COMP9315 Course Welcome

COMP9315 23T1
DBMS Implementation

(Data structures and algorithms inside relational DBMSs)

Convenor: **Dong Wen**

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

❖ Teaching Staffs

Name: **Dong Wen**

Office: K17-507 (send me an email before coming)

Email: dong.wen@unsw.edu.au

Lectures: Weeks 1-4 (Tuesday 12:00-14:00, Thursday 12:00-14:00)

Research: Big Data Processing
Graph Mining and Processing
Distributed Algorithms

Name: **Junhua Zhang**

Lectures: Weeks 5-10 (Tuesday 12:00-14:00, Thursday 12:00-14:00)

Email: junhua.zhang@unsw.edu.au

Research: Graph Analysis
Road Network Processing

❖ Course Admin

Name: **Yiheng Hu**

Email: yiheng.hu@student.unsw.edu.au

Research: Graph Neural Networks

We also have several staffs from DKR (Data and Knowledge Research Group) for marking and consultation.

❖ Important Links

Course Website
https://webcms3.cse.unsw.edu.au/COMP9315/23T1/

Moodle for assessment submission
https://moodle.telt.unsw.edu.au/course/view.php?id=72721

Ed Forum
https://edstem.org/au/courses/10647/discussion/

Special consideration
https://student.unsw.edu.au/special-consideration

Educational Adjustments
https://student.unsw.edu.au/els

❖ Problems?

Ed Forum for general questions.

- Before posting, check that your query is not already answered.
- The forum entry can be found on Webcms.
- Enroll the forum using UNSW email.
- Enrollment link can be found from Webcms notice.
- Do not post assignment/exam solutions on forum.

Email dong.wen@unsw.edu.au for immediate and private questions.

- Technical issues (e.g. problems compiling PostgreSQL)
- Detailed assignment questions (shared-screen debugging)

❖ Course Goals

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

Develop skills in:

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

❖ Pre-requisites

We assume that you are already familiar with

- the C language and programming in 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

PostgreSQL, Assignments and Exam all involve C programming.

❖ Learning/Teaching

What's available for you:

- Textbooks: describe some syllabus topics in detail
- Lectures & Slides: introduce **topics** and show **practical cases**
- Practical/theoretical exercise
- Readings: research papers on selected topics
- General consultations

The onus is on you to make use of this material.

❖ Learning/Teaching (cont)

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

Week 01	welcome, relational algebra, catalogs
Week 02	storage: disks, buffers, pages, tuples
Week 03	RA ops: scan, sort, projection
Week 04	selection: heaps, hashing, indexes
Week 05	selection: N-d matching, similarity
Week 06	*no new content, no online sessions*
Week 07	joins: naive, sort-merge, hash join
Week 08	query processing, optimisation
Week 09	transactions: concurrency, recovery
Week 10	database trends (guest lecturer?)

❖ Textbooks

No official text book; several are suitable ...

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

but not all cover all topics in detail

❖ Prac Work

In this course, we use PostgreSQL v15

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 MacOS at home
- PostgreSQL docs describe how to compile for Windows (WSL)

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

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

❖ Prac Work (cont)

PostgreSQL is a large software system:

- > 2000 source code files in the core engine/clients
- > 1,500,000 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.

We discuss relevant parts in lectures to help with this.

PostgreSQL books?

