Week 01 Lectures

COMP9315 19T2
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/~cs9315/19T2/)

Lecturer

2/100

Name:	John Shepherd
Office:	K17-410 (turn right from lift)
Phone:	9385 6494
Email:	jas@cse.unsw.edu.au
Consults:	still working out the details
Research:	Information Extraction/Integration Information Retrieval/Web Search e-Learning Technologies Multimedia Databases Query Processing

---

Course Admin

3/100

Email:	cs9315@cse.unsw.edu.au
Reasons:	Enrolment problems Special consideration Detailed assignment questions Technical issues

Course Goals

4/100

Introduce you to:

- architecture of relational DBMSs (e.g. 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 i ?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

Pre-requisites

5/100

We assume that you are already familiar with

- the C language and programming in C (or C++)
  (e.g. completed ≥ 1 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 very well, don't take this course.

Exercise 1: SQL (revision)

6/100

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 18s2
- for each student, give (student ID, name, mark)

Exercise 2: Unix File I/O (revision)

7/100

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.

Learning/Teaching

8/100

What's available for you:

- Textbooks: describe some syllabus topics in detail
- Notes: describe 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: Exercises and videos will be available only *after* the lecture.

... Learning/Teaching

9/100

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, Consults

- debugging is best done in person (can see full context)

Rough Schedule

10/100

Week 01	intro, dbms review, RA, catalogs
Week 02	storage: disks, buffers, pages, tuples
Week 03	RA ops: scan, sort, projection
Week 04	selection: heaps, hashing, indexes
Week 05	no lectures
Week 06	selection: N-d matching, similarity
Week 07	joins: naive, sort-merge, hash join
Week 08	query processing, optimisation

Week 09	transactions: concurrency, recovery
Week 10	distributed and non-SQL databases

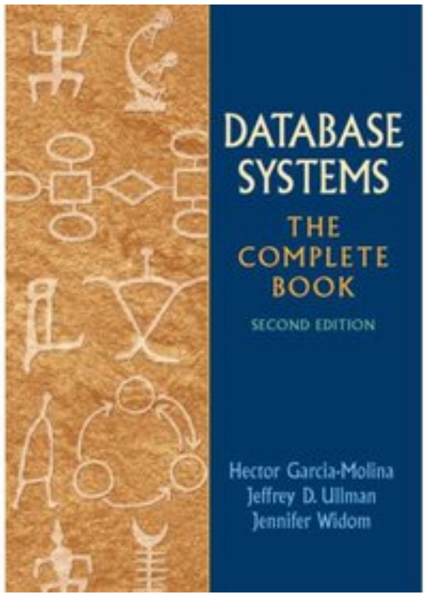
Textbooks

11/100

No official text book; several are suitable ...

- Garcia-Molina, Ullman, Widom
  "Database Systems: The Complete Book"
- Ramakrishnan, Gehrke
  "Database Systems Management"
- Silberschatz, Korth, Sudarshan
  "Database System Concepts"
- Kifer, Bernstein, Lewis
  "Database Systems: An algorithmic-oriented approach"
- Elmasri, Navathe
  "Database Systems: Models, languages, design ..."

but not all cover all topics in detail



Prac Work

12/100

In this course, we use PostgreSQL v11.3 (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

13/100

PostgreSQL is a *large* software system:

- > 1700 source code files in the core engine/clients
- > 1,000,000 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

14/100

Schedule of assignment work:

Ass	Description	Due	Marks
1	Storage Management	Week 5	13%
2	Query Processing	Week 10	17%

Assignments will be carried out in groups of size 1-3

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

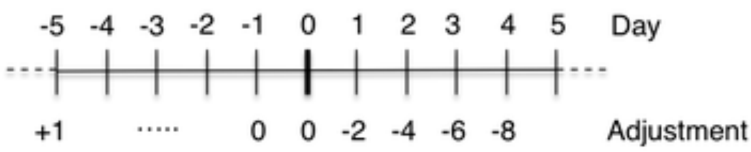
15/100

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

16/100

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

17/100

Three-hour exam in the August exam period.

