CS 387 Project (Spring 2022)

Goals

The goal is to walk you through the different phases of building a database information system (application) for a realistic application. The holistic approach ensures that you are exposed to every aspect from understanding requirements, analyzing them to come up with a design, implementing the system and testing it thoroughly. DBIS consists of multiple components beyond that of the data model - there is the business logic of the application that encodes decisions based on data and the interface with the users that allow them to interact meaningfully with the application. Hence the project will be a team effort where each member plays a different role for maximum productivity. ***Typical*** DBIS applications work on the MVC architecture - a model captures the data being stored by the system, a view exposes the model selectively to the users and a controller reads both the data and user input and makes decisions as to how to manipulate the data to achieve the use cases of the system.

For students interested in working solely within the DB engine, the project can take on a different form - that of manipulating the internals of the DB to improve performance or reliability of the system. An excellent project is to add transactions and some of the ACID properties to ToyDB. Both types of projects are equally acceptable as long as the instructor is satisfied with the quantum and complexity of the work involved for 4 members.

Some Project Ideas

I have put down a few ideas as representative samples. By no means is this comprehensive and I encourage you to come up with original ideas.

1. **Question bank and paper generator:** Allow users to enter questions and tag them with topics, sub topics, levels of difficulty, appropriateness for specific kinds of exams. Also allow data involving how well the question was received when it was used, how often was it used, in what kinds of exams, demographics of students who answered it or did not answer it etc. From this, allow faculty to generate question papers based on a series of parameters they supply. Also generate reports on how useful the questions in the bank are based on parameters supplied by the user. Allow for hierarchy of topics.
2. **An Application Monitor:** Systems such as Nagios help track how much resources are used over time on a computer system or network device. They embed what is termed a Time Series DB such as Influx. This project requires you to build a monitor for server side Applications - something that will track everything about an application - its deployment footprint, hard resources consumed (CPU, memory, Disk, Network bandwidth) with time and soft resources consumed (DB connections, file descriptors, threads, DB cache etc.) and predict issues before they can arise (for example the middle tier is about to run out of threads in 1 hour with the current request rate). This app can also help auto tune an application by suggesting the indexes to use for DB queries etc.
3. **A Car Network of Geo Spatial Data:** Today's cars emit a wealth of data enabled by detailed tracking of parameters both in and out of the car (such as Teslas). The data coming from the car tracks driving conditions, car performance etc. while external data may be traffic density, weather, road conditions etc. An application would analyze this data in real time to suggest driving patterns as well as do long term analysis to plan for events such as road widening or dynamic lane direction changes etc. There are geospatial extensions to Postgres that you will need for one of these.
4. **Restaurant Management System:** The restaurant maintains the catalog for the list of food and beverage items that it provides. The system needs to track ingredients purchased, dishes cooked and the ingredients used in each dish for the day. The restaurant provides tables on premises for which occupancy data needs to be tracked by time of day and day of week. Apart from providing food facilities

at their own premises, the restaurant takes orders online through their site. Orders on the phone are also entertained. To deliver the orders, they have delivery people. Each delivery person is assigned to the specific set of area codes as primary codes and a set of area codes as secondary codes. Each area code can be served by more than one delivery person The customer record is maintained so that premium customers can be awarded discounts.

5. **Using Time Series DBs in Visualization and Analytics:** The definition of Time series database is a type of database which is particularly built for time-stamped data. This can perform special operations like handling performance metrics, measurements, and time-stamped events. Like standard databases, the user will be able to perform DDL and DML to organize the time-stamped data in an effective way. The TSDB is specially designed to question the changes that took place over time. InfluxDB is one open source TSDB. In this project you will employ such a TSDB to visualize and analyze data from any source of your choice - weather data, Covid 19 data, economical growth data etc. Use a graphing tool such as Chronograf with InfluxDB for professional looking output.