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

❖ Assessment Summary

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

```
ass1   = mark for assignment 1       (out of 15)
ass2   = mark for assignment 2       (out of 20)
quiz   = mark for on-line quizzes    (out of 15)
exam   = mark for final exam         (out of 50)
okExam = exam > 20/50              (after scaling)

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

❖ Assignments

Schedule of assignment work:

Ass	Description	Due	Marks
1	Storage Management	Week 5	15%
2	Query Processing	Week 9	20%

Assignments will be done individually.

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

Test cases available before submsission (extra tests after submission)

Ultimately, submission is via Moodle.

Late penalties apply; plagiarism checking will be used

❖ Quizzes

Over the course of the semester ...

- five online quizzes
- taken in your own time (but there are deadlines)
- each quiz is worth a small number of marks
- release on Monday and due on Friday

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 exam period.

Exam is held in CSE Labs; online exam is not available

The exam is totally open-book ("open-web").

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 50% of the overall mark for this course.

** 3-hours worth of work; 4-hours allowed to complete

❖ Exam (cont)

If you cannot attend the final exam ...

- because of documented illness/misadventure
- and you apply for special consideration (within 3 working days)

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 = 20/50 (which is only 40%)

❖ General Etiquette

The course website is a *workplace* platform

- make all communication professional and respectful

Summary: work hard and be nice to each other.

❖ Course Outline

All of the above is described in detail in the Course Outline.

Read it.

It forms a contract between you and me on how this course will run.

Additional resources:

- The Nucleus, in the Library
- Forms for various requests: unsw.to/webforms
- Student Counselling: student.unsw.edu.au/counselling

❖ Others...

We are seeking self-motivated and excellent students (e.g., Mphil and PhDs).

- Big Data (Graph) Processing;
- Distributed Algorithms;
- Streaming/Dynamic Data Processing;
- GPU Acceleration;
- AI4DB, GNN;
- ...

Send me an email with your CV and transcript.

Our group: https://unswdb.github.io/

❖ Week 01

Things to Note ...

- Slides available Mon/Tue each week
- Prac P01 now available ... install PostgreSQL

This Week's Topics ...

- DBMS Revision (Architecture, SQL, RelAlg, Catalogs)
- PostgreSQL introduction and installation
- Storage: devices/files (E16, G11, K9.1, S11)

Next Week's Topics ...

- buffer pool, page internals, tuple representation

❖ Text References

References to material in texts use format:

- BookChapter.Section.... (e.g. E2.1.1)

Books are denoted by:

- G = Garcia-Molina/Ullman/Widom
- S = Silberschatz/Korth/Sudarshan
- E = Elmasri/Navathe
- K = Kifer/Bernstein/Lewis

Produced: 13 Feb 2023

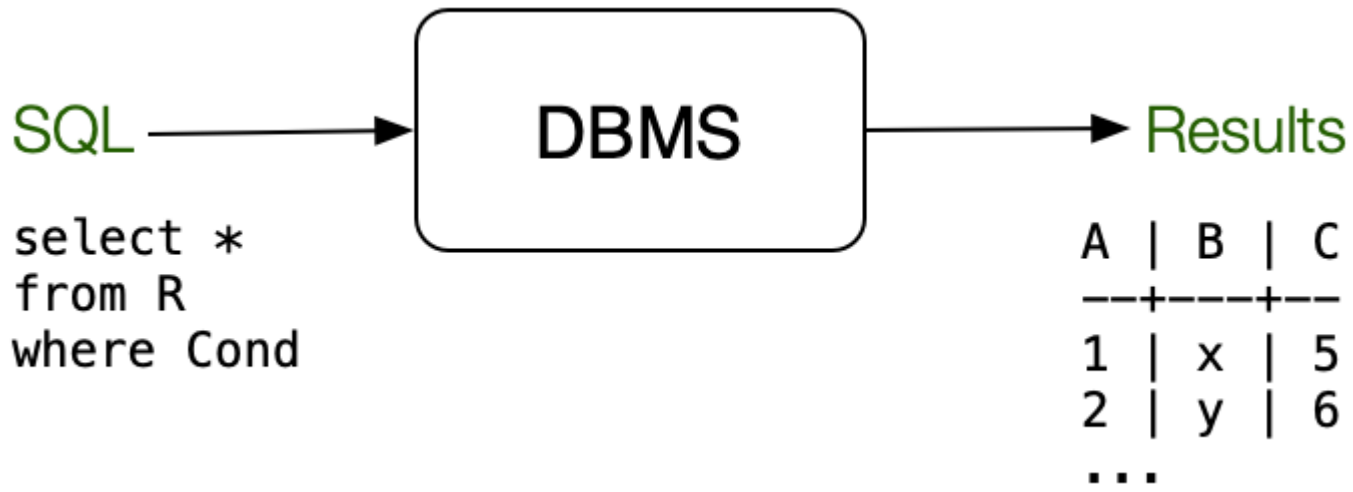
DBMS Overview

- DBMSs
- Query Evaluation
- Mapping SQL to RA
- Mapping Example
- Query Cost Estimation
- Implementations of RA Ops
- DBMS Architecture

❖ DBMSs

DBMS = DataBase Management System

Our view of the DBMS so far ...



A machine to process SQL queries.

❖ DBMSs (cont)

One view of DB engine: "relational algebra virtual machine"

Machine code for such a machine:

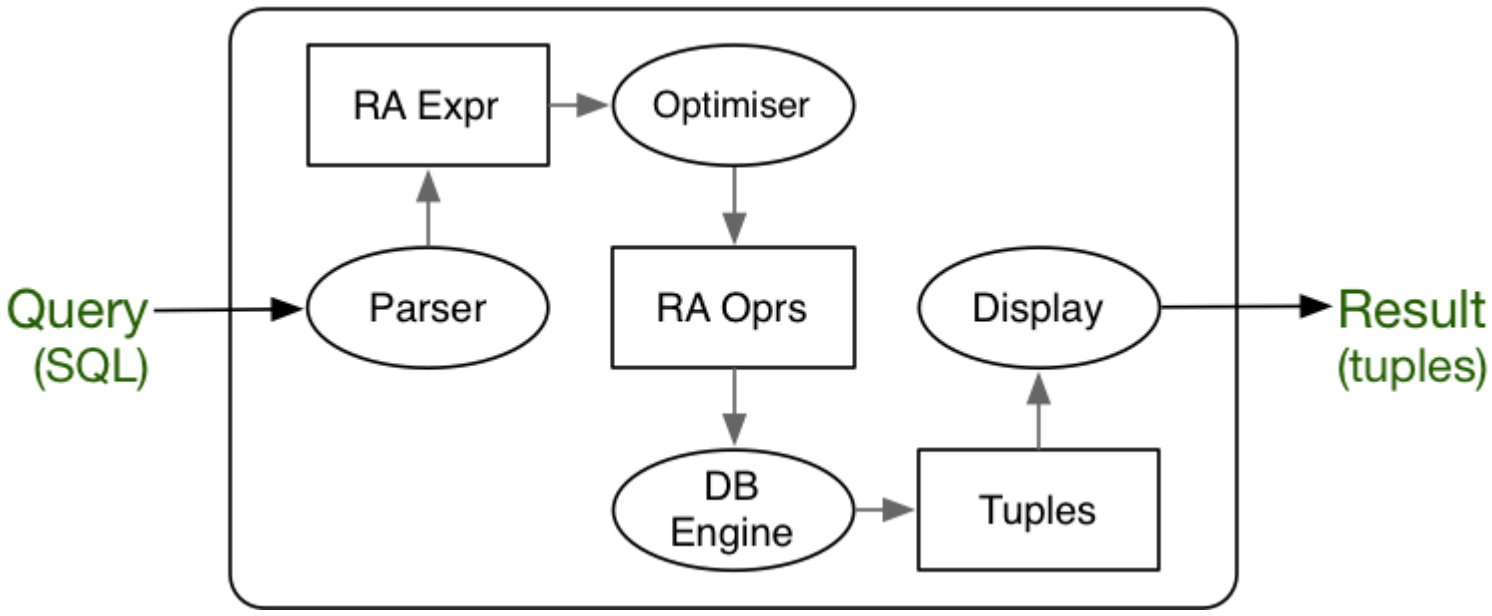
selection (σ)	projection (π)	join (⋈, ×)
union (∪)	intersection (∩)	difference (-)
sort	insert	delete

For each of these operations:

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

❖ Query Evaluation

The path of a query through its evaluation:



❖ Mapping SQL to RA

Mapping SQL to relational algebra, e.g.

