>>

Week 8 Exercises

^ >>

❖ Week 08

Things to note ...

- Quiz 4 ... due 9pm Friday 8 April (Friday this week)
- Assignment 1 ... auto-marking output available
  (via Assignment 1 - Submission > Collect Submission)
- Assignment 2 ... due 9pm Friday 15 April (Friday next week)

Topics for this week:

- query evaluation: translation, optimisation, execution

Important: Exam has moved to afternoon of Thu 12 May

<< ^ >>

❖ Assignment 1

Some stats:

- 350 submissions
- 200 people scored full marks (incl. unseen tests)
- 80 more people scored ≥ 14 marks
- 30 people scored zero (will be investigated)

Auto-marking was run on **`vxdb`**; did you test there?

Working our way through all the queries on marks.

Haven't yet run plagiarism checking

<< ^ >>

❖ Assignment 2

There are many "correct" variations to implementing MALH

- so we can't run simple **`diff`** matching for tests like Ass1

We *can* check the critical features of any correct solution

- we can test that you return the correct results from a query
- we can analyse the file structure to see if the file has grown
- we can check whether each tuple is in the correct bucket

You can check all of these yourself before submitting.

First point: use **`grep`** and **`select`**; second point: use **`stats`**; third point: you could modify **`dump`**

<< ^ >>

❖ Assignment 2 (cont)

Debugging your code ...

- can use **`od`** to examine binary data files
- can use **`gdb`** to monitor code execution
- can use **`valgrind`** to find memory leaks

Don't know **`gdb`** or **`valgrind`**?

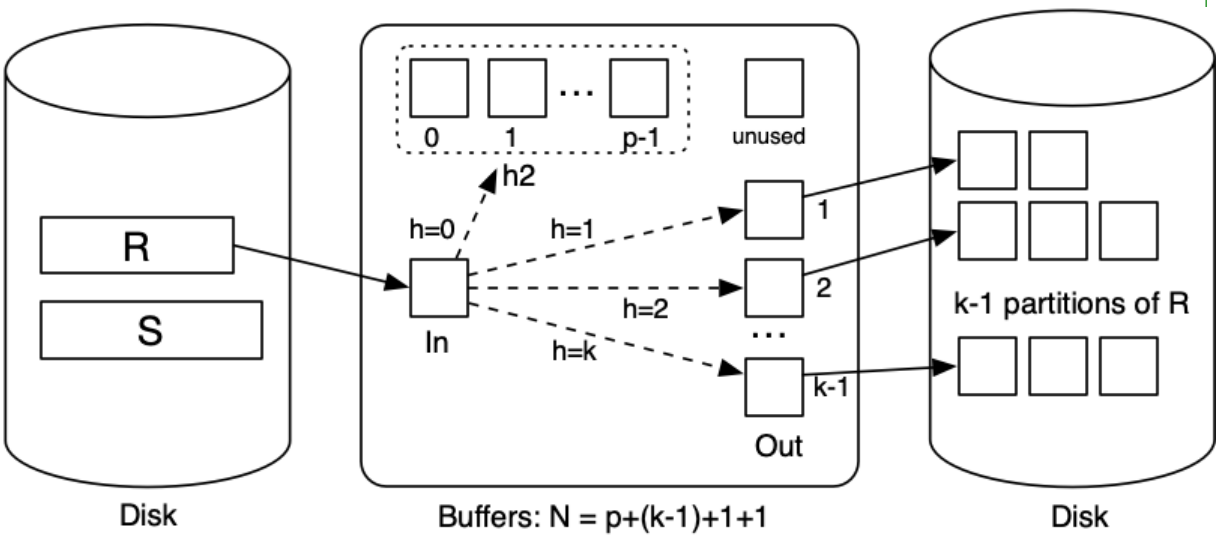
See https://www.cse.unsw.edu.au/~learn/debugging/