Exam is held in CSE Labs (learn the environment, VLab)

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

Everything else is potentially examinable.

Contains: descriptive questions, analysis, small programming exercises.

Exam contributes 60% of the overall mark for this course.

... Exam

18/100

If you cannot attend the final exam ...

- because of documented illness/misadventure

then you will be offered a Supplementary Exam.

You get *one chance* at passing the exam

- unsw's new fit-to-sit rule applies
- exam hurdle 24/60 (which is only 40%)

Assessment Summary

19/100

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

```
ass1    = mark for assignment 1      (out of 13)
ass2    = mark for assignment 2      (out of 17)
quiz    = mark for on-line quizzes   (out of 10)
exam    = mark for final exam        (out of 60)
okExam = exam > 24/60              (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

21/100

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, stored 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

22/100

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)
);
```

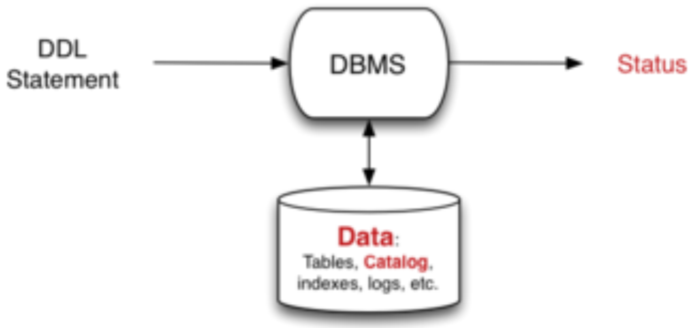
The above adds *meta-data* to the database.

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

... Data Definition

23/100

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



Result: meta-data in catalog is modified

Data Modification

24/100

Critical function of DBMS: changing data

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

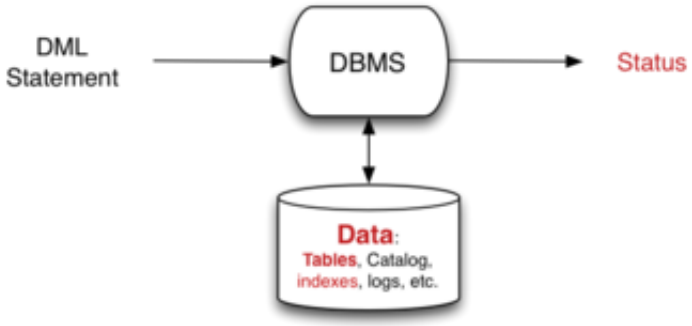
E.g.

```
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 = 3112233;
```

... Data Modification

25/100

Input: DML statements



Result: tuples are added, removed or modified

Query Evaluator

26/100

Most common function of relational DBMSs

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

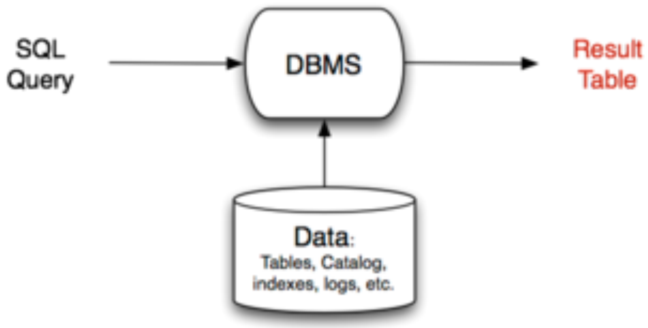
E.g.

