# SQL Assignment, CMSC724, Spring 2011

Instructor: Amol Deshpande

**The assignment is to be done by yourself. Due date: Wednesday March 9, 9am.**

This assignment has three parts: (1) write a simple client program in java that accesses the database using embedded SQL; (2) understand the ANALYZE capabilities provided by PostgreSQL; (3) query processing and optimization. We will use the PostgreSQL database system for the first two parts.

***Submission: a single hardcopy containing everything would be preferred, but you can also send me a single email with the Java program, and any pdf or text files as separate attachments (please do not submit a zip file).***

## 1 PART 1 [4 pts]

One of more prominent ways to use a database system is using an external client, using APIs such as ODBC and JDBC. This allows you to run queries against the database and access the results from within say a Java program.

Here are some useful links:
`http://en.wikipedia.org/wiki/Java_Database_Connectivity`
`http://www.mkyong.com/java/how-do-connect-to-postgresql-with-jdbc-driver-java/`
`http://jdbc.postgresql.org/index.html`

The last link has detailed examples in the "documentation" section.

**Your Task:** Write a Java program that computes the correlation coefficient between "totalminutes" played and the number of "matches", and between "totalminutes" played and "numgoals". In the former case, we expect to see a fairly strong correlation, but in the latter case, almost no correlation is expected because the numgoals is 0 and 1 for over 80% of the players.

We will use the "sample correlation coefficient" as defined on this page. See the definition of $r_{xy}$:
`http://en.wikipedia.org/wiki/Correlation_and_dependence`.

The output of the program should be simply:

```
The sample correlation coefficient between totalminutes and matches is: xx.xx.
The sample correlation coefficient between totalminutes and numgoals is: xx.xx.
```

You should submit the Java program, and the answer itself should be submitted along with the rest of the assignment.

## 2 PART 2 [1+1+1+1+2=6 pts]

Like most other database systems, PostgreSQL allows you to analyze query execution plans, and tune the execution. The main PostgreSQL commands are EXPLAIN and EXPLAIN ANALYZE. The first one simply prints out the plan, but the second one also runs the query and compares the estimated values with actual values observed during execution.

An important accompanying command is "VACUUM ANALYZE"; this recalculates all the statistics used by PostgreSQL optimizer.

The assignment here is to answer 5 questions using these tools. The questions are present in the accompanying "assignment-analyze.sql" file. Most of the questions use a modified TPC-H Benchmark database, more details are available at: `http://www.tpc.org/tpch`. I am reproducing the schema of the dataset on the next page.