<< ^ >>

❖ Hybrid Hash Join

A variant of grace hash join if we have sufficient memory buffers

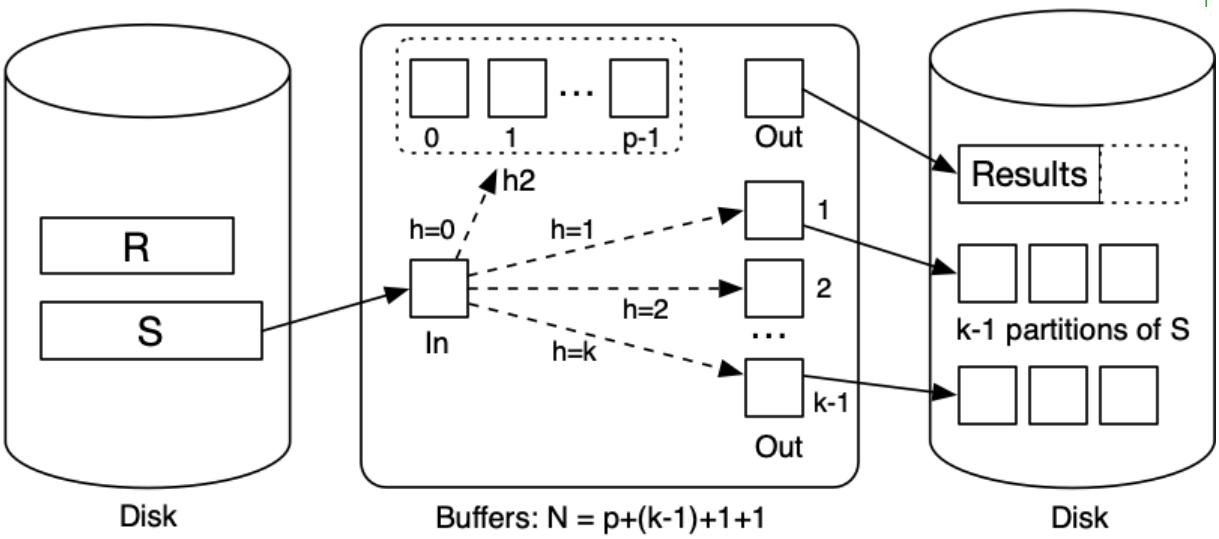
First phase of hybrid hash join (partitioning *R*):



<< ^ >>

❖ Hybrid Hash Join (cont)

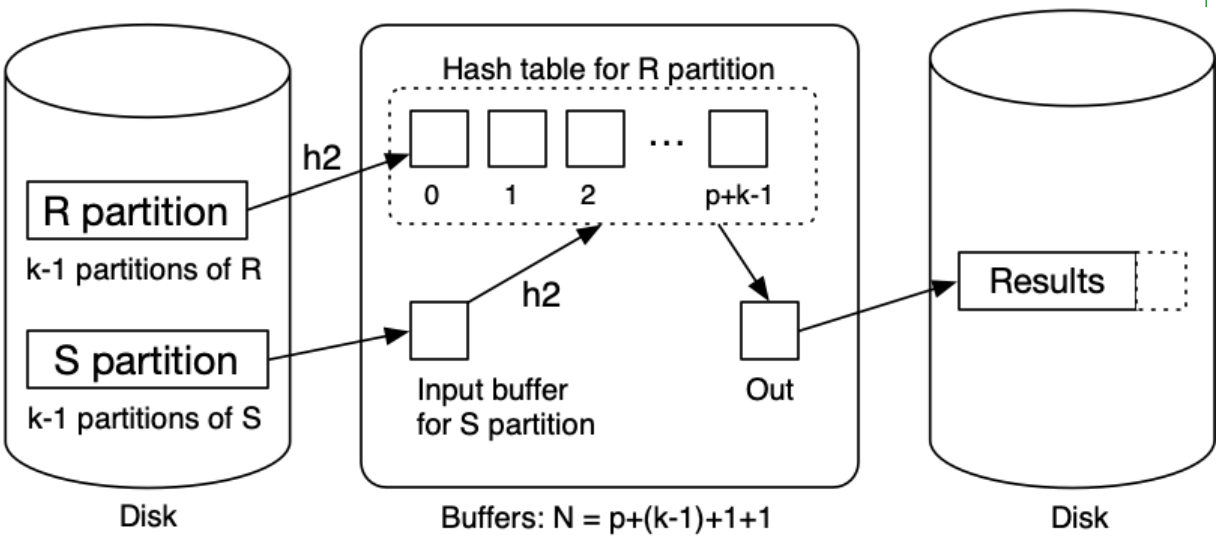
Next phase of hybrid hash join (partitioning *S*):



