CSCE 315 | Daugherity

Project 3: Design Document

Team 5

Russell Pier | Cole Boggus | Ellie Miller | Liuyi Jin
6-23-2017

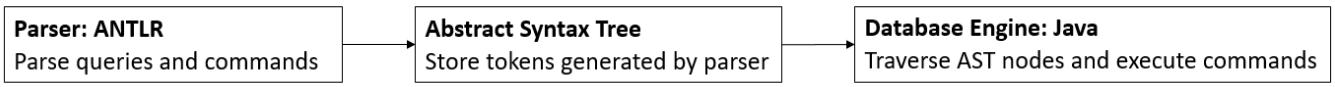
Design Document

Section 1: Purpose

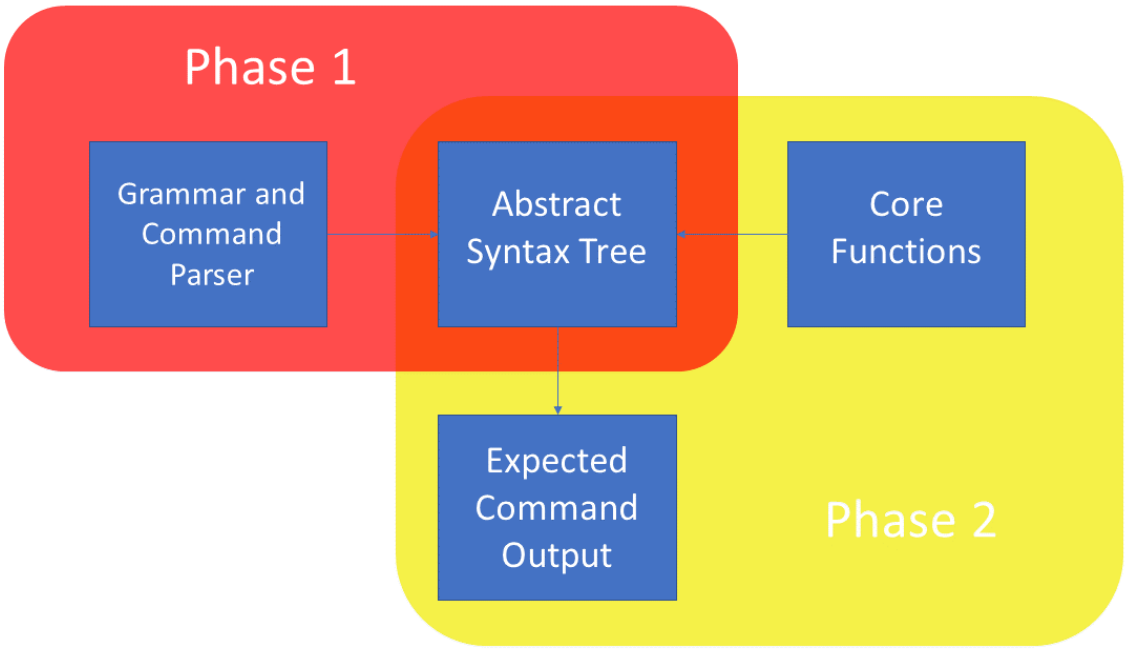
The overarching purpose of this project is to create a generic database management system that can carry out the commands provided in the data manipulation language based on the abstract syntax tree created from parsing these commands and queries. Phase 1 of the project involves creating a parser code that can accept or reject a command or query and parse the command or query accordingly. Phase 2 builds on Phase 1 and will actually execute given commands and queries by traversing the abstract syntax tree and evaluating the nodes.