```
-- schema: R(a,b) S(c,d)
select a as x
from   R join S on (b=c)
where  d = 100
-- could be mapped to
Tmp1(a,b,c,d) = R Join[b=c] S
Tmp2(a,b,c,d) = Sel[d=100](Tmp1)
Tmp3(a)       = Proj[a](Tmp2)
Res(x)        = Rename[Res(x)](Tmp3)
```

In general:

- **`SELECT`** clause becomes *projection*
- **`WHERE`** condition becomes *selection* or *join*
- **`FROM`** clause becomes *join*

❖ Mapping Example

Consider the database schema:

```
Person(pid, name, address, ...)
Subject(sid, code, title, uoc, ...)
Terms(tid, code, start, end, ...)
Courses(cid, sid, tid, ...)
Enrolments(cid, pid, mark, ..)
```

and the query: *Courses with more than 100 students in them?*

which can be expressed in SQL as

```
select s.sid, s.code
from   Course c join Subject s on (c.sid=s.sid)
       join Enrolment e on (c.cid=e.cid)
group  by s.sid, s.code
having count(*) > 100;
```

❖ Mapping Example (cont)

The SQL might be compiled to

```
Tmp1(cid,sid,pid)    = Course Join[c.cid = e.cid] Enrolment
Tmp2(cid,code,pid)   = Tmp1 Join[t1.sid = s.sid] Subject
Tmp3(cid,code,nstu) = GroupCount[cid,code](Tmp2)
Res(cid,code)        = Sel[nstu > 100](Tmp3)
```

or, equivalently

```
Tmp1(cid,code)       = Course Join[c.sid = s.sid] Subject
Tmp2(cid,code,pid)   = Tmp1 Join[t1.cid = e.cid] Enrolment
Tmp3(cid,code,nstu) = GroupCount[cid,code](Tmp2)
Res(cid,code)        = Sel[nstu > 100](Tmp3)
```

Which is better?

❖ Query Cost Estimation

The cost of evaluating a query is determined by

- the operations specified in the query execution plan
- size of relations (database relations and temporary relations)
- access mechanisms (indexing, hashing, sorting, join algorithms)
- size/number of main memory buffers (and replacement strategy)

Analysis of costs involves *estimating*:

- the size of intermediate results
- then, based on this, cost of secondary storage accesses

Accessing data from disk is the dominant cost in query evaluation

❖ Query Cost Estimation (cont)

Consider execution plans for: $\sigma_c (R \bowtie_d S \bowtie_e T)$ where R(c,d), S(d,e), T(e)

```
Tmp1(c,d,e)   :=  hash_join[d](R,S)
Tmp2(c,d,e)   :=  sort_merge_join[e](tmp1,T)
Res(c,d,e)    :=  binary_search[c](Tmp2)
```

or

```
Tmp1(d,e)     :=  sort_merge_join[e](S,T)
Tmp2(c,d,e)   :=  hash_join[d](R,Tmp1)
Res(c,d,e)    :=  linear_search[c](Tmp2)
```

or