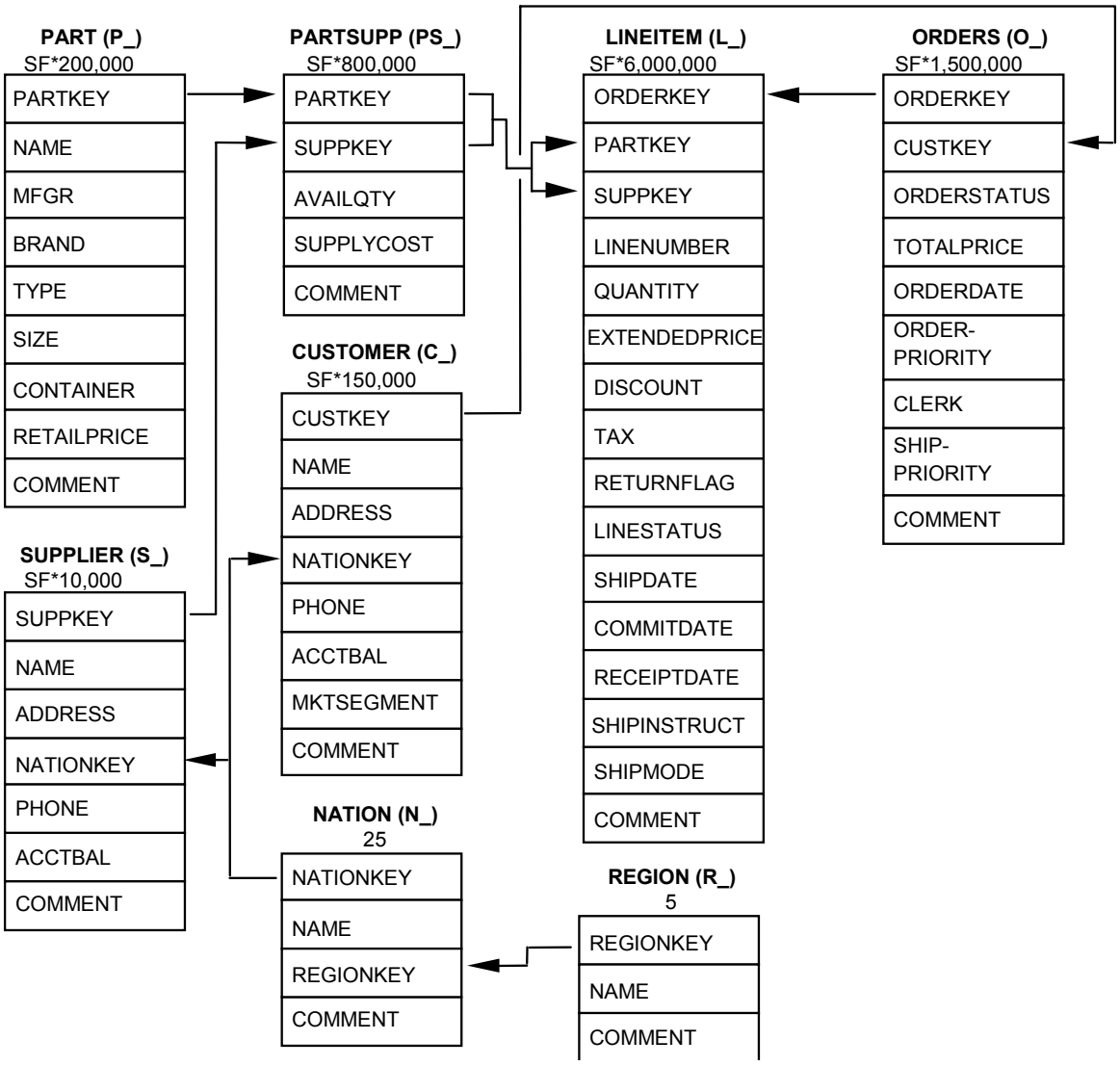


Figure 1: TPC-H Schema

Here is the file ”assignment-analyze.sql” verbatim.

```
---------------------------------
--- Soccer Data Set
---------------------------------
-- 1. Draw the query plan for the following query. Very briefly: how well did the
-- estimates match up the actual values?
select t1.name, t2.name, matchdate
from teams t1, teams t2, results
where ((t1.teamid = results.teamone and t2.teamid = results.teamtwo)
    or (t1.teamid = results.teamtwo and t2.teamid = results.teamone))
    and t1.groupname = t2.groupname and matchdate >= to_date('06-24-2006', 'MM-DD-YYYY');

---------------------------------
--- TPC-H Data Set
--- Download the tpch-load.sql file and the tpch.zip files. The first file
--- ("\i tpch-load.sql") will load the entire dataset.
---------------------------------

-- 2. PostgreSQL does not do a good job with the second and third query in the
-- following set, taking more time for (2) than for (1). Explain.
explain analyze select * from lineitem, orders where o_orderkey = l_orderkey;
explain analyze select * from lineitem, orders
        where extract(year from o_orderdate) = 1996 and o_orderkey = l_orderkey;
explain analyze select * from lineitem, orders
        where extract(year from o_orderdate) = 1997 and o_orderkey = l_orderkey;

-- 3. Inspite of a very selective join condition (returning a total of 1844
-- rows), the following query does not execute efficiently. Why?  How may
-- you fix it? There is a very easy fix which PostgreSQL suprising does not
-- automatically employ.
select count(*) from orders, lineitem where l_shipdate - o_orderdate = 1
        and extract(year from o_orderdate) = 1992;

-- 4. Ranking: doing ranking using the rank() function is easy and databases
-- optimize that. However, without using rank() you are forced to do something
-- like the following: here we assume c_acctbal is unique, and we want to rank
-- by that variable and find the top 10.  Draw the query plan for the following
-- query.
select c_custkey, c_name, (select count(*) from customer c2
                           where c2.c_acctbal > c1.c_acctbal) as rank
from customer c1
where (select count(*) from customer c2 where c2.c_acctbal > c1.c_acctbal) < 10
order by rank asc;

-- Explicitly using rank() is much more efficient, but rank() was not part of
-- earlier SQL standards, although most databases supported it (with differing
-- syntax) [[ You don't have to do anything here. ]]
explain analyze select c_custkey, c_name,  RANK() OVER (ORDER BY c_acctbal desc)
    from customer limit 10;

-- 5. For the following query, explain in some detail why the final result size
-- is underestimated in one case, and overestimated in the other.
explain analyze select * from supplier,customer