<< ^ >>

❖ Hybrid Hash Join (cont)

Final phase of hybrid hash join (finishing join):



<< ^ >>

❖ Hybrid Hash Join (cont)

Some observations:

- with *k* partitions, each partition has expected size *ceil(b_R/k)*
- holding 1 partition in memory needs *ceil(b_R/k)* buffers
- trade-off between in-memory partition space and #partitions

Other notes:

- if *N = b_R+2*, using block nested loop join is simpler
- cost depends on *N* (but less than grace hash join)

For *k* partitions, one memory partition: Cost ≅ (3-1/k).(b_R+b_S)

<< ^ >>

❖ Exercise: Hybrid Hash Join Cost

Consider executing *Join[i=j](R,S)* with the following parameters:

- $r_R = 1000$, $b_R = 50$, $r_S = 3000$, $b_S = 150$, $c_{Res} = 30$
- *R.i* is primary key, each *R* tuple joins with 2 *S* tuples
- DBMS has $N = 42$ buffers available for the join
- data + hash have reasonably uniform distribution

Calculate the cost for evaluating the above join

- using hybrid hash join with various *k*
- compute #pages read/written
- compute #join-condition checks performed
- assume that no *R* partition is larger than 40 pages

<< ^ >>

❖ Exercise: Join Cost Comparison

Consider the cost of each of

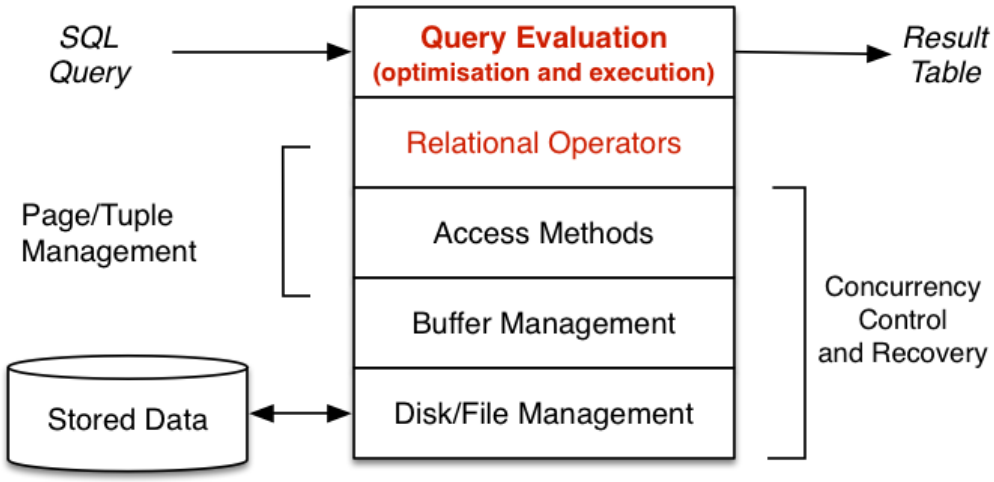
- block nested loop join
- index nested loop join
- sort-merge join
- hash join
- grace hash join
- hybrid hash join

on *Join[i=j](R,S)* from the previous exercises.

Is any one algorithm overall better than the others?

