CS307
MidTerm Review

With answers and some comments.

Design

Found on a forum:

I have a problem, and after having thought about it and done some research, I haven't really found an answer. I am trying to create a database storing the highest scores of a mini-game with hundreds of levels.
I already have a table that contains the titles and the identifiers of the levels, as well as a table that contains the names and identifiers of players. I am therefore trying to create a table with as many rows as players and as many fields as levels.
With Symfony (a PHP development framework) *to create tables you need register fields by hand; except that with as many levels, work is going to be long. Is there a way to create a base from two others in a simple way and then access it easily?*

What would you answer?

The question asked wasn't about design, but my answer would have been. This guy wants a table with one row per user (OK) but one "field" (= attribute or column) per levels − of which he says he has hundreds! You definitely don't want tables with hundreds of columns. The correct way of doing is to have one table with (userid, levelid, score) which makes it easy to compute the highest score per level, the sum of the scores per user − and to add new levels when needed. To say nothing that you don't need to store "null" when a user hasn't yet completed a level − there are simply no rows.

Bike Sharing

How would you design a database?

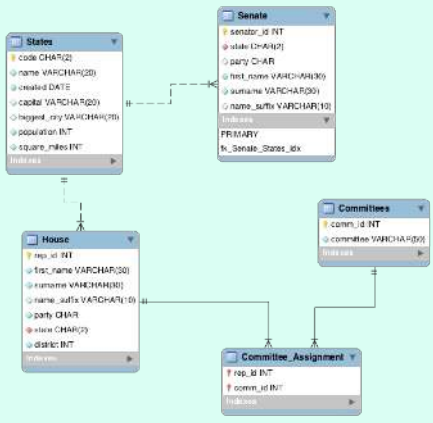
This is of course a wide, open question. Bikes can only be identified by a sequential number, because it's hard to tell a bike from another. However, I guess that companies must keep track of suppliers of bikes (and assess quality on the number of repairs), of when bikes where introduced in the circuit (which tells how old they are), of their current position, and possibly (although that would be redundant information) of who last used them. They may be available or being repaired or removed from sharing (broken beyond repair or lost). There may also be some technological information about the "generation" (several types of locks or models of bikes). You must probably track repairs for each bike (bike id, date, what was to be repaired), which lets you see problems with the quality of bikes provided by one supplier. For users, you probaby have a numerical id and use the phone number as unique identifier (plus whatever you can collect). Perhaps you want to be able to black-list a user. Trips associate a bike, a user, a point and time of departure and arrival. Bikes that haven't moved (or not much) in a while are probably broken and should be collected.

Queries

http://edu.konagora.com/exam

Access Code: KONAG9198

Password: jeffers0n



Write a query that returns the number of members of each party in the Congress, by house (Senate/House of Representatives) and by party.

```
SELECT x1.party, x1.numInSenate, x2.numInHouse,
       x1.numInSenate + x2.numInHouse as totalInParty
FROM (SELECT party, COUNT(*) as numInSenate
      FROM senate
      GROUP BY party) x1
     JOIN (SELECT party, COUNT(*) as numInHouse
           FROM house
           GROUP BY party) x2
       on x1.party = x2.party
```

May answer the question, but assumes that you have the same number of parties in both chambers ("Chamber" refers to either Senate or House of Representatives). It wasn't the case in the exam database (there were no independent folks in one of the chambers). So it doesn't always answer the question. It's relatively efficient otherwise.

```
SELECT "House republican", count(*) AS "count"
FROM house WHERE party = 'R'
UNION
SELECT "House democrat", count(*) AS "count"
FROM house WHERE party = 'D'
UNION
SELECT "House independent", count(*) AS "count"
FROM house WHERE party = 'I'
UNION
SELECT "Senate republican", count(*) AS "count"
FROM senate WHERE party ='R'
UNION
SELECT "Senate democrat", count(*) AS "count"
FROM senate WHERE party = 'D'
UNION
SELECT "Senate independent", count(*) AS "count"
FROM senate WHERE party = 'I'
```

This one works perfectly well, but isn't very efficient – multiple passes over each table. UNION should be UNION ALL (there cannot be duplicates)

```
select count(s.party) as "Senate Political Party" ,
       h.party as "House Political Party"
from senate s
     join house h
```

Completely off. No join condition. No GROUP BY. Not even sure that it runs (it would definitely not run on most database management systems).

```
SELECT hr.house_rep, hd.house_dem,
       hi.house_ind, sr.senate_rep,
       sd.senate_dem, si.senate_ind
FROM (SELECT count(*) house_rep FROM house
      WHERE party = 'R') hr,
     (SELECT count(*) house_dem FROM house
      WHERE party = 'D') hd,
     (SELECT count(*) house_ind FROM house
      WHERE party = 'I') hi,
     (SELECT count(*) senate_rep FROM senate
      WHERE party = 'R') sr,
     (SELECT count(*) senate_dem FROM senate
      WHERE party = 'D') sd,
     (SELECT count(*) senate_ind FROM senate
      WHERE party = 'I') si
```