Section 2: High-Level Entities

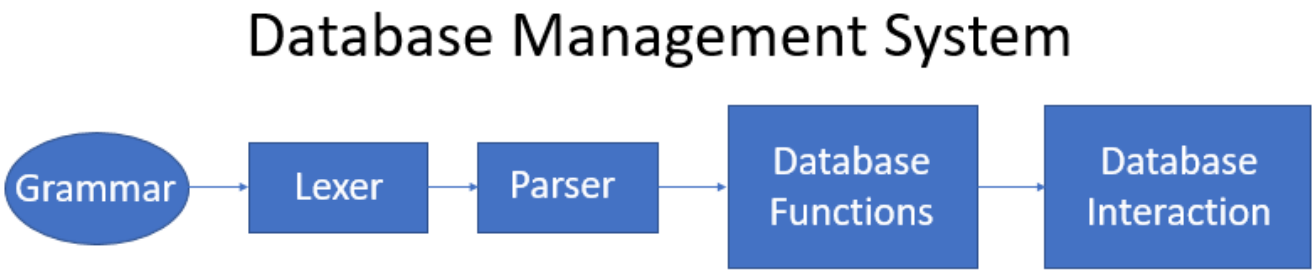
The first phase of this project involves implementing a database management system (DBMS) that is capable of supporting the data manipulation language (DML) provided in the project specifications to parse queries and commands into an abstract syntax tree (AST). In order to complete this, we will be implementing ANTLR - a parser generator. This parser generator will take the provided grammar and produce an AST. This tree object can then be traversed to carry out the second phase of the project - implementing the core functions. The AST will be the only object that is known by phase 1 and phase 2 of this project. All the relations implemented in the database will be user-defined and are independent of the parser.
Phase 2 involves implementing the core command functions of the DBMS. After parsing the commands of the DML, the DBMS will actually carry out these command functions based on the data stored in the database. The invocation of these functions will involve generic programming that will allow for any relation to be defined and then manipulated via the defined command functions. However, in preliminary trials of verifying these commands, we will be testing our own database. This test database will contain relations related to the Texas A&M course scheduler and will include relations such as Instructor, Student, Class, and the like. Since this program is built on test-driven development, creating our own database relations will be extremely beneficial to verify correctness of the core functions.
The entire process can be simply described in the following figure. As seen, the two phases of parsing the DML and executing the commands via a database engine are connected via the AST. ANTLR will automatically create the AST according to the programmed grammar so the main focus of our project will be creating the grammar and implementing the commands.



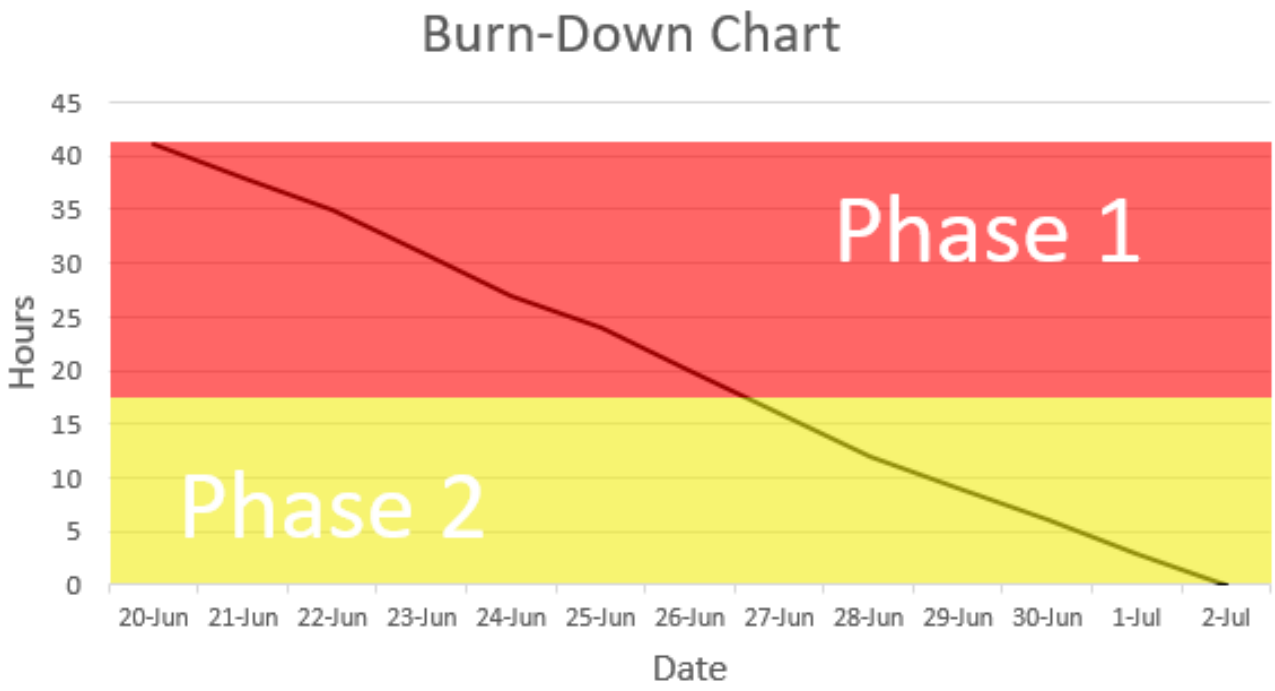
The interaction between Phase 1 and Phase 2 can be seen in the figure below.