<< ^ >>

❖ Query Processing



<< ^ >>

❖ Query Processing (cont)

A query in SQL:

- states *what* kind of answers are required (declarative)
- does not say *how* they should be computed (!procedural)

A query evaluator/processor :

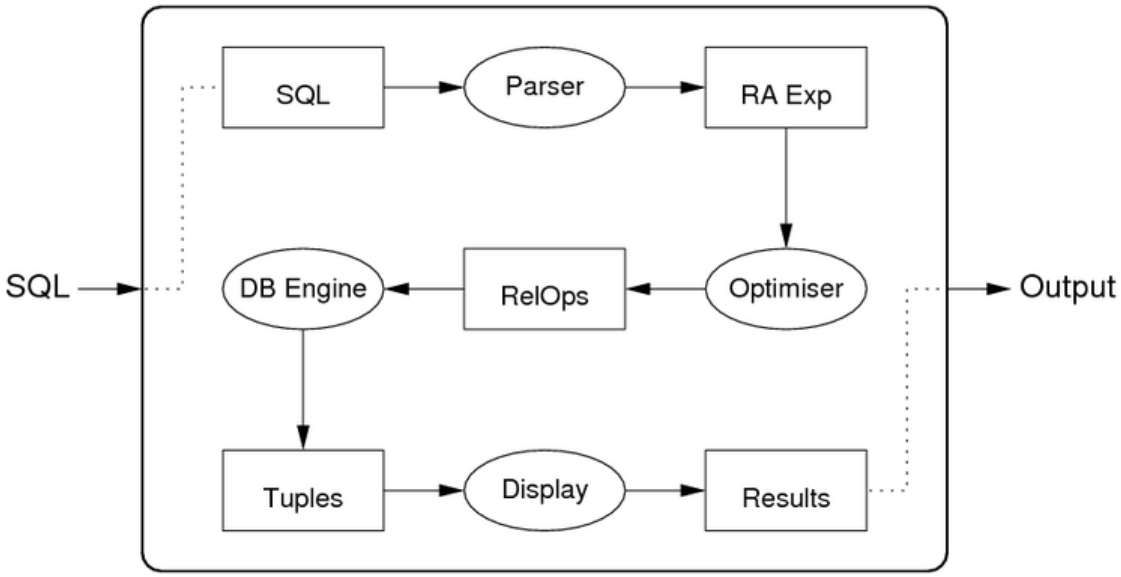
- takes declarative description of query (in SQL)
- parses query to internal representation (relational algebra)
- determines plan for answering query (expressed as DBMS ops)
- executes method via DBMS engine (to produce result tuples)

Some DBMSs can save query plans for later re-use.

<< ^ >>

❖ Query Processing (cont)

Internals of the query evaluation "black-box":



<< ^ >>

❖ Query Translation

Translation step: SQL text → RA expression

Example:

```
SQL: select name from Students where id=7654321;
-- is translated to
RA:  Proj[name](Sel[id=7654321]Students)
```

Processes: lexer/parser, mapping rules, rewriting rules.

Mapping from SQL to RA may include some optimisations, e.g.

```
select * from Students where id = 54321 and age > 50;
-- is translated to
Sel[age>50](Sel[id=54321]Students)
-- rather than ... because of index on id
Sel[id=54321&age>50](Students)
```

<< ^ >>

❖ Exercise: SQL → RelAlg

Convert the following queries into (efficient?) RA expressions

```
select * from R where a > 5;

select * from R where id = 1234 and a > 5;

select R.a from R, S where R.i = S.j;

select R.a from R join S on R.i = S.j;

select * from R, S where R.i = S.j and R.a = 6

select R.a from R, S, T where R.i = S.j and S.k = T.y;
```

Assume **`R.id`** is a primary key and **`R`** is hashed on **`id`**

Assume that there is a B-tree index on **`R.a`**

<< ^ >>

❖ Query Re-writing

Since RA is a well-defined formal system

- there exist many algebraic laws on RA expressions
- which can be used as a basis for expression rewriting
- in order to produce equivalent (more-efficient?) expressions