Returns everything on one row, but otherwise returns a correct result. As in a previous case, doing one aggregate per party and chamber isn't very efficient.

```
select party, count(*)
from house
group by party
union
select party, count(*)
from senate
group by party
```

Only one aggregation per table, which is good, but doesn't quite answer the question as it doesn't return the name of the chamber ('Senate' or 'House of Representatives'). UNION could also potentially make lines disappear – if you have 3 independents in each house, there would be two lines displaying the letter I and 3, and UNION would remove one. It would have been correct with a string constants saying 'House' for the first query and 'Senate' for the second one – and a UNION ALL. Note that if there are no independents in one chamber there is no line for them (some other proposed solutions return zero)

```
select house,
       case party
          when 'D' then 'Democrats'
          when 'R' then 'Republicans'
          else 'Independents'
       end as party,
       num
from (select 'Senate' as house,
             party,
             count(*) as num
      from senate
      group by party
      union all
      select 'House of Representatives' as house,
             party,
             count(*) as num
      from house
      group by party)
order by house, party
```

My own answer would probably have been a corrected version of the previous one, wrapped up in another query to display the party name in clear.

What are the names of the states (ordered by name) with a single representative in the House of Representatives?

```
select s.code, s.name
from states s
     join house h
where s.code = h.state
group by s.name
having count(*) = 1
```

The join condition is in the WHERE, which seems to be accepted by SQLite but may not be by other products for which it's either `from A, B where` *`join_condition`* or `from A join B on` *`join_condition`*, but not a mix of the two. Sqlite, like MySQL, also allows not to have the code in the GROUP BY but other products wouldn't. The result is nevertheless correct because both names and codes are unique and grouping by one or both is the same. The result is also sorted by name as a side-effect of GROUP BY, but the order can only be guaranteed with an explicit ORDER BY. Basically, good on SQLite, not necessarily so good elsewhere but the student couldn't really know.

```
SELECT state
FROM house
GROUP BY state
HAVING COUNT(state) = 1
```

Table house only contains state codes, not state names, so the query doesn't answer the question. But it was reasonably close – it just needed a join on the state table to retrieve the name, and ordering by name.

```
select x.state
from (select count(state) as amt, state
      from house
      group by state
      order by count()) x
where x.amt = 1
```

There is a more complicated version of the previous answer; it doesn't return names but codes (and in this case codes are wrongly ordered). The condition over the subquery is the same as HAVING, the ORDER BY COUNT() is a weird syntax that wouldn't work in mots contexts and its completely useless anyway.

```
select name
from states,
     (select state, count(rep_id) as num
      from house
      group by state
      having num=1) x
where code=state
```

Would have been perfect with the ORDER BY that was requested.

```
SELECT state FROM house
WHERE district = 0
ORDER BY state
```

This was cleverly using the fact that when there is a single representative, his or her district is numbered zero. Like many of the other queries, it lacks the join on table state to retrieve the name (it lists ordered codes).

```
select s.name as one_rep_state
from states s
     join house h
     on h.state = s.code
where h.district = 0
order by s.name
```

This is the previous query corrected.

```
select s.name as one_rep_state
from states s
     join (select state
           from house
           group by state
           having count(*) = 1) h
     on h.state = s.code
order by s.name
```

Or it was possible not to rely on the value assigned to the district when there is only one (it might change and simply be 1 one day) and count.

Who participates in the greatest number of House Committees ?

That was the most difficult question.

```
select hou.first_name, hou.surname, c.committee_count
from (select b.rep_id,
             b.committee_count
      from (select rep_id,
                   count(comm_id) committee_count
            from committee_assignment
            group by rep_id) b) c
    inner join house hou
       on hou.rep_id = c.rep_id
group by hou.surname
having count(c.committee_count) = 4
order by c.committee_count desc
```

Good start, with a correct counting of committes per rep_id and a join with house on rep_id. I don't understand why the student suddenly had the idea of grouping by surname, which is completely wrong as some surnames are common. The GROUP BY should be on everything that isn't aggregated, and other queries had told the student that the greatest value for committe count was 4, which can change any time.

```
select x.first_name
from (select count(*) num, h.first_name
     from committee_assignment c
          join house h
            on h.rep_id = c.rep_id
     group by h.first_name) x
where x.num = (select max(num)
               from (select count(*) num,
                            h.first_name
                     from committee_assignment c
                          join house h
                            on h.rep_id = c.rep_id
                     group by h.first_name))
```

Grouping by surname was bad, grouping by first_name is an even worse idea in the US, where the Johns, Peters and Bills are numerous. It was very close to be a perfect answer: it should have been grouped by rep_id, and of course the surname (and possibly the suffix, 'Jr.' or 'Sr.', less important) should have been returned, not only the first name. The wrong initial idea was probably grouping on a join, which was unnecessary here.

```
select first_name, surname
from (select max(c) m
      from (select rep_id, count(rep_id) c
            from committee_assignment
            group by rep_id))
    join (select rep_id, count(rep_id) c
          from committee_assignment
          group by rep_id) r
      on c = m
    join house h
      on r.rep_id = h.rep_id
```

