Interactive - 34 - explain

First you have to understand that different queries will work and perform in different ways. The biggest thing is the query hitting an index. That is not the only thing.

```
select
          t1.homework_id
        , t1.homework_no
        , case
            when t3.pts = 0 then 'n'
            when t3.pts is null then 'n'
            else 'y'
          end as "has_been_seen"
        , t1.homework_no::int as i_homework_no
        , coalesce(t3.tries,NULL,0,t3.tries) as tries
    from ct_homework as t1
        left outer join ct_homework_seen as t2 on ( t1.homework_id = t2.homework_id )
        left outer join ct_homework_grade as t3 on ( t1.homework_id = t3.homework_id )
    where exists (
            select 1 as "found"
            from ct_login as t3
            where t3.user_id = '7a955820-050a-405c-7e30-310da8152b6d'
        )
     and ( t3.user_id = '7a955820-050a-405c-7e30-310da8152b6d'
        or t3.user_id is null )
    order by 12 asc
;
```

v.s.

```
select
          t1.homework_id
        , t1.homework_no
        , case
            when t3.pts = 0 then 'n'
            when t3.pts is null then 'n'
            else 'y'
          end as "has_been_seen"
        , t1.homework_no::int as i_homework_no
        , coalesce(t3.tries,NULL,0,t3.tries) as tries
    from ct_homework as t1
        left outer join ct_homework_seen as t2 on ( t1.homework_id = t2.homework_id )
        left outer join ct_homework_grade as t3 on ( t1.homework_id = t3.homework_id
            and t3.user_id = '7a955820-050a-405c-7e30-310da8152b6d' )
    where exists (
            select 1 as "found"
```

```
            from ct_login as t3
            where t3.user_id = '7a955820-050a-405c-7e30-310da8152b6d'
        )
     order by 4 asc
;
```

both produce the same results. The first one will construct the set of data from the join and then filter it for the user. An index will not be used because of the `or` and the `null`.

The 2nd one pre-filters the table `ct_homework_graded` for just the single user. Since this is an equal condition an index will be used and this smaller set of data then is then used in the join.

First setup the tables and data for this by running `hw34_5.sql`.

A better implementation of the 2nd form is (run file `hw34_3.sql`):

```
explain analyze with per_user_ct_homework_grade as (
        select *
        from ct_homework_grade as t4
        where t4.user_id ='7a955820-050a-405c-7e30-310da8152b6d'
    )
select
          t1.homework_id
        , t1.homework_no
        , case
            when t3.pts = 0 then 'n'
            when t3.pts is null then 'n'
            else 'y'
          end as "has_been_seen"
        , t1.homework_no::int as i_homework_no
        , coalesce(t3.tries,NULL,0,t3.tries) as tries
    from ct_homework as t1
        left outer join ct_homework_seen as t2 on ( t1.homework_id = t2.homework_id )
        left outer join per_user_ct_homework_grade as t3 on ( t1.homework_id = t3.homework_i
     order by 4 asc
;
```

This removes the now unnecessary query to check that it is a valid user id (on ct_login) and uses a `with` to set up the per-user set of data.

The question is now how to figure out what the query planner is doing and what queries need to be optimized.

Validate: SQL-Select,"select 'PASS' as x"

FilesToRun: hw34_5.sql

FilesToRun: hw34_3.sql