The implementation process of the DBMS can be seen in the layout below.

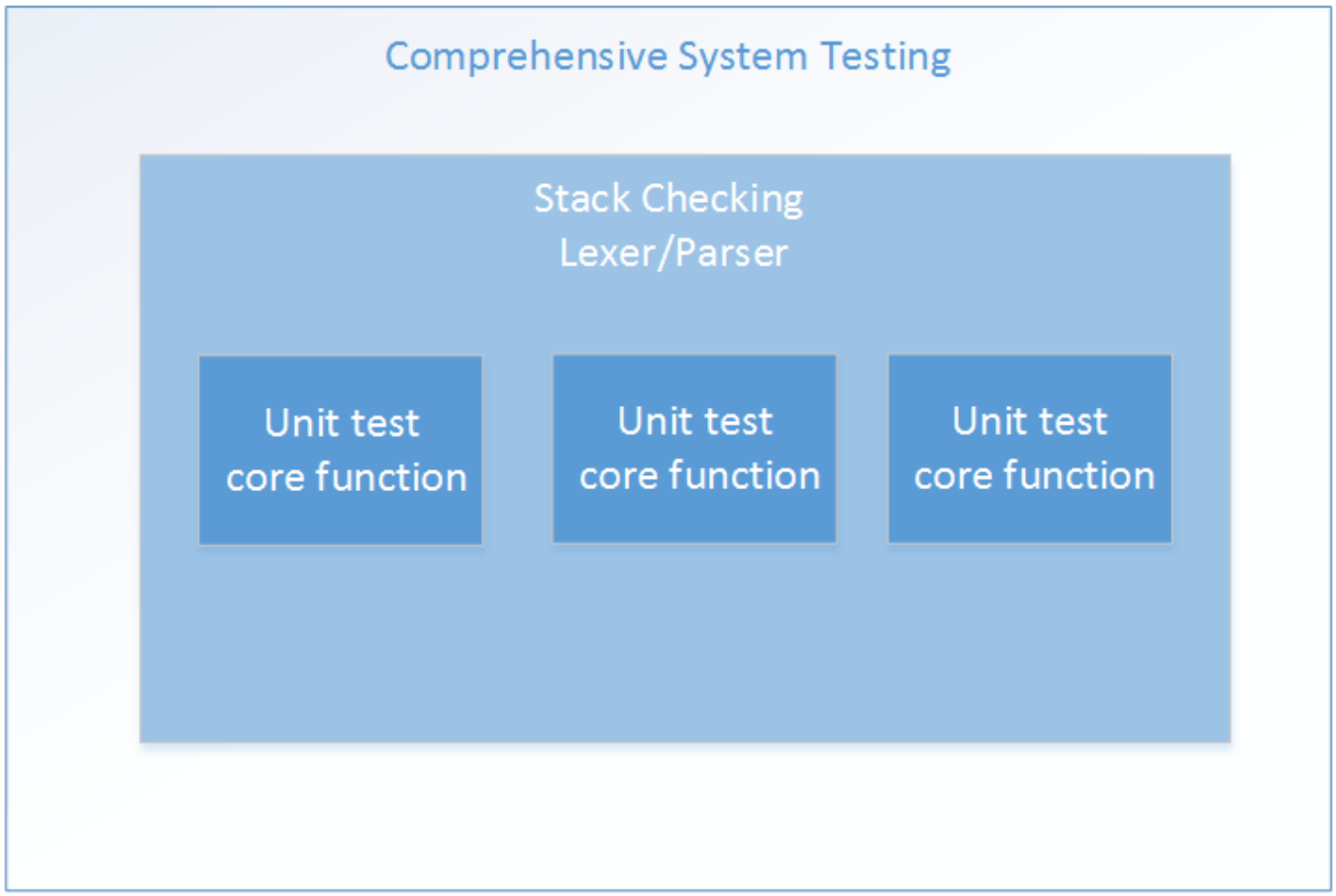


Our anticipated burn-down chart for this project:

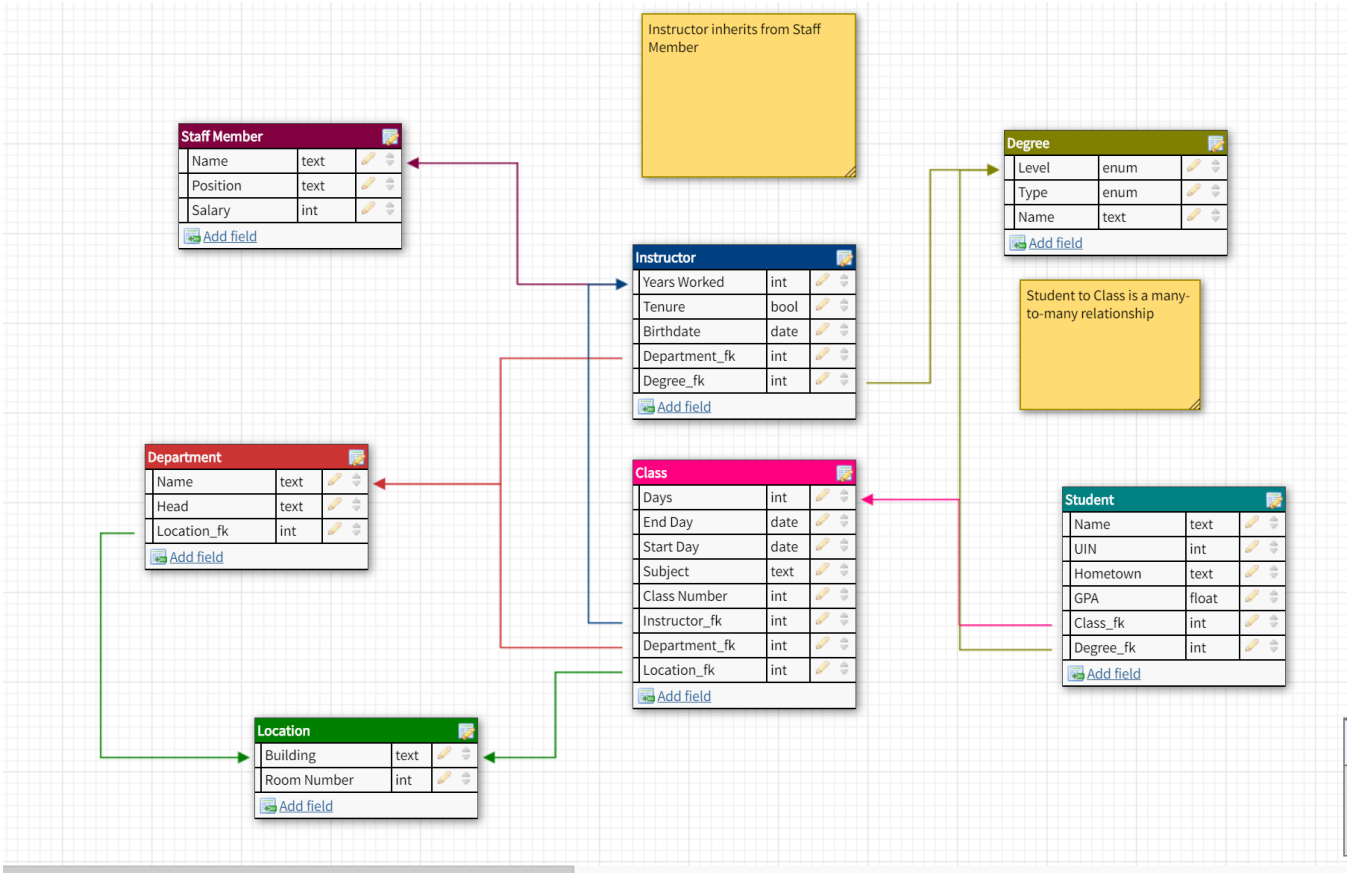


Section 3: Low-Level Design

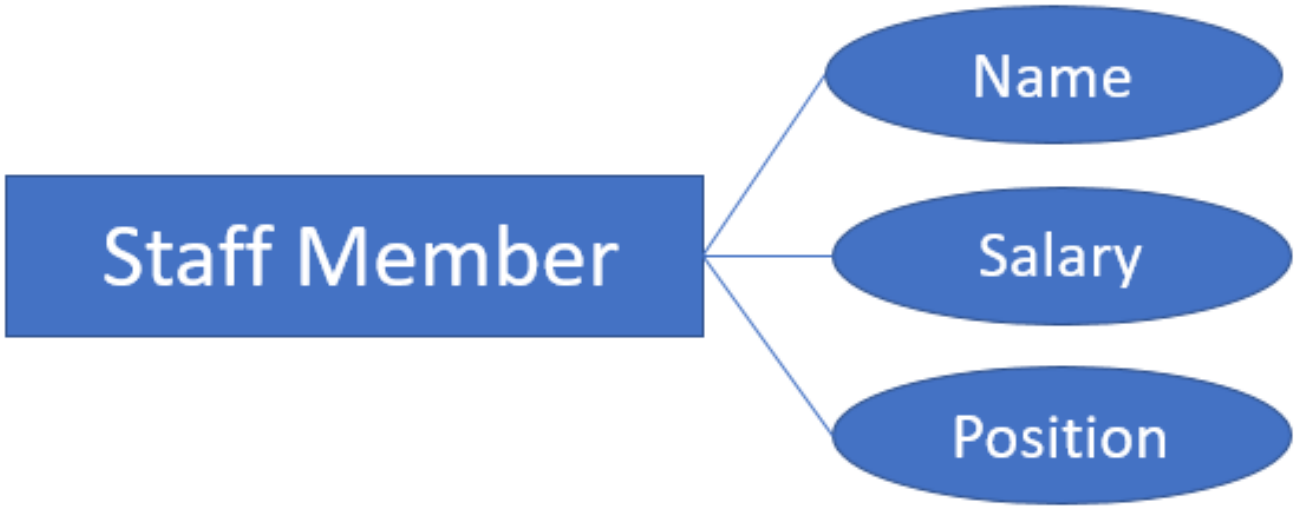
ANTLR takes care of a lot of the difficult and tedious parts of our design. It drives both the parser and the AST. We will create a lexer based on the grammar outlined in the specification document and ANTLR will turn that lexer into a parser and call the core functions as needed to execute the commands given by the input. To have a design-driven approach, we will first design a test that will check the expected output of the entire system. Next, we will test the lexer/parser by checking the stack of core functions that are called in response to a particular input. Lastly, we can unit test our individual core functions.