Expression transformation based on such rules can be used

- to simplify/improve SQL →RA mapping results
- to generate new plan variations to check in query optimisation

<< ^ >>

❖ Relational Algebra Laws

Commutative and Associative Laws:

- $R \bowtie S \leftrightarrow S \bowtie R$, $(R \bowtie S) \bowtie T \leftrightarrow R \bowtie (S \bowtie T)$ (natural join)
- $R \cup S \leftrightarrow S \cup R$, $(R \cup S) \cup T \leftrightarrow R \cup (S \cup T)$
- $R \bowtie_{Cond} S \leftrightarrow S \bowtie_{Cond} R$ (theta join)
- $\sigma_c(\sigma_d(R)) \leftrightarrow \sigma_d(\sigma_c(R))$

Selection splitting (where c and d are conditions):

- $\sigma_{c \wedge d}(R) \leftrightarrow \sigma_c(\sigma_d(R))$
- $\sigma_{c \vee d}(R) \leftrightarrow \sigma_c(R) \cup \sigma_d(R)$

<< ^ >>

❖ Relational Algebra Laws (cont)

Selection pushing ($\sigma_c(R \cup S)$ and $\sigma_c(R \cup S)$):

- $\sigma_c(R \cup S) \leftrightarrow \sigma_c R \cup \sigma_c S, \quad \sigma_c(R \cap S) \leftrightarrow \sigma_c R \cap \sigma_c S$

Selection pushing with join ...

- $\sigma_c (R \bowtie S) \leftrightarrow \sigma_c(R) \bowtie S$ (if c refers only to attributes from R)
- $\sigma_c (R \bowtie S) \leftrightarrow R \bowtie \sigma_c(S)$ (if c refers only to attributes from S)

If *condition* contains attributes from both R and S:

- $\sigma_{c' \wedge c''} (R \bowtie S) \leftrightarrow \sigma_{c'}(R) \bowtie \sigma_{c''}(S)$
- c' contains only R attributes, c'' contains only S attributes

<< ^ >>

❖ Query Optimisation

Convert RA expression into evaluation plan (collection of RelOps)

Cost-based query optimisers

- generate possible RelOp plans
- calculate/estimate cost of each
- choose plan with lowest cost

Note:

- can't generate all possible plans (O(n!))
- can't spend too much time generating plans
  (trade-off between optimisation time and execution time)

<< ^ >>

❖ Query Optimisation (cont)

DBMSs provide several "flavours" of each RA operation.

For example:

- several "versions" of selection (σ) are available
- each version is effective for a particular kind of selection, e.g

```
select * from R where id = 100   -- hashing
select * from S                  -- Btree index
where age > 18 and age < 35
select * from T                  -- MALH file
where a = 1 and b = 'a' and c = 1.4
```

Similarly, π and $\bowtie$ have versions to match specific query types.

<< ^ >>

❖ Query Optimisation (cont)

Example of query translation:

```
select s.name, s.id, e.course, e.mark
from   Student s, Enrolment e
where  e.student = s.id and e.semester = '05s2';
```

maps to

$\pi_{name,id,course,mark}(Stu \bowtie_{e.student=s.id} (\sigma_{semester=05s2} Enr))$

maps to

```
Temp1  = BtreeSelect[semester=05s2](Enr)
Temp2  = HashJoin[e.student=s.id](Stu,Temp1)
Result = Project[name,id,course,mark](Temp2)
```

<< ^ >>

❖ Exercise: Alternative Join Plans

Consider the schema

```
Students(id,name,....)   Enrol(student,course,mark)
Staff(id,name,...)    Courses(id,code,term,lic,...)
```

the following query on this schema

```
select c.code, s.id, s.name
from   Students s, Enrol e, Courses c, Staff f
where  s.id=e.student and e.course=c.id
       and c.lic=f.id and c.term='19T2'
       and f.name='John Shepherd'
```

Show some possible evaluation orders for this query.