        where s_nationkey=c_nationkey and s_acctbal < 5000;
explain analyze select * from supplier,customer
        where s_nationkey=c_nationkey and s_acctbal > 5000;
```

# 3 PART 3 [5 pts]

Answer the following questions. Be concise.

1. Bloom filter (`http://en.wikipedia.org/wiki/Bloom_filter`): You have to join two relations, $R(a, b)$ and $S(b, c)$ on $b$, that are on two different sites, and the result is to be collected at a third site. Assume all sites are equidistant from each other, and that all attributes are integer attributes (4 bytes each). One option to do such a join is to use a Bloom filter, wherein:

   - a Bloom filter is constructed at one of the sites on the join attribute (e.g. say on $R.b$),
   - it is sent to the second site ($S$),
   - only those tuples of $S$ that find matches in the Bloom filters are sent to $R$ (call this $S'$), and
   - $R \bowtie S'$ is sent to the third site.

   The naive option is to simply send $S$ to $R$ and send the result, $R \bowtie S$, to the third site.

   For each of the following scenarios, decide whether a Bloom filter would be useful for minimizing the total communication cost. If yes, find the optimal Bloom filter size (or write out an equation that needs to be solved for finding the optimal size).

   - $|R| = 10000, |S| = 10000$, $R.b$ and $S.b$ distributed uniformly between 0 and 100.
   - $|R| = 10000, |S| = 10000$, $R.b$ and $S.b$ distributed uniformly between 0 and 1000.
   - $|R| = 10000, |S| = 10000$, $R.b$ and $S.b$ distributed uniformly between 0 and 100000.

2. Prove that sorting $N$ blocks of data using at most two passes, needs at least $O(\sqrt{n})$ blocks of memory. For being able to sort larger amounts of data using just two passes, would this suggest increasing or decreasing the block size ? What are the cons of doing that ?

   What is the largest size relation you can sort assuming you have exactly $N$ tuples worth of memory, assuming that you can choose the block size ?

3. Explain the notion of "interesting orders" with an example. Can you suggest some other properties of the intermediate results that should be considered during query optimization?

4. Consider the "time to first tuple" metric. What kinds of operators would you favor to optimize for that metric?

5. Draw the dynamic programming tree (assume no interesting orders) for the following join query, where we do not allow Cartesian products but "bushy" plans are allowed. The tree should have one node for every intermediate relation that is considered (e.g., $RST$ indicates the intermediate relation $R \bowtie S \bowtie T$), and edges indicating the possible ways to construct an intermediate relation using other intermediate or base relations.

```
select *
from R, S, T, U, V
where R.a = S.a and S.b = T.b and T.c = U.c and T.d = V.d
```