This one gives the correct answer, and will still give the correct answer when all committees have different members.

```
SELECT h.first_name, h.surname, count(*)
FROM house h,
     committee_assignment a
WHERE h.rep_id = a.rep_id
GROUP BY h.rep_id
ORDER BY count(*) DESC
```

That was a lazy solution. The guy(s) in the most committees will indeed appear at the top of the result, but anybody in a committee will appear as well. So it doesn't really answer the question.

```
SELECT *,
       (SELECT COUNT(*)
        FROM committee_assignment a
        WHERE a.rep_id = h.rep_id) AS committee_count
FROM house h
WHERE committee_count =
(SELECT COUNT(*) AS committee_count
 FROM committee_assignment
 GROUP BY rep_id
 ORDER BY committee_count desc
 LIMIT 1)
```

Here the ORDER BY ... LIMIT 1 in the subquery is perfectly correct for finding the maximum number of committees; there cannot be a tie problem. Refering in the WHERE clause to an alias defined in the SELECT only works here because it's a correlated subquery (the student couldn't know, and it would have been easy to wrap it into another query).The query perfectly answers the question (it got full mark) but isn't very efficient (computes the number of committees for every row in table house, including people not in a committee)

```
select rep_id,
       count(*) as num_of_committees
from committee_assignment
group by rep_id
```

Starting point (that most students had right)

```
select rep_id,
       count(*) as num_of_committees
from committee_assignment
group by rep_id
having count(*) =
      (select max(num_of_committees)
       from (select rep_id,
                    count(*) as num_of_committees
             from committee_assignment
             group by rep_id) x)
```

This returns the identifier of the most active congressman.

```
select h.first_name, h.surname,
       s.name as state, z.num_of_committees
from (select rep_id,
             count(*) as num_of_committees
      from committee_assignment
      group by rep_id
      having count(*) =
            (select max(num_of_committees)
             from (select rep_id,
                          count(*) as num_of_committees
                   from committee_assignment
                   group by rep_id) x)) z
      join house h
        on h.rep_id = z.rep_id
      join states s
        on s.code = h.state
order by h.surname, h.first_name, s.name
```

A few joins and it's OK, with repeating bits that (on most database systems) can be factorized.

```
with comm as (select rep_id,
                     count(*) as num_of_committees
                     from committee_assignment)
select h.first_name, h.surname,
       s.name as state, z.num_of_committees
from (select rep_id, num_of_committees
      from comm
      where num_of_committees =
            (select max(num_of_committees)
             from comm)) z
      join house h
        on h.rep_id = z.rep_id
      join states s
        on s.code = h.state
order by h.surname, h.first_name, s.name
```

This is the factorized version. Note that the HAVING of the revious query becomes a WHERE, as the aggregate looks like a column in a new (virtual) table.

Who are the representatives that participate in no committees?

```
select first_name, surname
from (select rep_id from house
      except
      select rep_id
      from committee_assignment) x
     join house h
       on h.rep_id = x.rep_id
```

Simple and correct – all people, minus those in committees.

```
select h.first_name, h.surname
from (select rep_id, first_name, surname
      from house) h
     join committee_assignment c
       on c.rep_id != h.rep_id
group by h.surname
```

Completely incorrect. Crosses everybody from table house with everybody in committee_assignment. The "join" condition will be true for everybody in table house. Expect many duplicate rows.

```
select h.rep_id, h.first_name, h.surname
from house h
     join committee_assignment c
       on c.rep_id = h.rep_id
group by h.rep_id
having count(*) = 0
```

Wrong too, although in a different way. The join with committee_assignment only selects from house people that ARE in a committee. The GROUP BY isn't standard (should be GROUP BY H.REP_ID, H_FIRST_NAME, H.SURNAME) but can work because rep_id is unique, bt can in no way return zero.

```
select *
from house z
     left join committees x
       on x.comm_id = z.rep_id
where committee is null
```

The idea was good, SQL coding was bad. The left outer join should have been on committee_assignment, not on committee, and of course it makes no sense to match the identifier of a committee to the identifier of a congressman – especially as some congressmen are probably associated with a value that you may find as committee identifier.

```
SELECT FIRST_NAME || ' ' || SURNAME COMPLETE_NAME
FROM HOUSE
     INNER JOIN
        (SELECT H.REP_ID
         FROM HOUSE H
              LEFT OUTER JOIN COMMITTEE_ASSIGNMENT CA
                ON H.REP_ID = CA.REP_ID
        WHERE CA.REP_ID IS NULL) NOPART
       ON HOUSE.REP_ID = NOPART.REP_ID
```

This one also used the left outer join idea, but did it perfectly correctly (I would have written it in the same way). The subquery with the left outer join selects identifiers in table house for which there is no match in committee_assignment (what he calls "NOPART" for "NO PARTicipation in a committee"), then joins with table house to retrieve the name.