```
Tmp1(c,d)     :=  btree_search[c](R)
Tmp2(c,d,e)   :=  hash_join[d](Tmp1,S)
Res(c,d,e)    :=  sort_merge_join[e](Tmp2,T)
```

All produce same result, but have different costs.

❖ Implementations of RA Ops

Sorting (quicksort, etc. are not applicable)

- external merge sort (cost $O(N log_B N)$ with B memory buffers)

Selection (different techniques developed for different query types)

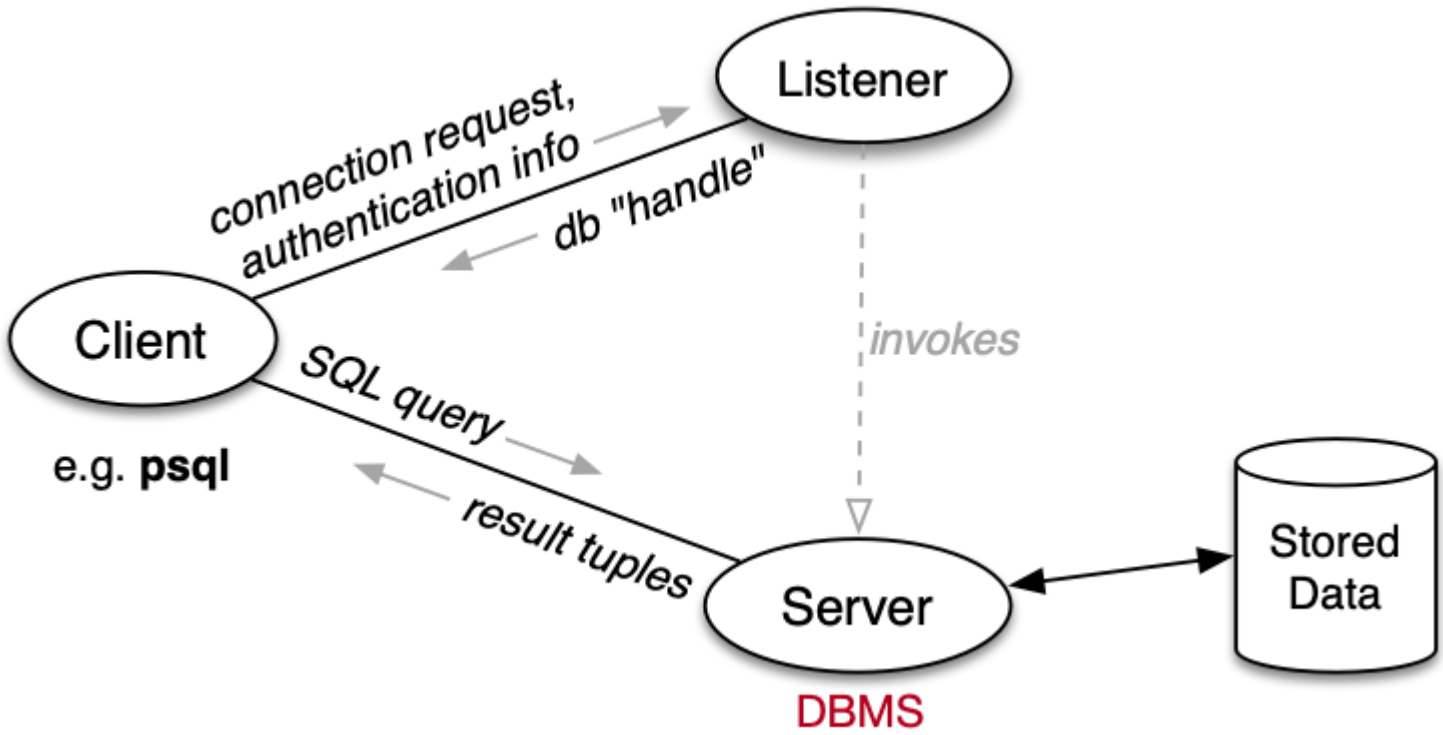
- sequential scan (worst case, cost $O(N)$)
- index-based (e.g. B-trees, cost $O(logN)$, tree nodes are pages)
- hash-based ($O(1)$ best case, only works for equality tests)

Join (fast joins are critical to success of relational DBMSs)

- nested-loop join (cost $O(N.M)$, buffering can reduce to $O(N+M)$)
- sort-merge join (cost $O(NlogN+MlogM)$)
- hash-join (best case cost $O(N+M.N/B)$, with B memory buffers)

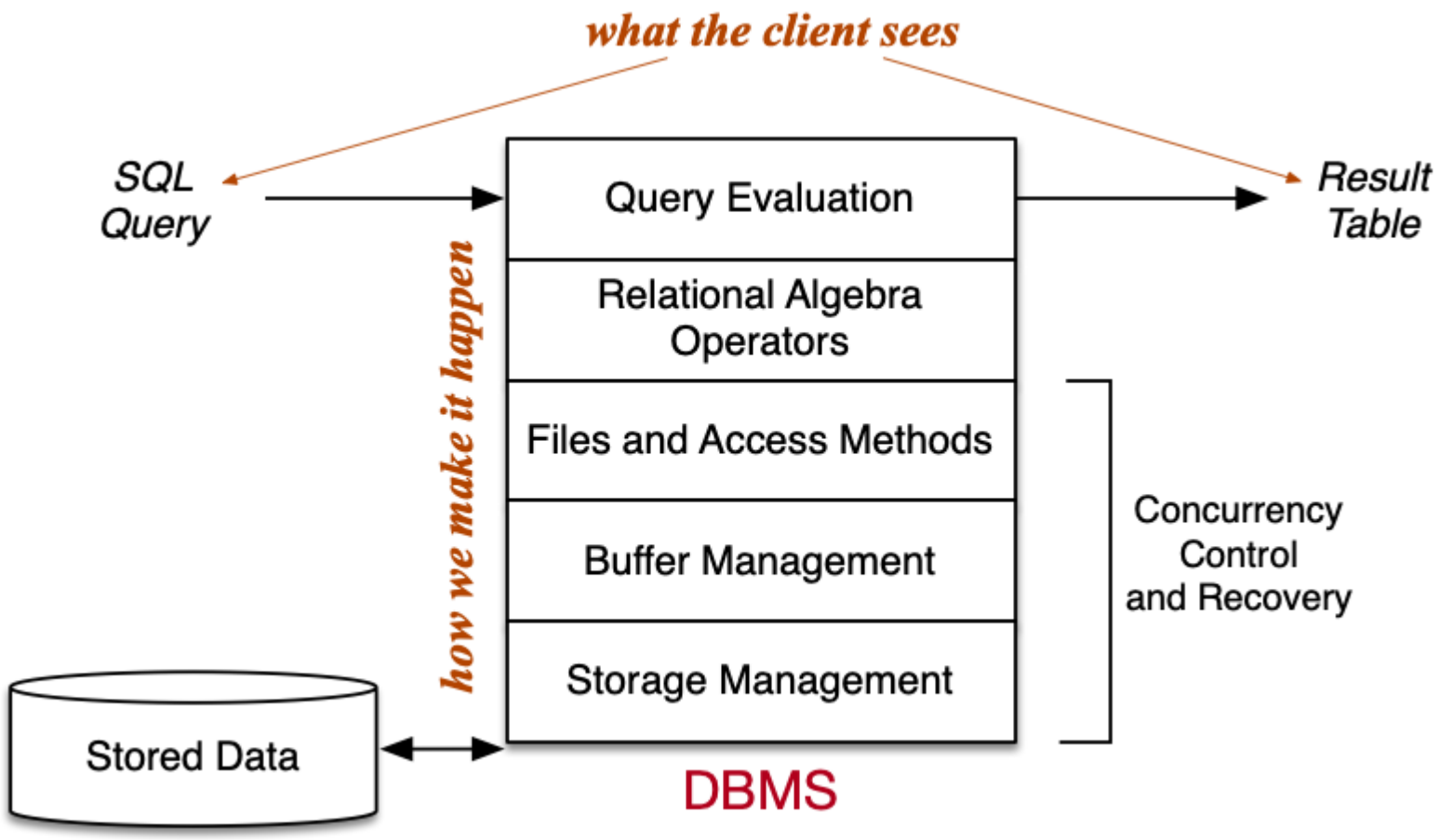
❖ DBMS Architecture

Most RDBMSs are client-server systems:



❖ DBMS Architecture (cont)

Layers within the DBMS server:



Produced: 29 Jan 2023

PostgreSQL Overview

- PostgreSQL
- PostgreSQL Online
- User View of PostgreSQL
- PostgreSQL Functionality
- PostgreSQL Architecture

❖ PostgreSQL

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 (based on genetic algorithm)
  - 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: Tom Lane, Andres Freund, Bruce Momjian, ...

Full list of developers: postgresql.org/contributors/

Source code on CSE server: /web/cs9315/23T1/postgresql/postgresql-15.1.tar.bz2

Documentation: postgresql.org/docs/15/index.html

❖ User View of PostgreSQL

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

```
$ psql webcms
psql (15.1)
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 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 (cont)

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-valued functions

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

- multiple langauges available for functions
  - PLpgSQL, Python, Perl, Java, R, Tcl, ...
  - function bodies are strings in whatever language

❖ PostgreSQL Functionality (cont)

Other variations in PostgreSQL's **`CREATE TABLE`**

- **`TEMPORARY`** tables
- **`PARTITION`**'d tables
- **`GENERATED`** attribute values (derived attributes)
- **`FOREIGN TABLE`** (data stored outside PostgreSQL)
- table type inheritance

```
create table R ( a integer, b text);
create table S ( x float, y float);
create table T inherits ( R, S );
```

❖ PostgreSQL Functionality (cont)

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

- where each ***`Arg`*** has a *Name* and *Type*

❖ PostgreSQL Functionality (cont)

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 (cont)

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 (until **`vacuum`**'d)
- need to check "visibility" of every tuple fetched

PostgreSQL also provides locking to enforce critical concurrency.

