**TOUGH VS EASY SQL QUERIES (NOTES)**

**How to recognize a “tough” sql query?**

3-liners are easy (Select-From-Where). Some other aggregate queries are also easy even though it takes 4-5 lines.

But many queries require nested subqueries. Often with correlated variables

Is there any way to know just by reading the query in English?

Yes!.

Semantic vs syntactic test

**Does a query require negation?**

**MONOTONE/NONMONOTONE queries**

A great test to “feel” if your query will need negation (EXCEPT, NOT EXISTS etc). Ask yourself the following question: could a tuple in the result of your query disappear as a result of adding more tuples to the underlying database? Remember the emphasis is on “could” – it does not have to \*always\* happen. One such example is enough. In fact you can empty the answer by adding some set of tuples, not ANY set of tuples. So it is enough to find one such set which when added to the database will result in a tuple (or a whole answer) disappear from the answer to the query. Some students are confused here by interpreting this test – as it should happen for any set of tuples added to the database. Not “should” – but “could”. Could is enough.

If so, the query is very likely “nonmonotonic” and will require negation, although some aggregates will also have this effect (i.e. COUNT, AVG, MAX, MIN will change as result of adding an extra tuple etc)

Examples (Monotonic or Not?)

Drinkers who do not frequent Joe’s bar

Drinkers who frequent only Joe’s bar

Drinkers who frequent exactly two bars

Drinkers who frequent at least two bars

Beers which are served in all bars

Bars which serve all beers that Steve likes

Drinkers who frequent bars which serve some beer they like

Drinkers who only frequent bars which serve beers they like

Bars with beer prices at most $6

Bars with beer prices at least at $6

**Example:**

**Drinkers who frequent ONLY ‘Cabana’**

SELECT DISTINCT f.drinker

FROM frequents f

WHERE f.bar =’Cabana’ AND

NOT EXISTS (Select \*

FROM frequents f1

WHERE f1.drinker=f.drinker AND f1.bar <>’Cabana’ )

**ENGLISH: Tough queries indicated by presence of keywords such as “not”, “only”, “all”, “at most”, “whenever”,**

**Simple queries:** 3-lines, Often termed “*Select-Project-Join*” queries. Probably a vast majority of real world queries are like this. The FROM clause lists all tables which are needed and WHERE specifies join and individual attribute conditions.

Select <Attributes> [PROJECT]

From R1, R2….Rn [JOIN]

Where <Condition> [SELECT]

*Self-joins* are used quite often as well. When would you join the relation with itself? When you want to refer to *two or more tuples in the same relation*.

**Simple queries**

**Examples** (all examples I recommend to practice on)

*Bars which serve ‘Corona’ or’ Bud’*

*Drinkers who frequent bars which serves Corona or Bud? Corona and Bud? ,*

*Drinkers who like at least one beer in Joe’s bar and who frequent that bar (Joins)*

*Drinkers who either frequent Joe’s Bar or like Bud, (and like Bud)*

*Beers which are served both at Joe’s Bar as well as at Havana (self-join)*

*Drinker who frequent bars which serve at least all beers that they (drinker) like*

*Bars which are frequented by both Steve and Bob*

**Nonmontone queries usually require SUBQUERIES in SQL**

We distinguish between sub-queries which are evaluated *only once for a query, or which evaluated repetitively, each time for a different value of a variable.* Easy sub-queries are evaluated only once when the query is executed. **Correlated subqueries are evaluated multiple times, depending on how many times the outer loop is evaluated.**

Tuple VARIABLES

“Find bars which serve just one beer”.

The way the solution was written is through correlated subquery which roughly did the following: It would loop through every beer and ask if there is a bar which serves it and for which there is no other beer which that bar serves.

select b.name,sells.bar

from beers b, sells

where sells.beer=b.name and NOT EXISTS

Select \*

From sells

where sells.beer<>b.name

For example the variable b may take on value – ‘Corona’ and subquery will be evaluated for ‘Corona’.

select b.name,sells.bar

from beers b, sells

where sells.beer=’corona” and NOT EXISTS

Select \*

From sells

where sells.beer<>’corona’

Now, that subquery will be evaluated as many times as there are beers. For each beer it is technically a different query.

**Examples of Non-monotone queries**

Bars which do not serve Corona

Bars which serve only Corona

Bars which serve all beers\* (here it is…no negation? In fact double negation…)

Drinkers who like only one beer (notice that you may solve this one using COUNT operator)

Bars which are frequented only by Drinkers who like Corona

The trick for non-monotone queries is to either negate the query and write negation first or use aggregate operators.

So…..Bars which serve only Corona*? Start with Bars which serve a beer which is not a Corona*

\*\*\*\*Challenge

Find Drinkers which do not frequent a single bar which Bob frequents

\*\*\*\*\*Challenge

“Find Bars which serve all beers that Joe’s bar serves and possibly more

\*\*\*\*CHALLENGE

Find beers which are served in all bars