<< ^ >>

❖ Exercise: Selection Size Estimation

Assuming that

- all attributes have uniform distribution of data values
- attributes are independent of each other

Give formulae for the number of expected results for

1. `select * from R where not A=k`
2. `select * from R where A=k and B=j`
3. `select * from R where A in (k,l,m,n)`

where *j*, *k*, *l*, *m*, *n* are constants.

Assume: *V(A,R) = 10* and *V(B,R)=100* and *r=1000*

<< ^ >>

❖ Exercise: Selection Size Estimation (ii)

Database stats for static table R(id,X,...)

- r = 5000, c = 50, tuples stored in X order, NULLs first
- V(X,R) = 5, **a,b,c,d,e,NULL** ~ 40:20:10:10:10:10

Estimate the number of result tuples and # pages read

1. `select * from R where X is not null`
2. `select * from R where X = 'a'`
3. `select * from R where X < 'a'`
4. `select * from R where X >= 'c'`
5. `select * from R where X between 'b' and 'd'`

Assume initial search is binary, if needed

<< ^ >>

❖ Exercise: Join Size Estimation

Assume **`S.id`** is a primary key, **`R.s`** is a FK referencing **`S.id`**

How many tuples are in the output from:

1. **`select * from R join S on R.s = S.id`**
2. **`select * from R join S on R.s <> S.id`**
3. **`select * from R join S on R.x = S.y`**
   where **`R.x`** and **`S.y`** have no connection except that *dom(R.x)=dom(S.y)*

Under what conditions will the first query have maximum/minimum size?

<< ^ >>

❖ Week 08

Things to note ...

- Quiz 4 ... due 9pm Friday 8 April (tomorrow!)
- Assignment 1 ... auto-marking output available
  (via Assignment 1 - Submission > Collect Submission)
- Assignment 2 ... due 9pm Friday 15 April (Friday next week)

Topics for this week:

- query evaluation: translation, optimisation, execution

Final Exam: Thursday 12 May, 1pm - 5pm (and cannot be changed)

<< ^ >>

❖ Exercise: Multi-attribute Linear Hashing

Consider the following hash values

```
h(1000) = ...010010101
h(bear) = ...000000100
h(meet) = ...111011000
h(fear) = ...101010101
```

If we have a MALH file with the following parameters:

```
d = 5,  sp = 0,
cv = [(0,0), (1,0), (2,0), (3,0), (0,1), (1,1), (2,1), ...]
```

How many pages are in the data file?

What is the hash value for the tuple **`(1000,bear,meat,fear)`** ?

<< ^ >>

❖ Exercise: Multi-attribute Linear Hashing (cont)

For each of the following queries:

- give the query "hash" as a single bit-string using `*`'s
- represent the query via known and unknow bit-strings
- show which pages will be examined in answering the query

```
?,bear,meat,fear
1000,bear,?,?
1000,?,?,?
?,?,?,fear
?,?,?,?
```

Repeat the above exercises if **`sp = 3`**

<< ^ >>

❖ Query Execution

A query execution plan:

- consists of a collection of RelOps
- executing together to produce a set of result tuples

Results may be passed from one operator to the next:

- materialization ... writing results to disk and reading them back
- pipelining ... generating and passing via memory buffers

<< ^ >>

❖ Exercise: EXPLAIN examples

Consider this database ...

```
CourseEnrolments(student, course, mark, grade, ...)
Courses(id, subject, semester, homepage)
People(id, family, given, title, name, ..., birthday)
ProgramEnrolments(id, student, semester, program, wam, ...)
Students(id, stype)
Subjects(id, code, name, longname, uoc, offeredby, ...)
-- plus many other table
```

with this view

```
create view EnrolmentCounts as
 select s.code, c.semester, count(e.student) as nstudes
   from Courses c join Subjects s on c.subject=s.id
        join Course_enrolments e on e.course = c.id
  group by s.code, c.semester;
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Some database statistics:

tab_name	n_records
courseenrolments	66853
courses	2603
people	5315
programenrolments	24942
students	5314
subjects	1094

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Predict how each of the following queries will be executed ...

1. `select max(birthday) from People`
2. `select max(id) from People`
3. `select family from People order by family;`
4.
```
select distinct p.id, p.name
from People p, CourseEnrolments e
where p.id=e.student and e.grade='FL';
```
5.
```
select * from EnrolmentCounts where
code='COMP9315';
```

Check your prediction using the **`EXPLAIN ANALYZE`** command.

Examine the effect of adding **`ORDER BY`** and **`DISTINCT`**.

Add indexes to improve the speed of slow queries.

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Select on non-indexed attribute

