336 PROJECT – 2018- DEFAULT (group of 2)

Bar-Beer-Drinker PLUS

We will extend the bar-beer-drinker scheme, with information about specific transactions by customers (drinkers). We will store all "bills" of all drinkers, who may also order food items (bar food etc).

For each bill we will store the details of the transaction – what items were purchased (may be beer, may be food, soft drinks etc), time when the bill was issued, name of the drinker and the transaction id. Each transaction will also total amount paid which will be sum of prices of all items on the bill plus 7% tax. There may also be a tip.

One of the first tasks will be to represent bills/transactions in ER diagram and in your relational scheme

In addition, information about each bar (bar table) must also have opening and closing hours of this bar for each day. These hours may vary from bar to bar.

You will have to populate your database with realistic, but synthetic, tuples. By realistic, I mean names of bars, drinker names, dollar figures etc when appropriate. No a1, b1, c1! No drinker X and drinker Y! Generate and load your db with the large number of synthetic tuples, may be 5,000? Of course even more transactions, may be 20,000? May be more. It is your choice. But your database instances should not be completely random. The following assertions (patterns) should hold:

PATTERNS

1) Transactions/bills cannot be issued at times when the given bar is closed

2) Drinkers cannot frequent bars in different state

3) For every two beers, b1 and b2, different bars may charge differently for b1 and b2 but b1 should either be less expensive than b2 in ALL bars or more expensive than b2 in ALL bars. Cannot be the case that in one bar Corona is more expensive than Bud and in another Bud is more expensive than Corona. But Corona may be more expensive than Bud in one bar, and have the same price as Bud in another.

LOADING YOUR DB: You are not going to load your database manually (just in case you wondered). You will write a program which will load the database for you. Data will be "almost" random, except the pattern(s). The logic of the pattern will be part of your program.

REQUIREMENTS

1) Correct ER diagram and database scheme definition

2) Realistic db instance generation (i.e. names of entities, not symbols) and db loading + Pattern embedding in the instance as well as validation of the pattern using SQL query. Have at least 5,000 tuples in your database instance excluding transactions/bills. At least 20,000 transactions. Provide Verification interface for your three patterns: which allows to us to verify assertions/patterns which you have embedded. Show the SQL query which you are using to verify your constraint and return TRUE or FALSE (in case assertion is not satisfied). So there will be three SQL verification queries – one for each pattern – as part of your front end interface (the final design is up to you)

3) FRONT END - QUERIES: you should have three pages: Drinker, Bar and Beer

a) **DRINKER PAGE**: Given a drinker, show all his/her transactions ordered by time and grouped by different bars. Show bar graphs of beers s/he orders the most. Also, bar graph of his/her spending in different bars, on different dates/weeks/months.

b) **BAR PAGE**: Given a bar, show bar graphs – 1) for top drinkers who are largest spenders, 2) for beers which are most popular and 3) for manufacturers who sell the most beers. Demonstrate time distribution of sales, show what are the busiest periods of the day and of the week. Also have a way to add a new transaction for a given date etc.

c) **BEER PAGE**: Given a beer – show bars where this beer sells the most (again only top), show also drinkers who are the biggest consumers of this beer as well as time distribution of when this beer sells the most.

d) **SQL QUERY INTERFACE:** Provide a box where we can type in sql query and get them evaluated on your database. Of course we will have to know the scheme (which will be part of your submission).

4) FRONT END – UPDATES/DELETIONS/INSERTIONS: Allow end user to modify every table in your databases.

Have MODIFICATION page, with one box for each table. If update is not accepted – provide the feedback message “violates foreign key” etc.

The integrity constraints should be implemented

a) **Foreign keys** for each of the three tables – frequents, likes and sells. Drinker, Bar and Beer should be present in tables Drinker, Bar, Beer *before* they can participate in tables Frequent, Likes and Sells.

b) **Key constraint** for sells table – on bar, beer (bar, beer -> price)

c) **ASSERTIONS**: All three patterns (1-3) from above have to be enforced as assertions. This means if an update (insert, update) violates the assertion it *should not be allowed* and proper warning/explanation has to be displayed – "not accepted due to violation of assertion X"

Submit your ER diagram and Relational Schema through Sakai by end of the day, **October 7th**

1. Populate your database with instances, embedded patterns and SQL queries which verify that pattern is satisfied by the data by **October 28th verification also, python, queries…**
2. Front end and creation of power point presentation describing what you did. Final submission of URL where we can test your product and other documentation (ppt, source code etc) – by **November 18**

FINAL SUBMISSION SHOULD CONTAIN

1. ER diagram and Relational DB scheme
2. URL where we can access your web application through the front end
3. Power Point Presentation (up to 10-15 slides, explaining what you did)
4. Source code

GROUPS

Default is: 2 students per group. It is self-selected. We do not want to connect random two students, this does not work!. So we will leave it up to you.

You can do this project alone (the winner group last year was a single student). I really do not recommend 3-student groups, you have to do more and it is not clear what it means plus my experience is that 3-student groups often do not work well.

We assume that everyone contributes equally. If it is not the case (one student complains) – we will ask each member of the group to itemize what they did.