Week 01 Lectures

COMP9315 18s2
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/~cs9315/18s2/)

Lecturer

Name:          John Shepherd
Office:        K17-410  (turn right from lift)
Phone:         9385 6494
Email:         jas@cse.unsw.edu.au
Consult:       Mon 2-3, Wed 11-12  (in K17-410)
Research:      Information Extraction/Integration
                Information Retrieval/Web Search
                e-Learning Technologies
                Multimedia Databases
                Query Processing

Course Admin

Name:          Jashank Jeremy
Email:         cs9315@cse.unsw.edu.au

(Email goes to both Jashank and me)

Reasons:       Enrolment problems
                Special consideration
                Detailed assignment questions
                Technical issues

Course Goals

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 non-relational databases

Develop skills in:
• analysing the performance of data-intensive algorithms
• the use of C to implement data-intensive algorithms

**Learning/Teaching**

What'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.

... 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)

**Rough Schedule**

Week 01    intro, dbms review, dbms architecture
Week 02    storage: disks, buffers, pages
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
"Mid"-term   no lectures
Week 10    distributed and non-SQL databases
Textbooks

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

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 18s2
- 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 v10.4 (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:

- > 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

Schedule of assignment work:

<table>
<thead>
<tr>
<th>Ass</th>
<th>Description</th>
<th>Due</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Storage Management</td>
<td>Week 5</td>
<td>10%</td>
</tr>
<tr>
<td>2</td>
<td>Query Processing</td>
<td>Week 10</td>
<td>15%</td>
</tr>
</tbody>
</table>

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 15% of your overall mark for the course.

Exam

Three-hour exam in the November 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

Everything else is potentially examinable.

Contains: descriptive questions, analysis, small programming exercises.

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

... Exam

If you cannot attend the final exam ...

- because of documented illness/misadventure
- and you have reasonable marks in Ass+Quiz

then you will be offered a Supplementary Exam.

There is no other way to get a Supp Exam.

You get one chance at passing the exam

- make sure you're fit and healthy on exam day
- score more than 24/60 (which is only 40%)

Assessment Summary

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

\[
\begin{align*}
\text{ass1} &= \text{mark for assignment 1} \quad \text{(out of 10)} \\
\text{ass2} &= \text{mark for assignment 2} \quad \text{(out of 15)} \\
\text{quiz} &= \text{mark for online quizzes} \quad \text{(out of 15)} \\
\text{exam} &= \text{mark for final exam} \quad \text{(out of 60)} \\
\text{okExam} &= \text{exam > 24/60} \quad \text{(after scaling)} \\
\text{mark} &= \text{ass1 + ass2 + quiz + exam} \\
\text{grade} &= \text{HD|DN|CR|PS, if mark \geq 50 && okExam}
\end{align*}
\]
Relational Database Revision

Relational DBMS Functionality

Relational DBMSs provide a variety of functionalities:

- storing/modifying data and meta-data (data definitions)
- 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

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 = 3112233;

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**

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 which may be useful outside the (relational) DBMS context

---

Path of a query through a typical DBMS:
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

Database Engine Operations

DB engine = "relational algebra virtual machine":

- selection (σ)
- projection (π)
- join (∧)
- union (∪)
- intersection (∩)
- 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:
... 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
```

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

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)

Describing Relational Algebra Operations

We define the semantics of RA operations using

- "conditional set" expressions e.g. `{ x | condition }
- tuple notations:
  - `{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

Selection