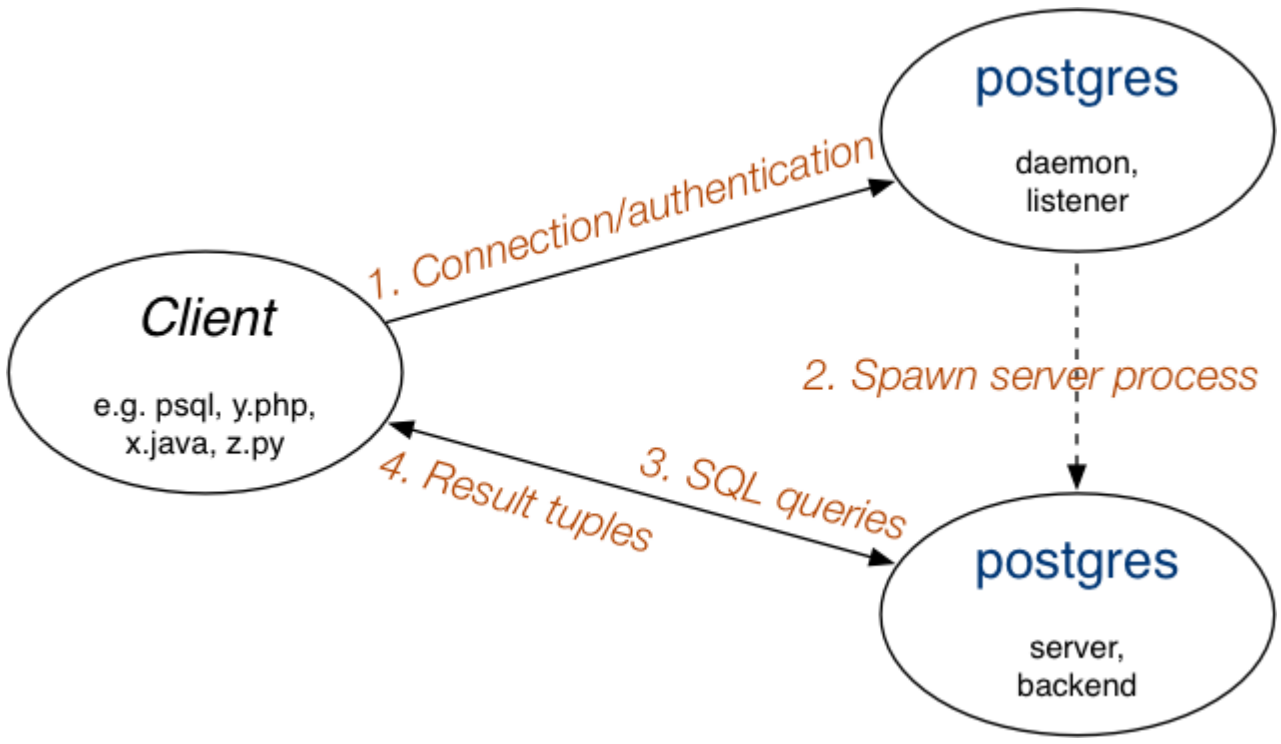
❖ PostgreSQL Functionality (cont)

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

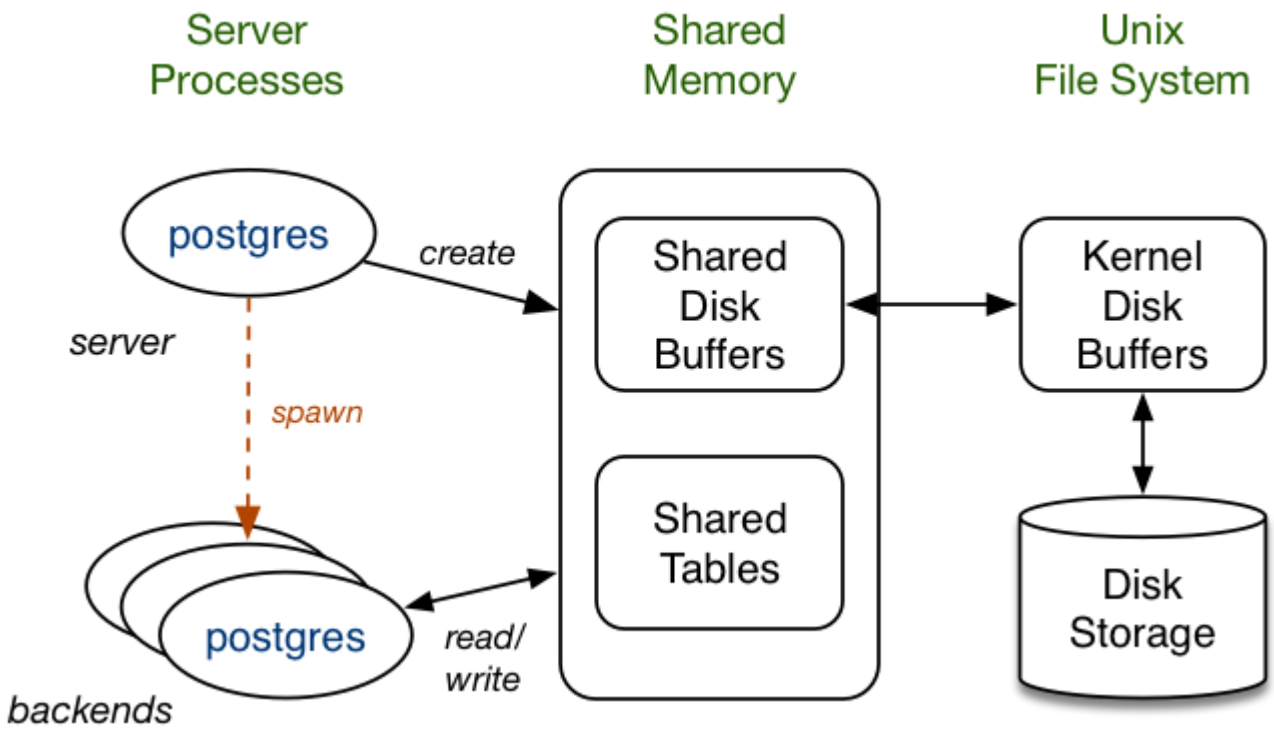
Client/server process architecture:



The listener process is sometimes called **`postmaster`**

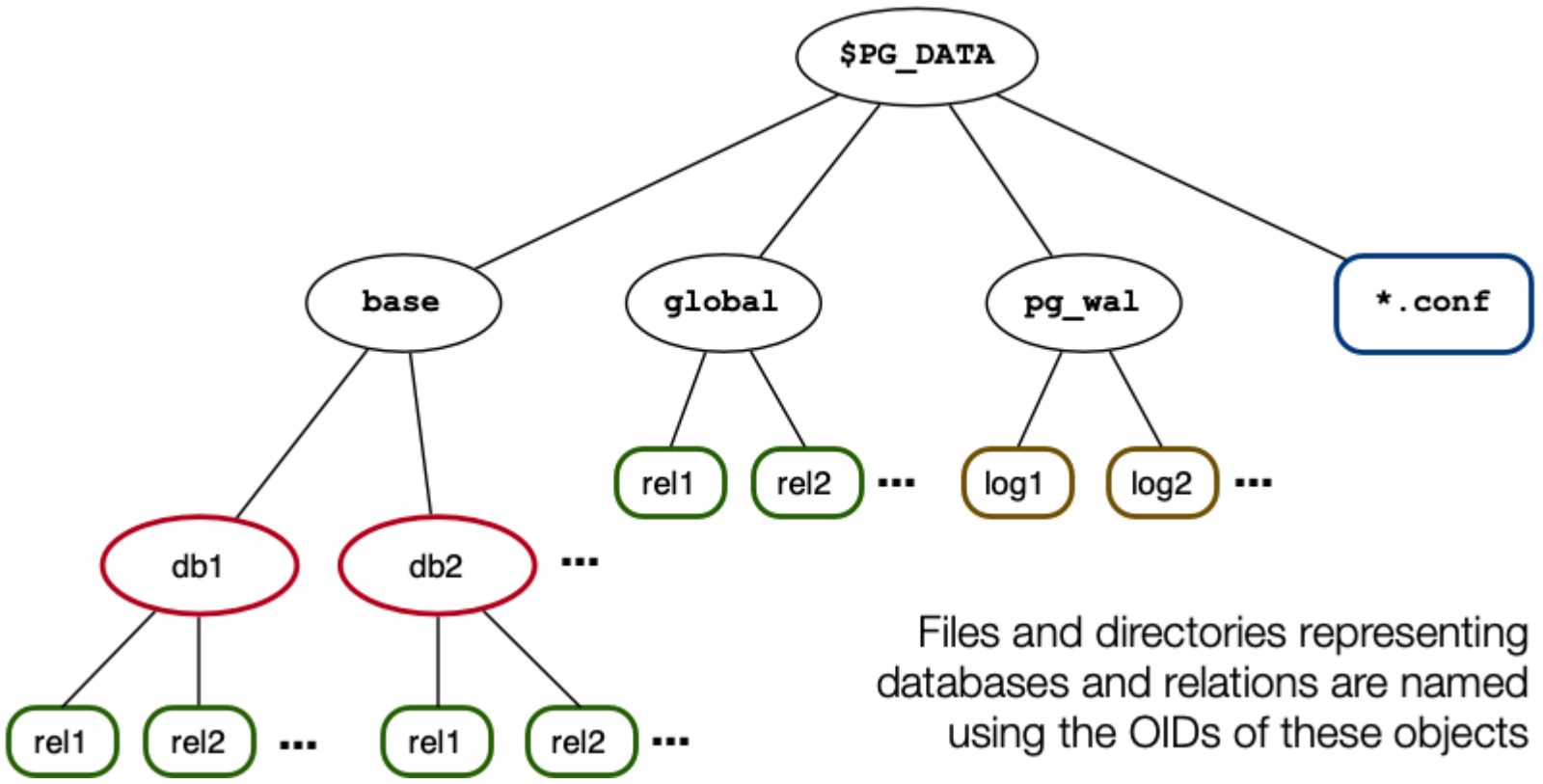
❖ PostgreSQL Architecture (cont)

Memory/storage architecture:



❖ PostgreSQL Architecture (cont)

File-system architecture:



Produced: 2 Feb 2023

PostgreSQL Install

- Installing/Using PostgreSQL
- Using PostgreSQL for Assignments
- Installing PostgreSQL
- Before Installing ...

❖ Installing/Using PostgreSQL

Environment setup for running PostgreSQL in COMP9315:

- **`nw-syd-vxdb`** a CSE server for database studies.
- See Prac Exercise 1 for details

Some hints/tips:

- Read instructions ... read log/error messages ... debug
- Give complete context when asking for help (OS, settings, etc)
- Don't work on other CSE servers
  - **`ssh zID@nw-syd-vxdb.cse.unsw.edu.au`**
  - binaries compiled on vxdb only run on vxdb
  - other incompatible versions of PostgreSQL exist
    try **`which psql`** ... if not the version under **`/localstorage/zID`** ...
- Stop the PostgreSQL server after playing with it

❖ 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 (cont)

If changes modify storage structures ...

```
$ edit source code
$ save a copy of postgresql.conf
$ pg_dump -O -x 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 (cont)

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

❖ Installing PostgreSQL

PostgreSQL is also available via the COMP9315 web site.
https://www.cse.unsw.edu.au/~cs9315/23T1/postgresql/postgresql-15.1.tar.bz2

File: **`src.tar.bz2`** is ~23MB **

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

Path on the CSE server:
`/web/cs9315/23T1/postgresql/postgresql-15.1.tar.bz2`

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

Place it under **`/localstorage/$USER/`** of the **`vxdb`** server which has large size quota

❖ Before Installing ...

If you have databases from previous DB courses

- the databases may not work under v12.5
- to preserve them, use dump/restore

E.g.

```
... login to the server or your machine ...
... 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 12.5 server ...
$ createdb myFavDB
$ psql myFavDB -f /srvr/YOU/myFavDB.dump
... your old database is restored under 12.5 ...
```

Produced: 11 Feb 2023

Catalogs

- Database Objects
- Catalogs
- Representing Databases
- Representing Tables

❖ 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, some 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

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 (cont)

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 (cont)

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 (cont)

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 (cont)

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 (cont)

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 (cont)

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.