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

... Query Evaluator

27/100

Input: SQL query



Output: table (displayed as text)

DBMS Architecture

28/100

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 (maybe) do a better job of writing efficient queries

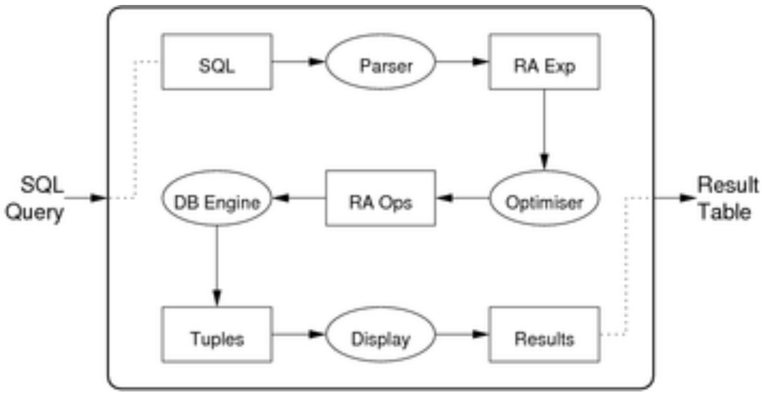
Educational reason:

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

... DBMS Architecture

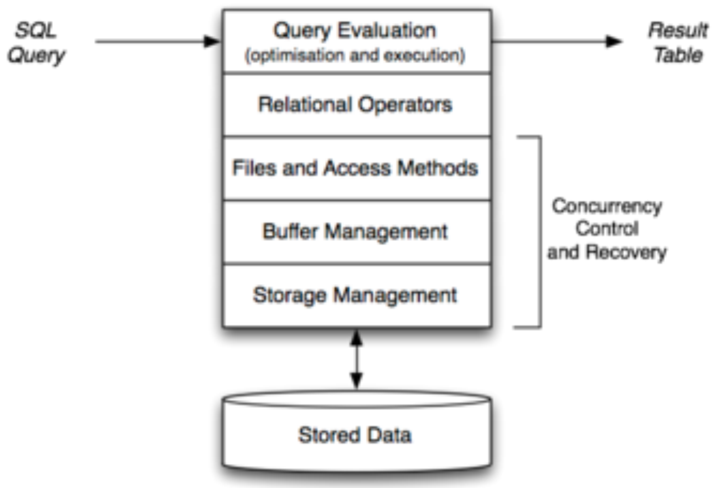
29/100

Path of a query through a typical DBMS:



... DBMS Architecture

30/100



... DBMS Architecture

31/100

Important factors related to DBMS architecture

- data is stored permanently on large slow devices**

- data is processed in small fast memory

Implications:

- data structures should minimise storage utilisation
- algorithms should minimise memory/disk data transfers

Modern DBMSs interact with storage via the O/S file-system.

** SSDs change things a little, but most high volume bulk storage still on disks

---

... DBMS Architecture

32/100

Implementation of DBMS operations is complicated by

- potentially multiple *concurrent accesses* to data structures
  (not just data tables, but indexes, buffers, catalogues, ...)
- *transactional* requirements (atomicity, rollback, ...)
- requirement for high *reliability* of raw data (recovery)

Require "concurrency-tolerant" data structures.

Transactions/reliability require some form of logging.

Database Engine Operations

33/100

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

34/100

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

35/100

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
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.

Describing Relational Algebra Operations

36/100

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

37/100

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

38/100

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

39/100

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 = r1
for each tuple t in relation r2
    result = result ∪ {t}
```

... Relational Algebra Operations

40/100

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 r1
    if (t ∈ r2) { result = result ∪ {t} }
```

... Relational Algebra Operations

41/100

Difference

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

Computational view:

```
result = {}
for each tuple t in relation r1
    if (!(t ∈ r2)) { result = result ∪ {t} }
```

... Relational Algebra Operations

42/100

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 t1 in relation r
   for each tuple t2 in relation s
      if (matches(t1,t2,C))
         result = result ∪ {concat(t1,t2)}
```

... 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 t1 in relation r
   nmatches = 0
   for each tuple t2 in relation s
      if (matches(t1,t2,C))
         result = result ∪ {combine(t1,t2)}
         nmatches++
   if (nmatches == 0)
      result = result ∪ {combine(t1,Snull)}
```

