>>

Week 9 Online Sessions

^ >>

❖ Assignment 2: T.I.Y.

How to investigate whether your program is "working"

- use **`grep`** to check that you're returning correct tuples
- use **`./stats`** to find out *b*, *r*, *c*, #sigs, etc.
- examine **Query Stats** after tuples to check plausibility

```
Query Stats:
# sig pages read:     A
# signatures read:    B
# data pages read:    C
# tuples examined:    D
# false match pages:  E
```

A,B should be consistent with stats; you do *not* have to match C,D,E exactly

<< ^ >>

❖ Things to Note

Things to note for this week and next week and ...

- Assignment 2 due before Monday 19 April at 11am
- Quiz 5 due before Friday 23 April at 9pm
- MyExperience due before April 29 at midnight
- Real lectures next week: Zoom Tue 2-4, Thu 2-4
  - Oliver Tan (CSE, Facebook, Dropbox, ...) on DBMS Trends
- Sample exam coming soon; Final exam Mon 3 May afternoon (AEST)

<< ^ >>

❖ Exercise: Explaining EXPLAIN

Examine the **`explain`** output from the following queries:

```
select max(birthday) from people;

select max(id) from People;

select family from People order by family;

select distinct p.id,p.name
from  People p, CourseEnrolments e
where p.id=e.student and e.grade='FL';

select * from EnrolmentCounts where code='COMP9315';
```

<< ^ >>

❖ Exercise: Update Anomaly

Consider the following transaction (expressed in pseudo-code):

```
-- Accounts(id,owner,balance,...)
transfer(src id, dest id, amount int)
{
   -- R(X)
   select balance from Accounts where id = src;
   if (balance >= amount) {
      -- R(X),W(X)
      update Accounts set balance = balance-amount
      where id = src;
      -- R(Y),W(Y)
      update Accounts set balance = balance+amount
      where id = dest;
}  }
```

If two transfers occur on this account simultaneously,
give a schedule that illustrates the "dirty read" phenomenon.

<< ^ >>

❖ Exercise: How many Schedules?

In the previous exercise, we looked at several schedules

For a given set of tx's T_1 ... T_n

- how many serial schedules are there?
- how many total schedules are there?

<< ^ >>

❖ Exercise: Recoverability/Serializability

Recoverability and **S**erializability are orthogonal, i.e.

- a schedule can be R & S, !R & S, R &!S, !R & !S

Consider the two transactions:

```
T1:   W(A)   W(B)   C
T2:   W(A)   R(B)   C
```

Give examples of schedules on T1 and T2 that are

- recoverable and serializable
- not recoverable and serializable
- recoverable and not serializable

<< ^ >>

❖ Exercise: Serializability Checking

Is the following schedule view/conflict serializable?

```
T1:         W(B)  W(A)
T2:  R(B)                   W(A)
T3:                   R(A)        W(A)
```

Is the following schedule view/conflict serializable?

```
T1:         W(B)  W(A)
T2:  R(B)               W(A)
T3:                           R(A)  W(A)
```

<< ^ >>

❖ Exercise: Deadlock Handling

Consider the following schedule on four transactions:

```
T1:  R(A)          W(C)                                        W(D)
T2:        R(B)                              W(C)
T3:                     R(D)          W(B)
T4:                           R(E)                    W(A)
```

Assume that: each **R** acquires a shared lock; each **W** uses an exclusive lock; two-phase locking is used.

Show how the wait-for graph for the locks evolves.

Show how any deadlocks might be resolved via this graph.

<< ^

❖ Exercise: Locking in PostgreSQL

How could we solve this problem via locking?

```
create or replace function
    allocSeat(paxID int, fltID int, seat text)
    returns boolean
as $$
declare
    pid int;
begin
    select paxID into pid from SeatingAlloc
    where  flightID = fltID and seatNum = seat;
    if (pid is not null) then
        return false;  -- someone else already has seat
    else
        update SeatingAlloc set pax = paxID
        where  flightID = fltID and seatNum = seat;
        commit;
        return true;
    end if;
end;
$$ langauge plpgsql;
```

Produced: 15 Apr 2021