2. Spawn server process
3. SQL queries and results
postgres
server/backend

Note: nowadays the `postmaster` process is also called `postgres`.

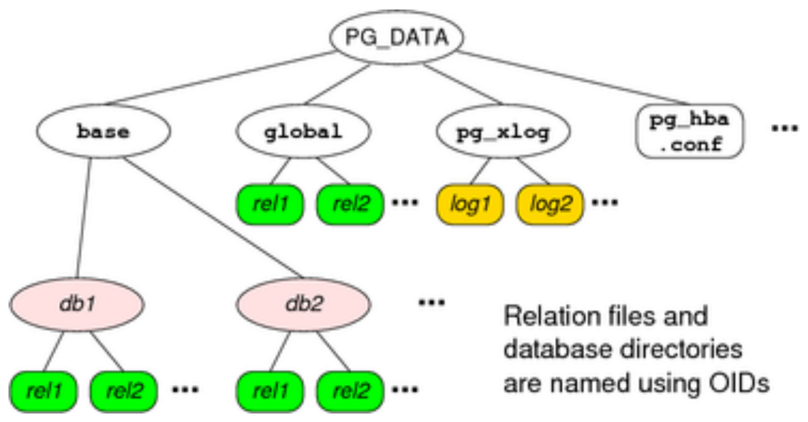
... PostgreSQL Architecture

Memory/storage architecture:



... PostgreSQL Architecture

File-system architecture:



PostgreSQL Source Code

Top-level of PostgreSQL distribution contains:

- *README,INSTALL*: overview and installation instructions
- *config**: scripts to build localised Makefiles
- *Makefile*: top-level script to control system build
- *src*: sub-directories containing system source code
- *doc*: FAQs and documentation (removed to save space)
- *contrib*: source code for contributed extensions

... PostgreSQL Source Code

The source code directory (*src*) contains:

- *include*: ***.h** files with global definitions (constants, types, ...)
- *backend*: code for PostgreSQL database engine
- *bin*: code for clients (e.g. psql, pg_ctl, pg_dump, ...)
- *pl*: stored procedure language interpreters (e.g. plpgsql)
- *interfaces* code for low-level C interfaces (e.g. libpq)

along with Makefiles to build system and other directories not relevant for us Code for backend (DBMS engine)

- 1500 files (880.c,620.h,6.y,7.l), *10*^{6} lines of code

... PostgreSQL Source Code

How to get started understanding the workings of PostgreSQL:

- become familiar with the user-level interface (psql, pg_dump, pg_ctl, etc.)
- start with the ***.h** files, then move to ***.c** files
  (note that: ***.c** files live under src/backend/*, ***.h** files live under src/include)
- start globally, then work one subsystem-at-a-time

Some helpful information is available via:

- PostgreSQL link on web site
- Readings link on web site

... PostgreSQL Source Code

PostgreSQL documentation has detailed description of internals:

- Section VII, Chapters 47,48,54-59
- Ch.47 is an overview; a good place to start
- other chapters discuss specific components

See also "How PostgreSQL Processes a Query"

- `src/tools/backend/index.html`

... PostgreSQL Source Code

`exec_simple_query(const char *query_string)`

- defined in `src/backend/tcop/postgres.c`
- entry point for evaluating SQL queries
- assumes `query_string` is one or more SQL statements

```
parsetree_list = pg_parse_query(query_string);
foreach(parsetree, parsetree_list) {
  querytree_list = pg_analyze_and_rewrite(parsetree, ...);
  plantree_list = pg_plan_queries(querytree_list, ...);
  portal = CreatePortal(...); // query execution env
  PortalDefineQuery(portal, ..., plantree_list, ...);
  receiver = CreateDestReceiver(dest); // client
  PortalRun(portal, ..., receiver, ...);
  ...
}
```

Catalogs

Database Objects

RDBMSs manage different kinds of objects

- databases, schemas, tablespaces
- relations/tables, attributes, tuples/records
- constraints, assertions
- views, stored procedures, triggers, rules

Many objects have names (and, in PostgreSQL, all have OIDs).