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

Exercise 3: Relational Algebra

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

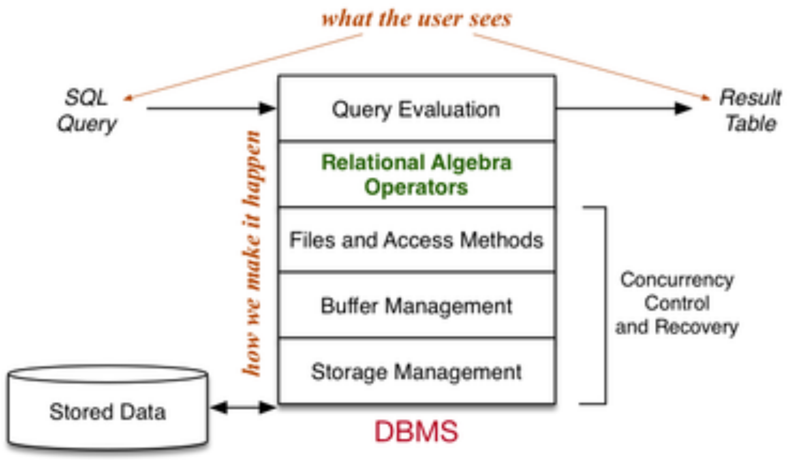
```
Students(sid, name, degree, ...)
Courses(cid, code, term, title, ...)
Enrolments(sid, cid, mark, grade)
```

Write relational algebra expressions to solve the problem

- find all students who passed COMP9315 in 18s2
- for each student, give (student ID, name, mark)

Express it as a sequence of steps, where each step uses one RA operation.

A Relational Algebra Engine



PostgreSQL

PostgreSQL

47/100

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

- provides a relational engine with:
  - efficient implementation of relational operations
  - transaction processing (concurrent access)
  - 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

48/100

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/~cs9315/19T2/postgresql/src.tar.bz2

Documentation is available via WebCMS menu.

User View of PostgreSQL

49/100

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

```
$ psql webcms
psql (11.3)
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

50/100

PostgreSQL systems deal with various kinds of entities:

- *users* ... who can access the system
- *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

51/100

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

- attributes containing arrays of atomic values

  ```
  create table R ( id integer, values integer[] );
  insert into R values ( 123, '{5,4,3,2,1}' );
  ```

- table type inheritance

  ```
  create table S ( x float, y float);
  create table T inherits ( R, S );
  ```

- table-valued functions

  ```
  create function f(integer) returns setof TupleType;
  ```

... PostgreSQL Functionality

52/100

PostgreSQL stored procedures differ from SQL standard:

- only provides functions, not procedures
  (but functions can return `void`, effectively a procedure)
- allows function overloading
  (same function name, different argument types)
- defined at different "lexical level" to SQL

- provides own PL/SQL-like language for functions

```
create function ( ArgTypes ) returns ResultType
as $$
... body of function definition ...
$$ language FunctionBodyLanguage;
```

... PostgreSQL Functionality

53/100

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

54/100

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

55/100

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 engine
- can add new data types, operators, aggregates, indexes
  - 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/Using PostgreSQL

Installing PostgreSQL

57/100

PostgreSQL is available via the COMP9315 web site.

Provided as tar-file in `~cs9315/web/19T2/postgresql/`

File: `src.tar.bz2` is ~20MB **

Unpacked, source code + binaries is ~130MB **

If using on CSE, do not put it under your home directory

Place it under `/srvr/`*`YOU`*`/` which has 500MB quota

Before Installing ...

58/100

If you have databases from previous DB courses

- the databases will no longer work under v11.3
- to preserve them, use dump/restore

E.g.