```
uni=# explain
uni=# select * from Students where stype='local';
                       QUERY PLAN
-------------------------------------------------------
 Seq Scan on students
             (cost=0.00..562.01 rows=23544 width=9)
   Filter: ((stype)::text = 'local'::text)
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Select on non-indexed attribute with actual costs

```
uni=# explain analyze
uni=# select * from Students where stype='local';
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on students
             (cost=0.00..562.01 rows=23544 width=9)
             (actual time=0.052..5.792 rows=23551 loops=1)
   Filter: ((stype)::text = 'local'::text)
   Rows Removed by Filter: 7810
 Planning time: 0.075 ms
 Execution time: 6.978 ms
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Select on indexed, unique attribute

```
uni=# explain analyze
uni-# select * from Students where id=100250;
                        QUERY PLAN
-----------------------------------------------------------
 Index Scan using student_pkey on student
           (cost=0.00..8.27 rows=1 width=9)
           (actual time=0.049..0.049 rows=0 loops=1)
   Index Cond: (id = 100250)
 Planning Time: 0.274 ms
 Execution Time: 0.109 ms
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Select on indexed, unique attribute

```
uni=# explain analyze
uni-# select * from Students where id=1216988;
                        QUERY PLAN
-----------------------------------------------------------
 Index Scan using students_pkey on students
           (cost=0.29..8.30 rows=1 width=9)
           (actual time=0.011..0.012 rows=1 loops=1)
   Index Cond: (id = 1216988)
 Planning time: 0.273 ms
 Execution time: 0.115 ms
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Join on a primary key (indexed) attribute (2016)

```
uni=# explain analyze
uni-# select s.id,p.name
uni-# from Students s, People p where s.id=p.id;
                        QUERY PLAN
-----------------------------------------------------------
Hash Join (cost=988.58..3112.76 rows=31048 width=19)
          (actual time=11.504..39.478 rows=31048 loops=1)
  Hash Cond: (p.id = s.id)
  -> Seq Scan on people p
         (cost=0.00..989.97 rows=36497 width=19)
         (actual time=0.016..8.312 rows=36497 loops=1)
  -> Hash (cost=478.48..478.48 rows=31048 width=4)
          (actual time=10.532..10.532 rows=31048 loops=1)
          Buckets: 4096  Batches: 2  Memory Usage: 548kB
       ->  Seq Scan on students s
               (cost=0.00..478.48 rows=31048 width=4)
               (actual time=0.005..4.630 rows=31048 loops=1)
 Planning Time: 0.691 ms
 Execution Time: 44.842 ms
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Join on a primary key (indexed) attribute (2018)

```
uni=# explain analyze
uni-# select s.id,p.name
uni-# from Students s, People p where s.id=p.id;
                        QUERY PLAN
-----------------------------------------------------------
Merge Join  (cost=0.58..2829.25 rows=31361 width=18)
            (actual time=0.044..25.883 rows=31361 loops=1)
  Merge Cond: (s.id = p.id)
  ->  Index Only Scan using students_pkey on students s
            (cost=0.29..995.70 rows=31361 width=4)
            (actual time=0.033..6.195 rows=31361 loops=1)
        Heap Fetches: 31361
  ->  Index Scan using people_pkey on people p
            (cost=0.29..2434.49 rows=55767 width=18)
            (actual time=0.006..6.662 rows=31361 loops=1)
Planning time: 0.259 ms
Execution time: 27.327 ms
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Join on a non-indexed attribute (2016)