```
Jake=rwad/Simon
```

❖ 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 (cont)

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, ...**

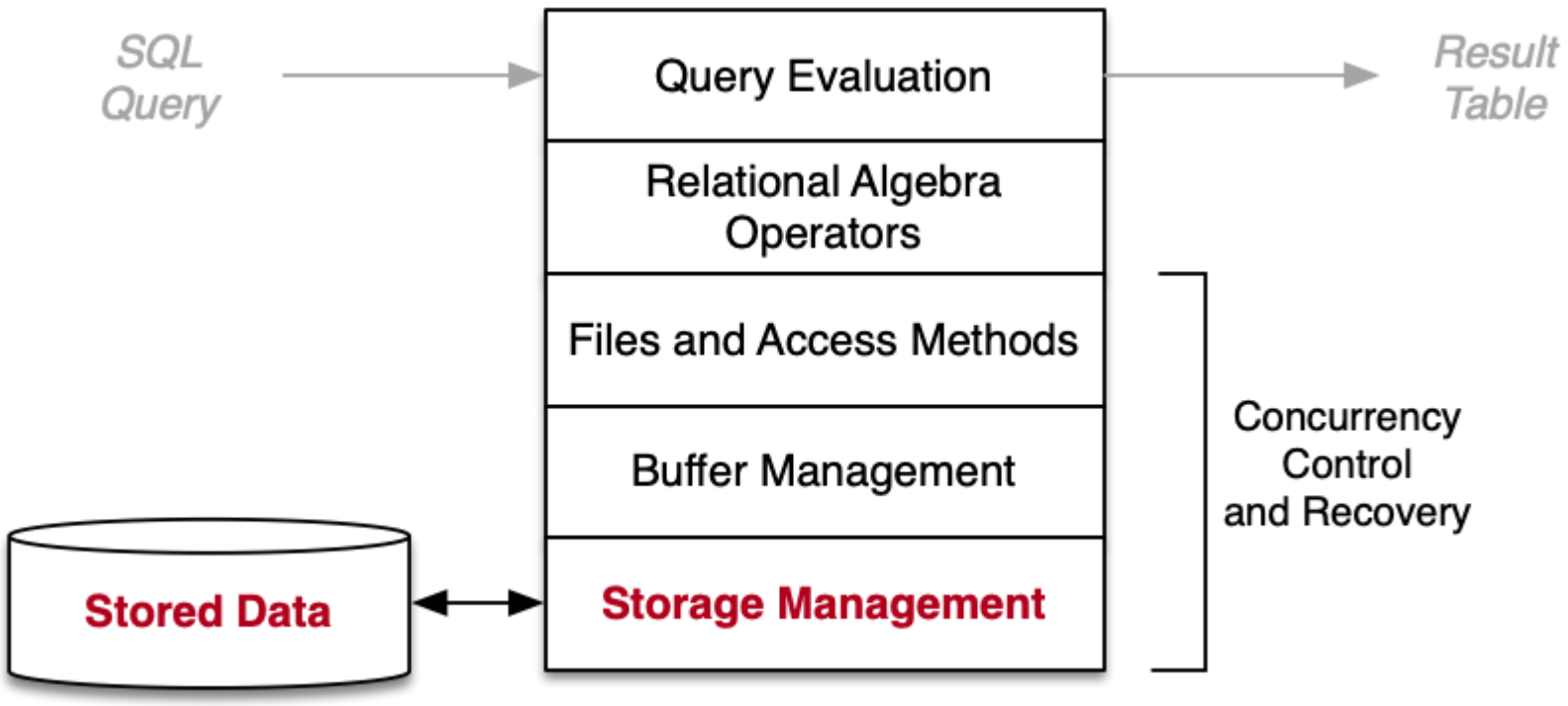
Produced: 2 Feb 2023

Storage Management

- Storage Management
- Storage Technology
- Views of Data in Query Evaluation
- Storage Management
- Cost Models

❖ Storage Management

Lowest levels of DBMS related to storage management:



❖ Storage Technology

Persistent storage is

- large, cheap, relatively slow, accessed in blocks
- used for long-term storage of data

Computational storage is

- small, expensive, fast, accessed by byte/word
- used for all analysis of data

Access cost HDD:RAM ≅ 100000:1, e.g.

- 10ms to read block containing two tuples
- 1μs to compare fields in two tuples

❖ Storage Technology (cont)

Hard disk drives (HDD) are well-established, cheap, high-volume, ...

- spinning magnetic medium
- access requires moving r/w head to position
- transfers blocks of data (e.g. 1KB)

Latency: move to track + spin to block = ~10ms (avg)

Volume: one HDD can store up to 20TB (typically 4TB/8TB/...)

Summary: very large, persistent, slow, block-based transfer

❖ Storage Technology (cont)

Solid state drives (SSD) are modern, high-volume devices ...

- faster than HDDs, no latency
- can read single items
- update requires block erase then write
- over time, writes "wear out" blocks
- require controllers that spread write load

Volume: one SSD can store up to 8TB (typically 1TB/2TB/...)

Summary: large, persistent, fast, (partly) block-based transfer

❖ Storage Technology (cont)

Comparison of storage device properties:

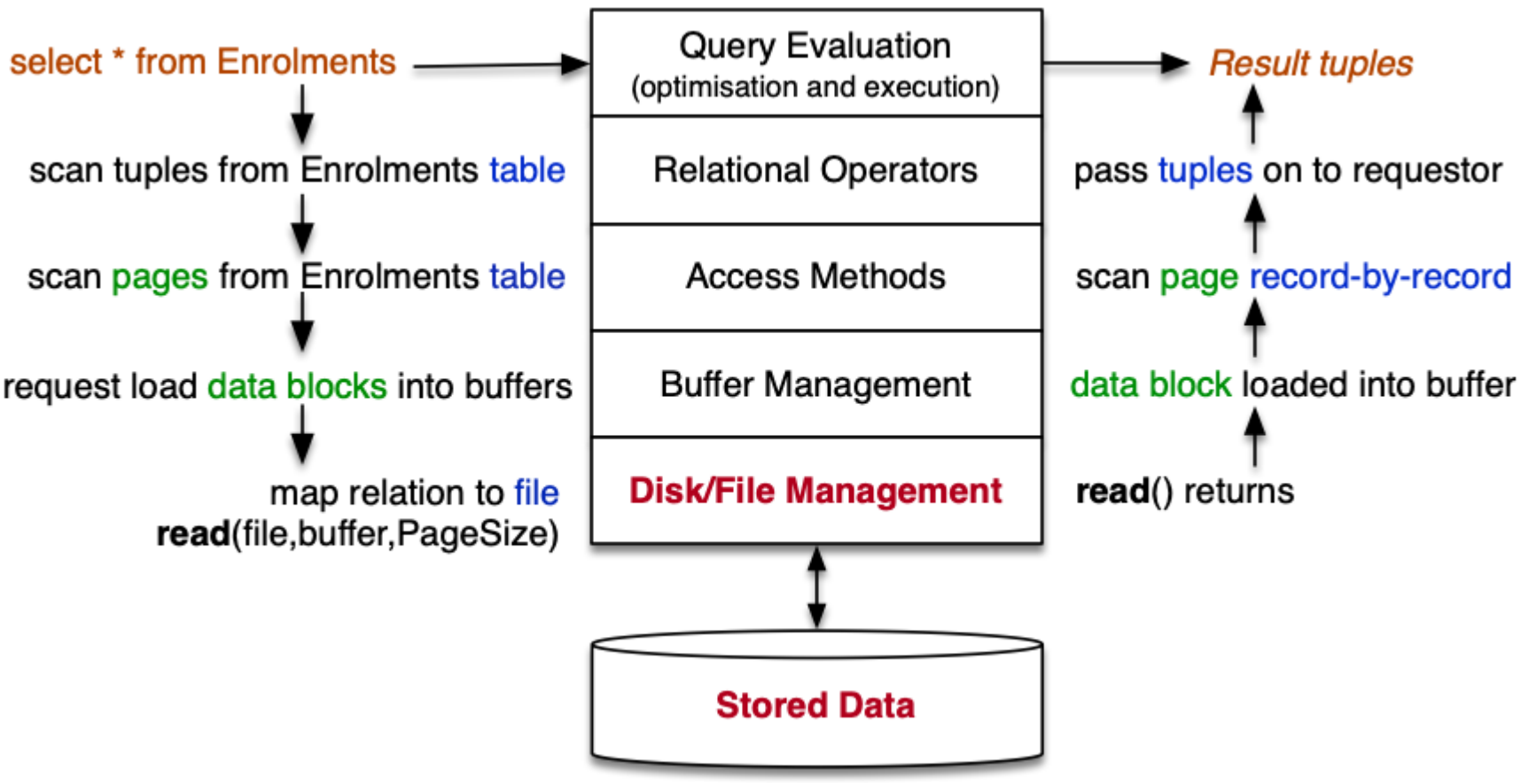
	RAM	**HDD**	**SDD**
Capacity	~ 32GB	~ 8TB	~ 2TB
Cost/byte	~ $10 / GB	~ $40 / TB	~ $200 / TB
Read latency	~ 1µs	~ 10ms	~ 50µs
Write latency	~ 1µs	~ 10ms	~ 900µs
Read unit	byte	block (e.g. 1KB)	byte
Writing	byte	write a block	write on empty block

❖ Storage Technology (cont)

Aims of storage management in DBMS:

- provide view of data as collection of pages/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
- basis for file structures used by access methods

❖ Views of Data in Query Evaluation



❖ Views of Data in Query Evaluation (cont)

Representing database objects during query execution:

- **`DB`** (handle on an authorised/opened database)
- **`Rel`** (handle on an opened relation)
- **`Page`** (memory buffer to hold contents of disk block)
- **`Tuple`** (memory holding data values from one tuple)

Addressing in DBMSs:

- **`PageID = FileID+Offset`** ... identifies a block of data
  - where **`Offset`** gives location of block within file
- **`TupleID = PageID+Index`** ... identifies a single tuple
  - where **`Index`** gives location of tuple within page

❖ Storage Management

Topics in storage management ...

- 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
- DB Object Management (Catalog)
  - how tables/views/functions/types, etc. are represented