```
... login to grieg ...
... run your old server for the last time ...
$ pg_dump -O -x myFavDB > /srvr/YOU/myFavDB.dump
... stop your old server for the last time ...
... remove data from your old server ...
$ rm -fr /srvr/YOU/pgsql
... install and run your new PostgreSQL 11.3 server ...
$ createdb myFavDB
$ psql myFavDB -f /srvr/YOU/myFavDB.dump
... your old database is restored under 11.3 ...
```

Installing/Using PostgreSQL

59/100

Environment setup for running PostgreSQL in COMP9315:

```
# 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"
```

Will probably work (with tweaks) on home laptop if Linux or MacOS

... Installing/Using PostgreSQL

60/100

Brief summary of installation:

```
$ tar xfj ..../postgresql/src.tar.bz2
  # create a directory postgresql-11.3
$ source ~/your/environment/file
  # set up environment variables
$ configure --prefix=$PGHOME
$ make
$ make install
$ initdb
  # set up postgresql configuration ... done once?
$ edit postgresql.conf
$ pg_ctl start -l $PGDATA/log
  # do some work with PostgreSQL databases
$ pg_ctl stop
```

On CSE machines, `~cs9315/bin/pgs` can simplify some things

Using PostgreSQL for Assignments

61/100

If changes don't modify storage structures ...

```
$ edit source code
$ pg_ctl stop
$ make
$ make install
$ pg_ctl start -l $PGDATA/log
  # run tests, analyse results, ...
$ pg_ctl stop
```

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

... Using PostgreSQL for Assignments

62/100

If changes modify storage structures ...

```
$ edit source code
$ save a copy of postgresql.conf
$ pg_dump testdb > testdb.dump
$ pg_ctl stop
$ make
$ make install
$ rm -fr $PGDATA
$ initdb
$ restore postgresql.conf
$ pg_ctl start -l $PGDATA/log
$ createdb testdb
$ psql testdb -f testdb.dump
  # run tests and analyse results
```

Old databases will not work with the new server.

... Using PostgreSQL for Assignments

63/100

Troubleshooting ...

- read the `$PGDATA/log` file
- which socket file are you trying to connect to?

- check the `$PGDATA` directory for socket files
- remove `postmster.pid` if sure no server running
- ...

Prac Exercise P01 has useful tips down the bottom

Catalogs

Database Objects

65/100

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?

Catalogs

66/100

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

Standard for catalogs in SQL:2003: `INFORMATION_SCHEMA`

... Catalogs

67/100

The catalog is affected by several types 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  schema = 'public' and name = 'groups';
```

... Catalogs

68/100

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`)

... Catalogs

69/100

You can explore the PostgreSQl catalog via `psql` commands

- **`\d`** gives a list of all tables and views
- **`\d`** ***`Table`*** gives a schema for *`Table`*
- **`\df`** gives a list of user-defined functions
- **`\df+`** ***`Function`*** gives details of *`Function`*
- **`\ef`** ***`Function`*** allows you to edit *`Function`*
- **`\dv`** gives a list of user-defined views
- **`\d+`** ***`View`*** gives definition of *`View`*

You can also explore via SQL on the catalog tables

... Catalogs

70/100

A PostgreSQL installation (cluster) typically has many DBs

Some catalog information is global, e.g.

- catalog tables defining: databases, users, ...
- one copy of each such table for the whole PostgreSQL installation
- shared by all databases in the cluster (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

---

... Catalogs

71/100

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

72/100

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

73/100

`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

74/100

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, ...`

... Representing Tables

75/100

Details of catalog tables representing database tables `pg_class` holds core information about tables

- `relname, relnamespace, reltype, relowner, ...`
- `relkind, relnatts, relhaspkey, relacl[], ...`

`pg_attribute` contains information about attributes

- `attrelid, attname, atttypid, attnum, ...`

`pg_type` contains information about types

- `typname, typnamespace, typowner, typlen, ...`
- `typtype, typrelid, typinput, typoutput, ...`

Exercise 4: Table Statistics

76/100

Using the PostgreSQL catalog, write a PLpgSQL function

- to return table name and #tuples in table
- for all tables in the `public` schema

```
create type TableInfo as (table text, ntuples int);
create function pop() returns setof TableInfo ...
```

Hints:

- `table` is a reserved word
- you will need to use dynamically-generated queries.