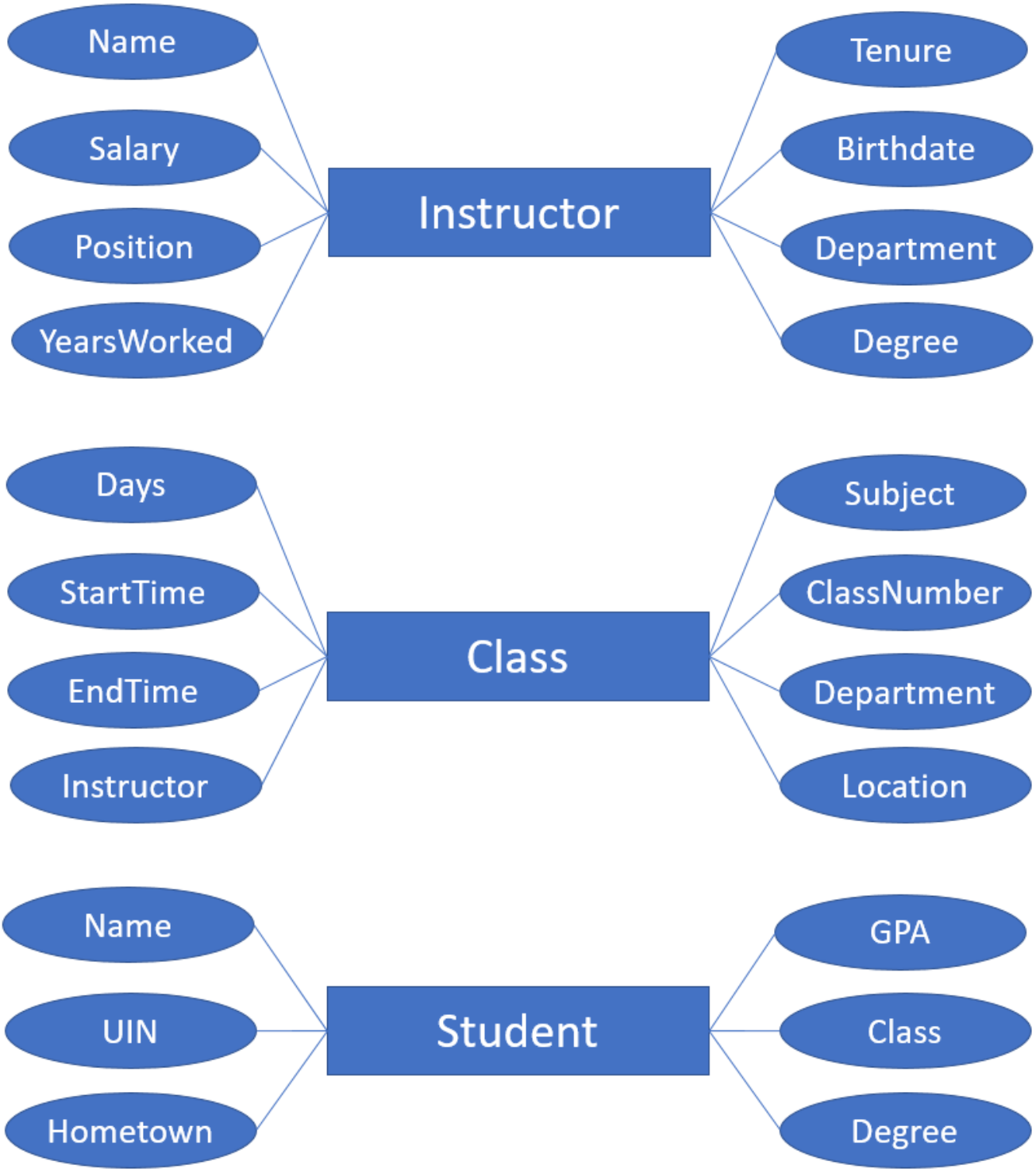


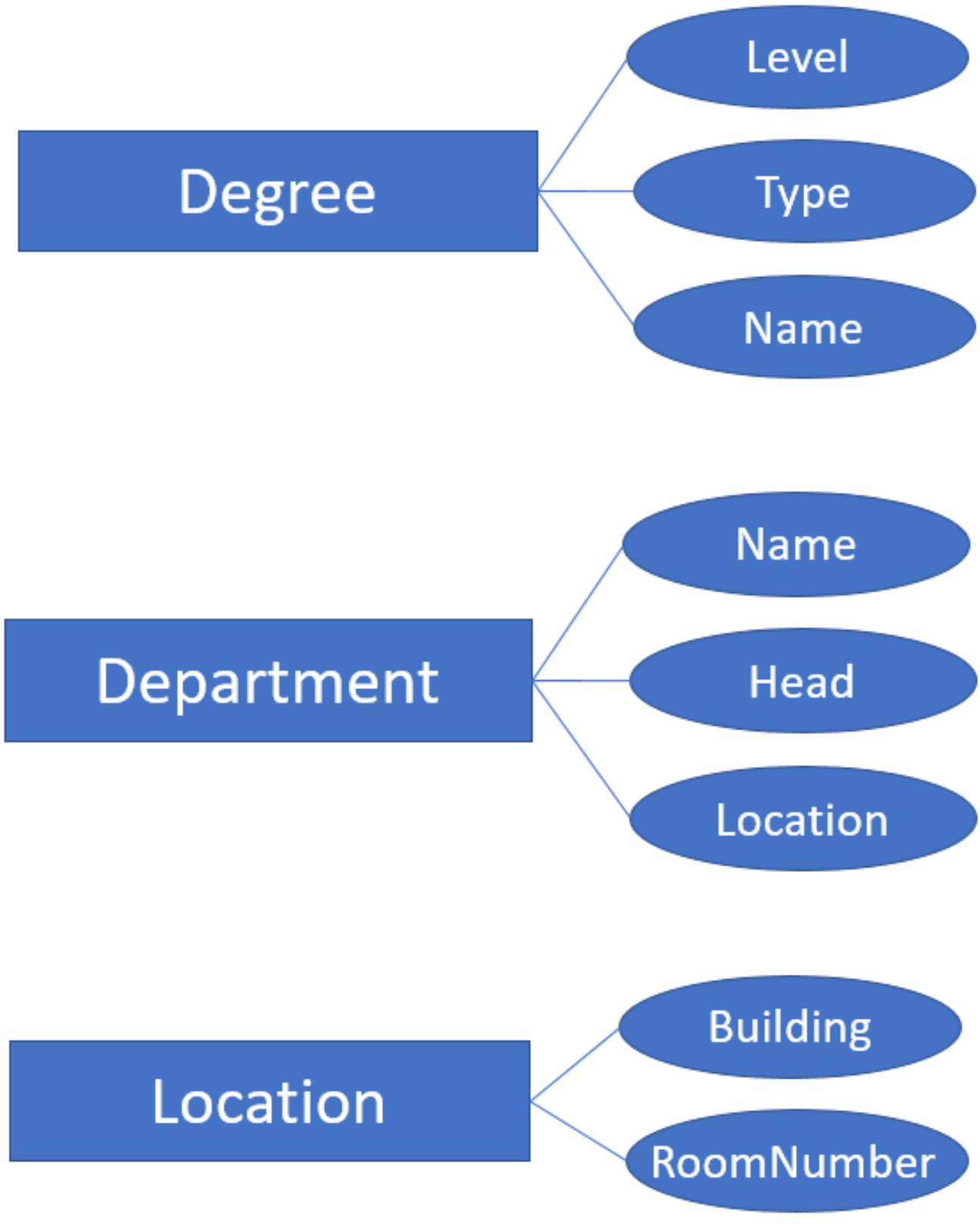
The figure below displays the entity relation diagram for the relations we created. This is one entity relation that we can use, though the database engine will work for any generic database. This database describes a model of classes at TAMU including students, instructors, departments, buildings, and degrees. Most relations are one-to-many (e.g. ONE department contains MANY instructors) with the exception of the Class/Student relation being many-to-many, and the Staff Member/Instructor relationship being a simple inheritance (not a database relation).



The individual relation schemas can be seen in the following figures that display the attributes of each individual relation.




Name
Salary
Position
YearsWorked
Instructor
Tenure
Birthdate
Department
Degree
Days
StartTime
EndTime
Instructor
Class
Subject
ClassNumber
Department
Location
Name
UIN
Hometown
Student
GPA
Class
Degree


Degree
Level
Type
Name
Department
Name
Head
Location
Location
Building
RoomNumber

Section 4: Benefits, Assumptions, Risks and Issues

Benefits

- One benefit of our design is that we are utilizing ANTLR in our parser. ANTLR is a powerful parser generator that will expedite the creation of our parser as well as ease the process of test sessions. Our preliminary research shows that ANTLR can generate a syntax tree based on a grammar and can also traverse that syntax tree which will be incredibly useful when Phase 2 requires implementing the database core functions.
- ANTLR will allow editing our grammar when necessary, allowing adaptability.
- Using the IntelliJ IDE we can easily implement ANTLR4 into Java.
- Using a sprint burndown, we can easily assign tasks to all of our four team members so that we are operating with maximum efficiency.
- Using a MySQL (over strict SQL) based language to implement the query commands simplifies interacting with the database.

Assumptions

- One assumption in our design is that the language will not vary from the one provided in the project description. Therefore, if a command is given that was not explicitly defined in the project description, an error will be thrown before asking the user for valid input.
- As was stated in the specification document, the database engine we create can be simplistic and does not needs to include complex features such as network sockets, secure logins, etc..

Risks

- One risk of our design is implementing ANTLR in our parser. While it seems it will be easy to create the syntax tree and traverse the tree via ANTLR, it could prove to be difficult to implement the core functions executions due to combining the grammar parser and command functions.
- Learning ANTLR may take longer than it would take to just write our own parser.
- Modular code and splitting the workload up requires more detailed planning and outlining to know exactly how modules work together.