❖ Cost Models

Throughout this course, we compare costs of DB operations

Important aspects in determining cost:

- data is always transferred to/from disk as whole blocks (pages)
- cost of manipulating tuples in memory is negligible
- overall cost determined primarily by #data-blocks read/written

Complicating factors in determining costs:

- not all page accesses require disk access (buffer pool)
- tuples typically have variable size (tuples/page ?)

More details later ...

Produced: 29 Jan 2023

Week 1 Exercises

- Exercise 1: Unix File I/O (revision)
- Exercise 2: PostgreSQL files
- PostgreSQL servers
- Exercise 3: Table Statistics
- Exercise 4: Extracting a Schema
- Exercise 5: Enumerated Types
- Relational Algebra (RA)
- Exercise 6: Relational Algebra (RA)

❖ Exercise 1: 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.

❖ Exercise 2: PostgreSQL files

Explore the PostgreSQL **`pg_catalog`** to determine ...

How many tables are in the catalog?

What users? namespaces? tablespaces? are there

What information is stored about tables, attributes, types?

How many tables are in the **`public`** schema?

Which directory contains the database *X* ?

Which files in that directory correspond to table *Y* ?

❖ PostgreSQL servers

Script to create your **`/localstorage/$USER`** is now working

PostgreSQL-14.1 has removed the **`Ready to install.`** message.

`Leaving directory` is not an error.

What the **`Makefile`** does ...

- changes into a directory
- runs the **`Makefile`** in that directory
- leaves that directory
- repeat until all directories are done

❖ PostgreSQL servers (cont)

Some people have been asking (paraphrasing)

- can I just use a pre-compiled binary version of PostgreSQL

That would be ok if we were just using PostgreSQL.

COMP9315 requires you to modify PostgreSQL.

You can't do this if you start from a binary; you need the source code.

❖ Exercise 3: Table Statistics

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
    (tabname text, ntuples int);

create function pop()
    returns setof TableInfo
as $$
...
$$ language plpgsql;
```

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

❖ Exercise 4: Extracting a Schema

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 5: Enumerated Types

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)
```

❖ Relational Algebra (RA)

Relational algebra is the "machine language" of RDBMSs

SQL is translated to RA before being executed

Reminder (I hope) ...

- Select ... selects a subset of tuples based on a condition
- Project ... selects a subset of fields based on a list
- Join ... "merges" two tables accoring to a condition

❖ Exercise 6: Relational Algebra (RA)

Translate each of the following SQL statements to RA

- `select * from R`
- `select a,b from R`
- `select * from R where a > 5`
- `select * from R join S on R.a = S.y`

Assume a schema: `R(a,b,c), S(x,y)`

Indicate: the fields and # tuples in the result

Produced: 1 Feb 2023