Exercise 5: Extracting a Schema

77/100

Write a PLpgSQL function:

- `function schema() returns setof text`
- giving a list of table schemas in the `public` schema

It should behave as follows:

```
db=# select * from schema();
            tables
---------------------------
 table1(x, y, z)
 table2(a, b)
 table3(id, name, address)
...
```

Exercise 6: Enumerated Types

78/100

PostgreSQL allows you to define enumerated types, e.g.

```
create type Mood as enum ('sad', 'happy');
```

Creates a type with two ordered values `'sad' < 'happy'`

What is created in the catalog for the above definition?

Hint:

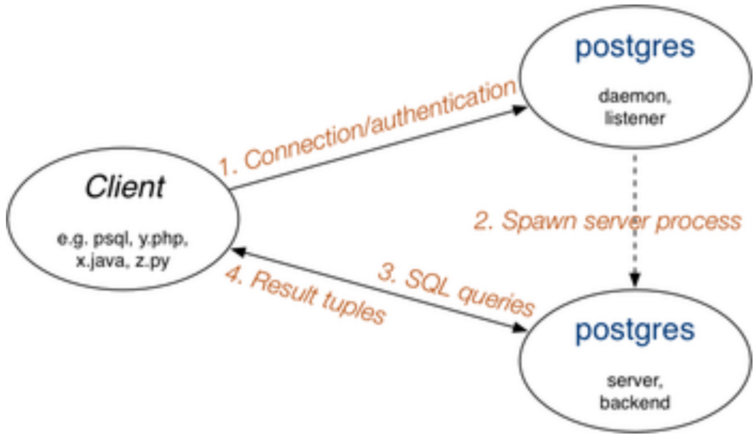
```
pg_type(oid, typname, typelen, typetype, ...)
pg_enum(oid, enumtypid, enumlabel)
```

PostgreSQL Architecture

PostgreSQL Architecture

80/100

Client/server architecture:

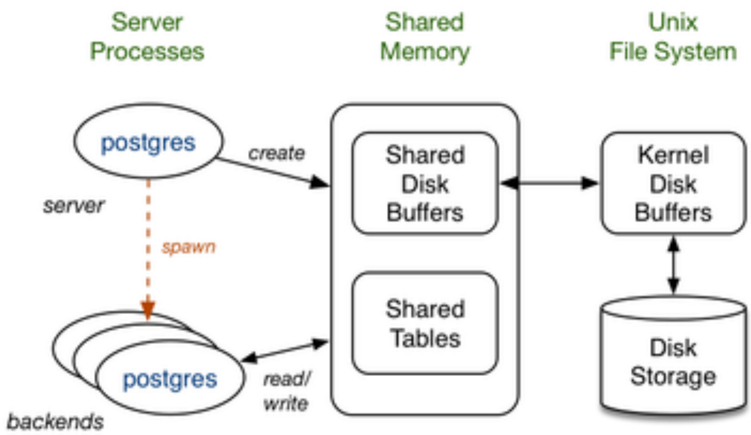


The listener process is sometimes called `postmaster`

... PostgreSQL Architecture

81/100

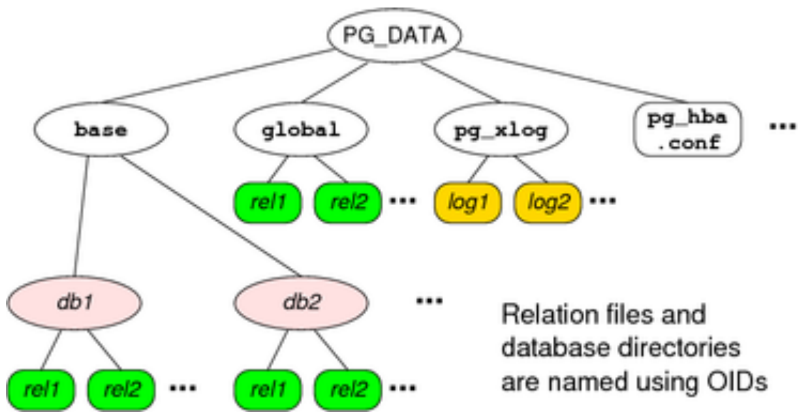
Memory/storage architecture:



... PostgreSQL Architecture

82/100

File-system architecture:



Exercise 7: PostgreSQL Data Files

83/100

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

Relevant catalog info ...

```
pg_database(oid,datname,...)
pg_class(oid,relname,...)
```

PostgreSQL Source Code

84/100

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

85/100

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 ...

Code for backend (DBMS engine)

- ~1700 files (~1000.c, ~700.h, 8.y, 10.l), 10^6 lines of code

---

... PostgreSQL Source Code

86/100

How to get started understanding the workings of PostgreSQL:

- become familiar with the user-level interface
  - psql, pg_dump, pg_ctl
- start with the ***.h** files, then move to ***.c** files
  - ***.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 Doco link on web site
- Readings link on web site

... PostgreSQL Source Code

87/100

PostgreSQL documentation has detailed description of internals:

- Section VII, Chapters 51 - 70
- Ch.51 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`

Life-cycle of a PostgreSQL query

88/100

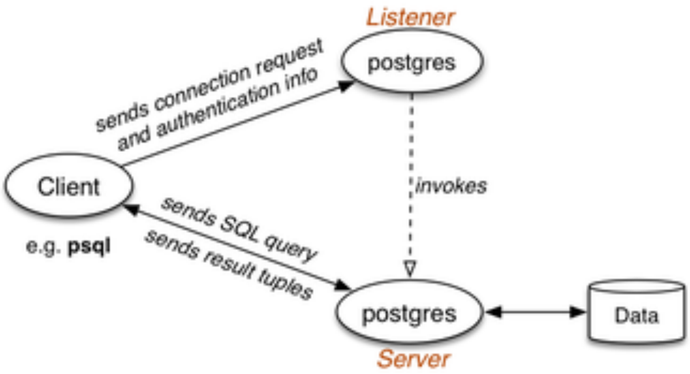
How a PostgreSQL query is executed:

- SQL query string is produced in client
- client establishes connection to PostgreSQL
- dedicated server process attached to client
- SQL query string sent to server process
- *server parses/plans/optimises query*
- server executes query to produce result tuples
- tuples are transmitted back to client
- client disconnects from server

... Life-cycle of a PostgreSQL query

89/100

Data flow to get to execute a query:



PostgreSQL server

90/100

`PostgresMain(int argc, char *argv[], ...)`

- defined in `src/backend/tcop/postgres.c`
- PostgreSQL server (`postgres`) main loop
- performs much setting up/initialisation
- reads and executes requests from client
- using the frontend/backend protocol (Ch.46)
- on `Q` request, evaluates supplied query
- on `X` request, exits the server process

... PostgreSQL server

91/100

As well as handling SQL queries, `PostgresqlMain` also

- handles "utility" commands e.g. `CREATE TABLE`
  - most utility commands modify catalog (e.g. `CREATE` *X*)
  - other commands affect server (e.g. `vacuum`)
- handles `COPY` command
  - special `COPY` mode; context is one table
  - reads line-by-line, treats each line as tuple
  - inserts tuples into table; at end, checks constraints