Project Deliverables:

We divide the project into 3 phases each with its own set of deliverables:

1. **Requirements and Analysis phase:**
    a. Describe the problem concisely (less than about 300 words) - this involves a short description of the domain and the intended use of the application assuming its a black box - don't describe the how - focus on WHAT the application is to do. Why is a database (as opposed to, say, a simple set of files) a good idea for this task? What kind of db (graph, time series, relational, key value store etc.) and why?
    b. List the intended classes of users of your database system - for example, a restaurant manager, a delivery organizer, delivery person, head waiter, Billing user etc. For each class, put down HOW you expect them to use the system as a set of use cases. A use case sets out the user class, operation the user is trying to accomplish, the set of inputs he/she provides and the interaction flow with the system (assume the system is a black box for this as well).
    c. Identify 10-15 main "things" (entities) about which you will need to keep information, and the details you will need to keep for each entity. For example: a table in a restaurant has a unique identifier, its location (window side etc.), its occupancy etc.
    d. Sketch the user interfaces (i.e., forms) required for each of the use cases (both input and output - in some cases the output will just be a confirmation). The outputs correspond roughly to "views" on the relational schema.
    e. Construct conceptual (ER) diagrams for the system. This should describe the entities, its attributes and the relationships across these entities. Some of these entities may be abstract describing a relation rather than correspond to a physical entity. For example, you may choose to have a table-customer relation that captures certain aspects of customers who specifically requested such tables when booking one.

    Steps d. & e. represent the analysis activities here.

2. **Design Phase:**
    a. Derive a ***logical schema*** from the above. Normalize at least to 3NF.
    b. Identify and explain all ***integrity constraints***. Describe which attributes are allowed to be NULL, why, and what a NULL value means for each attribute.
    c. If you need any (materialized) ***views***, specify them at this step and describe why you need these views.
    d. For each use case of 1b. describe the transaction(s) in terms of the DB operations needed to implement the use case.
    e. Create the DDL.sql file and a InsertData.sql file that will install the schema and install data. If you have additional views that you need, this file should contain instructions to install these views.
    f. For each transaction of 2d, write the ***SQL*** and test using psql/pgadmin4.
    g. Based on these, identify the ***indexes*** you believe will help optimize performance of the SQL queries.
    h. For the user interaction, sketch the ***forms*** that each use case will need - decide the technologies for implementing these forms.
    i. Design and describe a ***business logic controller*** to drive the use cases. A case may have multiple steps with a viewing form and a set of possible actions per step. Here is where you will need to embed your free SQL into the application logic.

j. Have a ***test plan document*** that is a enumeration of the test cases to drive each use case under all possible situations of inputs the users can provide - each test case should list expected output.

3. **<u>Test & Submission Phase</u>**
    a. Integrate the model, views and controller to produce a complete working application.
    b. Test the application *functionally* based on your test plan - mark the test cases that pass, those that fail clearly and try and identify reasons why these cases failed. Maintain detailed documentation.
    c. If applicable, test the application under some scale - have a few clients using a load testing tool such as JMeter for example. Load tests reveal performance problems - your job is to profile the application under load and chart its behavior with increasing load. This will be part of your final report.
    d. Submit your github repo which should include the ***<u>detailed report</u>*** - ***requirements, designs, test plan, test results and profiling charts under load.***

Timeline of submissions for the project (approx and subject to some change)

Project Deliverable	Approx deadline (to be finalized on Moodle)	% of project grade
Team formation, first sketch of the project requirements defining scope.	**Mar 7th**	**5**
Complete requirements & Analysis: Detailed enumerated english requirements, Complete ER diagram, forms, use cases in detail.	**Mar 20th**	**15**
Design Doc: Normalized schema, Indexes, DDL, Data load scripts, SQL for transactions, Detailed forms, (optionally) Controller logic	**Apr 1st**	**30**
Detailed Test Plan including load testing	**Apr 1st**	**10**
Final report (everything so far + test results + load testing results + conclusions)	**Apr 20th**	**10**
Demo and viva	**TBA**	**30**