```
σC(r) = Sel[C](r) = `{ t | t ∈ r ∧ 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 U {t} }
```

... Relational Algebra Operations

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

Computational view:

\[
\text{result} = \{ \}
\text{for each tuple } t \text{ in relation } r
\quad \text{result} = \text{result} \cup \{ t[X] \}
\]

---

### Relational Algebra Operations

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

**Union**

\[ r_1 \cup r_2 = \{ t \mid t \in r_1 \lor t \in r_2 \}, \quad \text{where } r_1(R), r_2(R) \]

Computational view:

\[
\text{result} = r_1
\text{for each tuple } t \text{ in relation } r_2
\quad \text{result} = \text{result} \cup \{ t \}
\]

---

### Relational Algebra Operations

**Intersection**

\[ r_1 \cap r_2 = \{ t \mid t \in r_1 \land t \in r_2 \}, \quad \text{where } r_1(R), r_2(R) \]

Computational view:

\[
\text{result} = \{ \}
\text{for each tuple } t \text{ in relation } r_1
\quad \text{if } (t \in r_2) \{ \text{result} = \text{result} \cup \{ t \} \}
\]

---

### Relational Algebra Operations

**Theta Join**

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

Computational view:

\[
\text{result} = \{ \}
\text{for each tuple } t_1 \text{ in relation } r
\quad \text{for each tuple } t_2 \text{ in relation } s
\quad \text{if } (\text{matches}(t_1, t_2, C))
\quad \quad \text{result} = \text{result} \cup \{ \text{concat}(t_1, t_2) \}
\]

---

### Relational Algebra Operations

**Left Outer Join**

\[ \text{Join}_{LO}[C](R,S) \text{ includes entries for all } R \text{ tuples} \]
\[ \text{even if they have no matches with tuples in } S \text{ under } C \]

Computational description of \( r(R) \text{ LeftOuterJoin } s(S) \):

\[
\text{result} = \{ \}
\text{for each tuple } t_1 \text{ in relation } r
\quad \text{nmatches} = 0
\]
for each tuple $t_2$ in relation $s$
  if (matches($t_1$, $t_2$, $C$))
    result = result $\cup$ \{combine($t_1$, $t_2$)}
    nmatches++
  if (nmatches == 0)
    result = result $\cup$ \{combine($t_1$, $S_{null}$)}

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

---

**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](http://www.postgresql.org)

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

Full list of developers: [www.postgresql.org/developer/bios](http://www.postgresql.org/developer/bios)

Local copy of source code:

http://www.cse.unsw.edu.au/~cs9315/18s2/postgresql/src.tar.bz2

Documentation is available via WebCMS menu.

---

**User View of PostgreSQL**

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

```sql
$ psql webcms
psql (10.4)
Type "help" for help.
webcms=# \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

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

- attributes containing arrays of atomic values
  
  ```sql
  create table R ( id integer, values integer[] );
  insert into R values ( 123, '{5,4,3,2,1}' );
  ```
  
- table type inheritance
  
  ```sql
  create table S ( x float, y float);
  create table T inherits ( R, S );
  ```

- table-valued functions
  
  ```sql
  create function f(integer) returns setof TupleType;
  ```

### PostgreSQL Functionality

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

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

### Example:

```sql
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

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

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

---

PostgreSQL Architecture

Client/server architecture:

The listener process is sometimes called postmaster

... PostgreSQL Architecture

Memory/storage architecture:
PostgreSQL Architecture

File-system architecture:

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

Relevant catalog info ...

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

Installing PostgreSQL

PostgreSQL is available via the COMP9315 web site.

Provided as tar-file in `~cs9315/web/18s2/postgresql/

File: src.tar.bz2 is ~15MB **

Unpacked, source code is ~130MB **

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

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

** Smaller than "normal" PG distribution ... documentation removed
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 PostgreSQL

Brief summary of installation:

$ tar xfj .../postgresql/src.tar.bz2
  # create a directory postgresql-10.4
$ 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

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

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
Old databases will not work with the new server.

... Using PostgreSQL for Assignments

Troubleshooting ...

- read the $PGDATA/log file
- which socket file are you trying to connect to?
- check the $PGDATA directory for socket files
- remove postmaster.pid if sure no server running
- ...

Prac Exercise P01 has useful tips down the bottom

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

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

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 documentation has detailed description of internals:

- Section VII, Chapters 50 - 68
Ch.60 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

```c
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, ...);
    ...
}
```

---

**Storage Management**

**Aims of storage management in DBMS:**

- provide view of data as collection of tables/tuples
- map from database objects (e.g. tables) to disk files
- manage transfer of data to/from disk storage
- use buffers to minimise disk/memory transfers
- interpret loaded data as tuples/records
- give foundation for file structures used by access methods

---

**Views of Data**

Users and top-level query evaluator see data as
- a collection of tables, each with a schema (tuple-type)
- where each table contains a set (sequence) of tuples

---

### Views of Data

Relational operators and access methods see data as

- sequence of fixed-size pages, typically 1KB to 8KB
- where each page contains tuple data or index data

---

### Views of Data

File manager sees both DB objects and file store

- maps (tableName, pageIndex) to (file, offset)

---

### Storage Management Topics

Topics to be considered:

- DB Object Management (Catalog)
  - how tables/functions/types, etc. are represented
- Disks and Files
  - performance issues and organisation of disk files
- Buffer Management
  - using caching to improve DBMS system throughput
- Tuple/Page Management
  - how tuples are represented within disk pages
Storage Manager Interface

The storage manager provides higher levels of system

- with an abstraction based on relations/pages/tuples
- which maps down to files/blots/records (via buffers)

Example: simple scan of a relation:

```sql
select student, course from Enrolments;
```

High-level view of result: sequence of tuples.

How is this mapped to accesses to files/blots/records?

... Storage Manager Interface

The query:

```sql
select student, course from Enrolments;
```

(Roughly) how it's executed:

```c
DB db = openDatabase("myDB");
Reln r = openRel(db,"Enrolments");
Scan s = startScan(r);
Tuple t;  Results res = NULL;
while ((t = nextTuple(s)) != NULL) {
    int stuid = getField(t,"student");
    char *course = getField(t,"course");
    res = addTuple(res, mkTuple(stuid,course));
}
```

... Storage Manager Interface

The storage manager provides mechanisms for:

- representing database objects during query execution
  - DB (handle on an authorised/opened database)
  - Reln (handle on an opened relation)
  - Page (memory buffer to hold contents of data block)
  - Tuple (memory holding data values from one tuple)
- referring to database objects (addresses)
  - symbolic (e.g. database/schema/table/field names)
  - abstract physical (e.g. PageId, TupleId)
Examples of references (addresses) used in DBMSs:

- **PageID** ... identifies (locates) a block of data
  - typically, \( \text{PageID} = \text{FileID} + \text{Offset} \)
  - where Offset gives location of block within file
- **TupleID** ... identifies (locates) a single tuple
  - typically, \( \text{TupleID} = \text{PageID} + \text{Offset} \)
  - where Offset gives location of tuple within page

Note that Offsets may be indexes into mapping tables giving real address.

Possible implementation for DB object ...

```c
typedef struct Database {
    char    *name; // database name
    Catalog  cat; // meta-data
    ...
} *DB;
```

Possible implementation of Reln object ...

```c
typedef struct Relation {
    char    *name; // table name
    File     file; // fd for table file
    ...
} *Reln;
```

Possible implementation for Scan object ...

- query executor wants to see result tuple-at-a-time
- DBMS read *blocks* from files (page-of-tuples-at-a-time)

```c
typedef struct ScanData {
    File  file; // file holding table data
    Page  page; // most recently read data
    int   pageno; // current block within file
    int   tupno; // current tuple within page
    ...
} *Scan;
```

startScan() might be implemented as:

```c
Scan startScan(Reln r)
{
    Scan s = MemAlloc(struct ScanData);
    s->file = r->file;
    s->page = null;
    s->pageno = 0;
    s->tupno = 0;
    return s;
}
```

And nextTuple() might be implemented as:
Tuple nextTuple(Scan s)
{
    if (noMoreTuplesIn(s->page,s->tupno))
        if (noMorePagesIn(s->file))
            return NULL;
    s->page = getPage(s->file,s->pageno);
    s->pageno++;
    s->tupno = 0;
}
Tuple t = getTuple(s->page,s->tupno);
s->tupno++;
return t;

From Symbolic to Internal

How do we determine ...

- information about a database, given its name
- information about a table, given its name

DBMSs use catalog data in special tables

E.g. for PostgreSQL

pg_database(oid, datname, datdba, datacl[], ...)
pg_namespace(oid, nspname, nspowner, nspacl[], ...)
pg_class(oid, relname, relnamespace, ..., relkind, reltuples, relnatts, relhaspkey, relacl[] ...)
pg_attribute(oid, attrelid, attname, atttypid, attnum, ...)
pg_type(oid, typname, typnamespace, typowner, typlen, ...)

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

Standard for catalogs in SQL:2003: INFORMATION_SCHEMA

... Database Objects
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';

---

**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 (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

---

**Database Objects**

Side-note: PostgreSQL tuples contain

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>unique identifying number for tuple (optional)</td>
</tr>
<tr>
<td>tableoid</td>
<td>which table this tuple belongs to</td>
</tr>
<tr>
<td>xmin/xmax</td>
<td>which transaction created/deleted tuple (for MVCC)</td>
</tr>
</tbody>
</table>

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, ...

... Representing Tables

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 5: Table Statistics
Using the PostgreSQL catalog, write a PL/pgSQL 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 ... 

Hint: you will need to use dynamically-generated queries.

Exercise 6: Extracting a Schema

Write a PL/pgSQL 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)
   ...

Produced: 26 Jul 2018