```
uni=# explain analyze
uni=# select s1.code, s2.code
uni-# from Subjects s1, Subjects s2
uni=# where s1.offeredBy=s2.offeredBy;
                          QUERY PLAN
-------------------------------------------------------------------
Merge Join (cost=4449.13..121322.06 rows=7785262 width=18)
           (actual time=29.787..2377.707 rows=8039979 loops=1)
 Merge Cond: (s1.offeredby = s2.offeredby)
 ->  Sort (cost=2224.57..2271.56 rows=18799 width=13)
          (actual time=14.251..18.703 rows=18570 loops=1)
     Sort Key: s1.offeredby
     Sort Method: external merge  Disk: 472kB
     ->  Seq Scan on subjects s1
             (cost=0.00..889.99 rows=18799 width=13)
             (actual time=0.005..4.542 rows=18799 loops=1)
 ->  Sort (cost=2224.57..2271.56 rows=18799 width=13)
          (actual time=15.532..1100.396 rows=8039980 loops=1)
     Sort Key: s2.offeredby
     Sort Method: external sort  Disk: 552kB
     ->  Seq Scan on subjects s2
             (cost=0.00..889.99 rows=18799 width=13)
             (actual time=0.002..3.579 rows=18799 loops=1)
Total runtime: 2767.1 ms
```

<< ^ >>

❖ Exercise: EXPLAIN examples (cont)

Example: Join on a non-indexed attribute (2018)

```
uni=# explain analyze
uni=# select s1.code, s2.code
uni-# from Subjects s1, Subjects s2
uni-# where s1.offeredBy = s2.offeredBy;
                          QUERY PLAN
-------------------------------------------------------------------
Hash Join  (cost=1286.03..108351.87 rows=7113299 width=18)
           (actual time=8.966..903.441 rows=7328594 loops=1)
  Hash Cond: (s1.offeredby = s2.offeredby)
  ->  Seq Scan on subjects s1
         (cost=0.00..1063.79 rows=17779 width=13)
         (actual time=0.013..2.861 rows=17779 loops=1)
  ->  Hash  (cost=1063.79..1063.79 rows=17779 width=13)
            (actual time=8.667..8.667 rows=17720 loops=1)
        Buckets: 32768  Batches: 1  Memory Usage: 1087kB
        ->  Seq Scan on subjects s2
               (cost=0.00..1063.79 rows=17779 width=13)
               (actual time=0.009..4.677 rows=17779 loops=1)
Planning time: 0.255 ms
Execution time: 1191.023 ms
```

<< ^

❖ Exercise: EXPLAIN examples (cont)

Example: Join on a non-indexed attribute (2018)

```
uni=# explain analyze
uni=# select s1.code, s2.code
uni-# from Subjects s1, Subjects s2
uni-# where s1.offeredBy = s2.offeredBy and s1.code < s2.code;
                          QUERY PLAN
-------------------------------------------------------------------
Hash Join  (cost=1286.03..126135.12 rows=2371100 width=18)
           (actual time=7.356..6806.042 rows=3655437 loops=1)
  Hash Cond: (s1.offeredby = s2.offeredby)
  Join Filter: (s1.code < s2.code)
  Rows Removed by Join Filter: 3673157
  ->  Seq Scan on subjects s1
          (cost=0.00..1063.79 rows=17779 width=13)
          (actual time=0.009..4.602 rows=17779 loops=1)
  ->  Hash  (cost=1063.79..1063.79 rows=17779 width=13)
            (actual time=7.301..7.301 rows=17720 loops=1)
        Buckets: 32768  Batches: 1  Memory Usage: 1087kB
        ->  Seq Scan on subjects s2
                (cost=0.00..1063.79 rows=17779 width=13)
                (actual time=0.005..4.452 rows=17779 loops=1)
Planning time: 0.159 ms
Execution time: 6949.167 ms
```

Produced: 7 Apr 2022