

Query Evaluation

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

❖ DBMS Architecture

COMP3311 is not a course on DBMS Architecture (that's COMP9315)

But knowing just a little about how DBMSs work can help

- to avoid/fix inefficiencies in database applications

DBMSs attempt to handle this issue in modules for ...

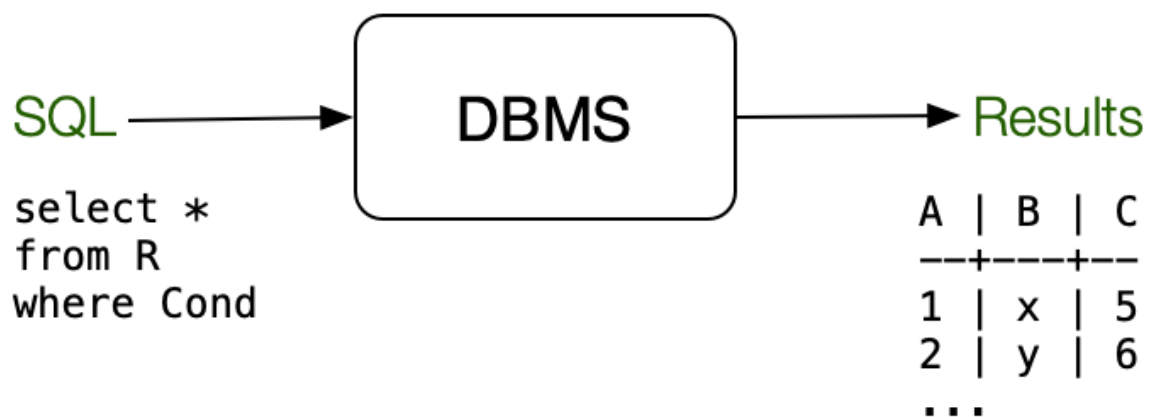
- query processing (QP) .. methods for evaluating queries

As a programmer, you cede a lot of control to the DBMS, but can

- use QP knowledge to make DB applications **efficient**

❖ DBMS Architecture (cont)

Our view of the DBMS so far ...




A machine to process SQL queries.

❖ DBMS Architecture (cont)

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

Machine code for such a machine:

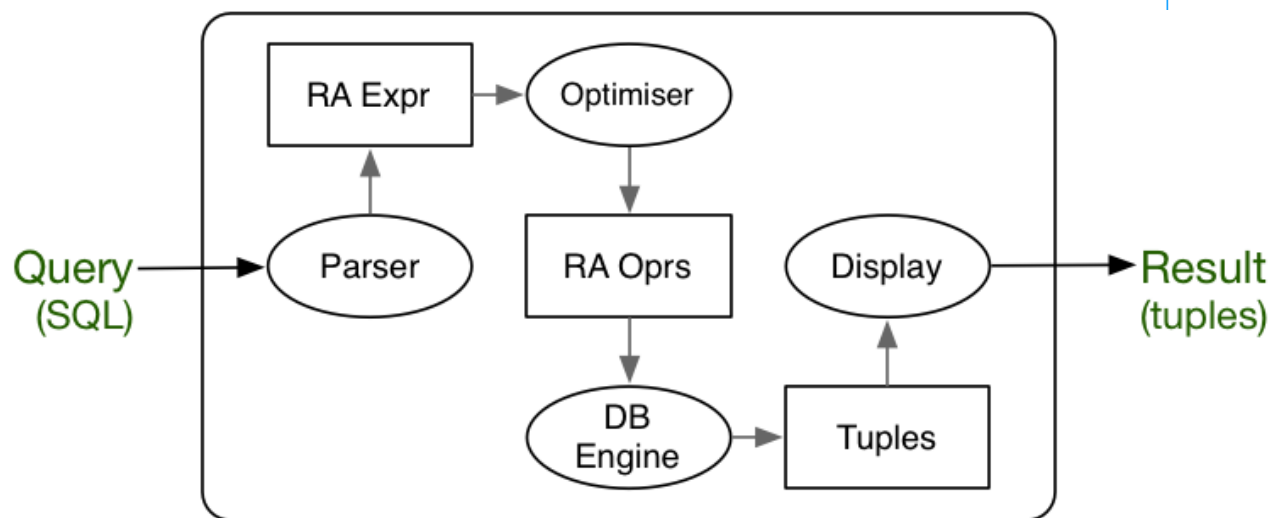
selection (σ)	projection (π)	join ( , \times)
union (\cup)	intersection (\cap)	difference ($-$)
sort	insert	delete

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:



COMP3311 21T1 ♦ Query Evaluation ♦ [4/12]

❖ 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)

An **execution plan** is a sequence of relational operations.

Consider execution plans for: $\sigma_c(R \bowtie_d S \bowtie_e T)$

```
tmp1    := hash_join[d] (R, S)
tmp2    := sort_merge_join[e] (tmp1, T)
result  := binary_search[c] (tmp2)
```

or

```
tmp1    := sort_merge_join[e] (S, T)
tmp2    := hash_join[d] (R, tmp1)
result  := linear_search[c] (tmp2)
```

or

```
tmp1    := btree_search[c] (R)
tmp2    := hash_join[d] (tmp1, S)
result  := sort_merge_join[e] (tmp2)
```

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 $O(\log N)$, 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 $O(N \log N + M \log M)$)
- hash-join (best case cost $O(N+M.N/B)$, with B memory buffers)

❖ Query Optimisation

What is the "best" method for evaluating a query?

Generally, *best* = lowest cost = fastest evaluation time

Cost is measured in terms of pages read/written

- data is stored in fixed-size blocks (e.g. 4KB)
- data transferred disk ↔ memory in whole blocks
- cost of disk ↔ memory transfer is highest cost in system
- processing in memory is very fast compared to I/O

❖ Query Optimisation (cont)

A DBMS **query optimiser** works as follows:

Input: relational algebra expression

Output: execution plan (sequence of RA ops)

```
bestCost = INF; bestPlan = none
while (more possible plans) {
    plan = produce a new evaluation plan
    cost = estimated_cost(plan)
    if (cost < bestCost) {
        bestCost = cost; bestPlan = plan
    }
}
return bestPlan
```

Typically, there are very many possible plans

- smarter optimisers generate likely subset of possible plans

Produced: 25 Mar 2021