How are the different types of objects represented?

How do we go from a name (or OID) to bytes stored on disk?

... Database Objects

Consider what information the RDBMS needs about relations:

- name, owner, primary key of each relation
- name, data type, constraints for each attribute
- authorisation for operations on each relation

Similarly for other DBMS objects (e.g. views, functions, triggers, ...)

This information is stored in the *system catalog* tables

Most DBMSs implement their own internal catalog structure

Standard for catalogs in SQL:2003: `INFORMATION_SCHEMA`

(implemented in PostgreSQL as a set of views on the catalog, Ch.34)

... Database Objects

The catalog is manipulated by a range of SQL operations:

- `create` *Object* `as` *Definition*
- `drop` *Object* ...
- `alter` *Object* *Changes*
- `grant` *Privilege* `on` *Object*

where *Object* is one of table, view, function, trigger, schema, ...

E.g. `drop table Groups;` produces something like

```
delete from Tables where name = 'Groups';
```

... Database Objects

In PostgreSQL, the system catalog is available to users via:

- special commands in the `psql` shell (e.g. `\d`)
- SQL standard `information_schema`
  (e.g. `select * from information_schema.tables;`)

The low-level representation is available to sysadmins via:

- a global schema called `pg_catalog`
- a set of tables/views in that schema (e.g. `pg_tables`)

... Database Objects

A PostgreSQL installation typically has several databases.

Some catalog information is global, e.g.

- databases, users, ...
- one copy of each such table for the whole PostgreSQL installation
- shared by all databases in the installation (lives in `PGDATA/pg_global`)

Other catalog information is local to each database, e.g

- schemas, tables, attributes, functions, types, ...
- separate copy of each "local" table in each database
- a copy of many "global" tables is made on database creation

---

... Database Objects

Side-note: PostgreSQL tuples contain

- owner-specified attributes (from `create table`)
- system-defined attributes

`oid`	unique identifying number for tuple (optional)
`tableoid`	which table this tuple belongs to
`xmin/xmax`	which transaction created/deleted tuple (for MVCC)

OIDs are used as primary keys in many of the catalog tables.

Representing Databases

Above the level of individual DB schemata, we have:

- *databases* ... represented by `pg_database`
- *schemas* ... represented by `pg_namespace`
- *table spaces* ... represented by `pg_tablespace`

These tables are global to each PostgreSQL cluster.

Keys are names (strings) and must be unique within cluster.

... Representing Databases

`pg_database` contains information about databases:

- `oid, datname, datdba, datacl[], encoding, ...`

`pg_namespace` contains information about schemata:

- `oid, nspname, nspowner, nspacl[]`

`pg_tablespace` contains information about tablespaces:

- `oid, spcname, spcowner, spcacl[]`

PostgreSQL represents access via array of access items:

`Role=Privileges/Grantor`

where *Privileges* is a string enumerating privileges, e.g.

```
jas=arwdRxt/jas,fred=r/jas,joe=rwad/jas
```

Representing Tables

Representing one table needs tuples in several catalog tables.

Due to O-O heritage, base table for tables is called `pg_class`.

The `pg_class` table also handles other "table-like" objects:

- views ... represents attributes/domains of view
- composite (tuple) types ... from `CREATE TYPE AS`
- sequences, indexes (top-level defn), other "special" objects

All tuples in `pg_class` have an OID, used as primary key.

Some fields from the `pg_class` table:

- `oid, relname, relnamespace, reltype, relowner`
- `relkind, reltuples, relnatts, relhaspkey, relacl, ...`

Exercise 4: PostgreSQL Data Files

PostgreSQL uses OIDs as

- the name of the directory for each database
- the name of the files for each table

Using the `pg_catalog` tables, find ..

- the directory for the `pizza` database
- the data files for the `Pizzas` and `People` tables

Produced: 5 Mar 2016