PostgreSQL Data Types

92/100

Data types defined in `*.h` files under `src/include/`

Two important data types: **`Node`** and **`List`**

- `Node` provides generic structure for nodes
  - defined in `src/include/nodes/nodes.h`
  - specific node types defined in `src/include/nodes/*.h`
  - functions on nodes defined in `src/backend/nodes/*.c`
  - `Node` types: parse trees, plan trees, execution trees, ...
- `List` provides generic singly-linked list
  - defined in `src/include/nodes/pg_list.h`
  - functions on lists defined in `src/backend/nodes/list.c`

PostgreSQL Query Evaluation

93/100

`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
- performs much setting up/initialisation
- parses the SQL string (into one or more parse trees)
- for each parsed query ...
  - perform any rule-based rewriting
  - produces an evaluation plan (optimisation)
  - execute the plan, sending tuples to client

... PostgreSQL Query Evaluation

94/100

`pg_parse_query(char *sqlStatements)`

- defined in `src/backend/tcop/postgres.c`
- returns list of parse trees, one for each SQL statement

`pg_analyze_and_rewrite(Node *parsetree, ...)`

- defined in `src/backend/tcop/postgres.c`
- converts parsed queries into form suitable for planning

... PostgreSQL Query Evaluation

95/100

Each query is represented by a **`Query`** structure

- defined in `src/include/nodes/parsenodes.h`
- 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`
- queries may be nested, so forms a tree structure

... PostgreSQL Query Evaluation

96/100

`pg_plan_queries(querytree_list, ...)`

- defined in `src/backend/tcop/postgres.c`
- converts analyzed queries into executable "statements"
- uses `pg_plan_query()` to plan each `Query`
  - defined in `src/backend/tcop/postgres.c`
- uses `planner()` to actually do the planning
  - defined in `optimizer/plan/planner.c`

... PostgreSQL Query Evaluation

97/100

Each executable query is represented by a **`PlannedStmt`** node

- defined in `src/include/nodes/plannodes.h`
- contains information for execution of query, e.g.
  - which relations are involved, output tuple struecture, etc.
- most important component is a tree of **`Plan`** nodes

Each `Plan` node represents one relational operation

- types: `SeqScan`, `IndexScan`, `HashJoin`, `Sort`, ...

- each `Plan` node also contains cost estimates for operation

... PostgreSQL Query Evaluation

98/100

`PlannedStmt *planner(Query *parse, ...)`

- defined in `optimizer/plan/planner.c`
- `subquery_planner()` performs standard transformations
    - e.g. push selection and projection down the tree
- then invokes a cost-based optimiser:
    - choose possible plan (execution order for operations)
    - choose physical operations for this plan
    - estimate cost of this plan (using DB statistics)
    - do this for *sufficient* cases and pick cheapest

... PostgreSQL Query Evaluation

99/100

Queries run in a **`Portal`** environment containing

- the planned statement(s) (trees of `Plan` nodes)
- run-time versions of `Plan` nodes (under `QueryDesc`)
- description of result tuples (under `TupleDesc`)
- overall state of scan through result tuples (e.g. `atStart`)
- other context information (transaction, memory, ...)

`Portal` defined in `src/include/utils/portal.h`

`PortalRun()` function also requires

- destination for query results (e.g. connection to client)
- scan direction (forward or backward)

... PostgreSQL Query Evaluation

100/100

How query evaluation happens in `exec_simple_query()`:

- parse, rewrite and plan ⇒ `PlannedStmts`
- for each `PlannedStmt` ...
- create `Portal` structure
- then insert `PlannedStmt` into `portal`
- then set up `CommandDest` to receive results
- then invoke `PortalRun(portal,...,dest,...)`
- `PortalRun...()` invokes `ProcessQuery(plan,...)`
- `ProcessQuery()` makes `QueryDesc` from `plan`
- then invoke `ExecutorRun(qdesc,...)`
- `ExecutorRun()` invokes `ExecutePlan()` to generate result

Produced: 5 Jun 2019