**Project Title:** Employment Application Database for 49er Golf Cart Service

a term project submitted by

**Team ER(R)**

**Group Members:**  
Richard Alaimo

Rachel Pullen

Lizzy Thomas

**Date:** May 1st, 2018



**Executive Summary**

This project consisted of developing a database system for a potential golf cart service at the University of North Carolina at Charlotte (UNC Charlotte). The use case for the database system is to store information that is recorded when a user is interested in applying for a job for the golf cart service. Storing this information is critical for job recruiters when they are reviewing applications. Therefore, there is a strong business need for a database solution to be implemented for this use case. It was required from the project advisor, Dr. Thompson, that the database system satisfies the following requirements:

* The enhanced entity relationship diagram (EERD) is fully normalized
* The database contains eight to ten tables
* Generalization/specialization is implemented in the database
* A many-to-many relationship is included in the database

Relationships between entities (i.e., tables) are described using the EERD, as well as attributes (i.e., fields) that are used to describe each entity. It was decided that MySQL Workbench would be the best software to use since it is relatively simple for the database tables to be created once the EERD is generated. Following the creation of the tables, it was necessary to populate them with test data to ensure that they are structured correctly. When generating the test data, it was important for all possible scenarios as to how data could be stored in the database to be considered. There is a website that was used by the team, called GenerateData.com, to create the test data.

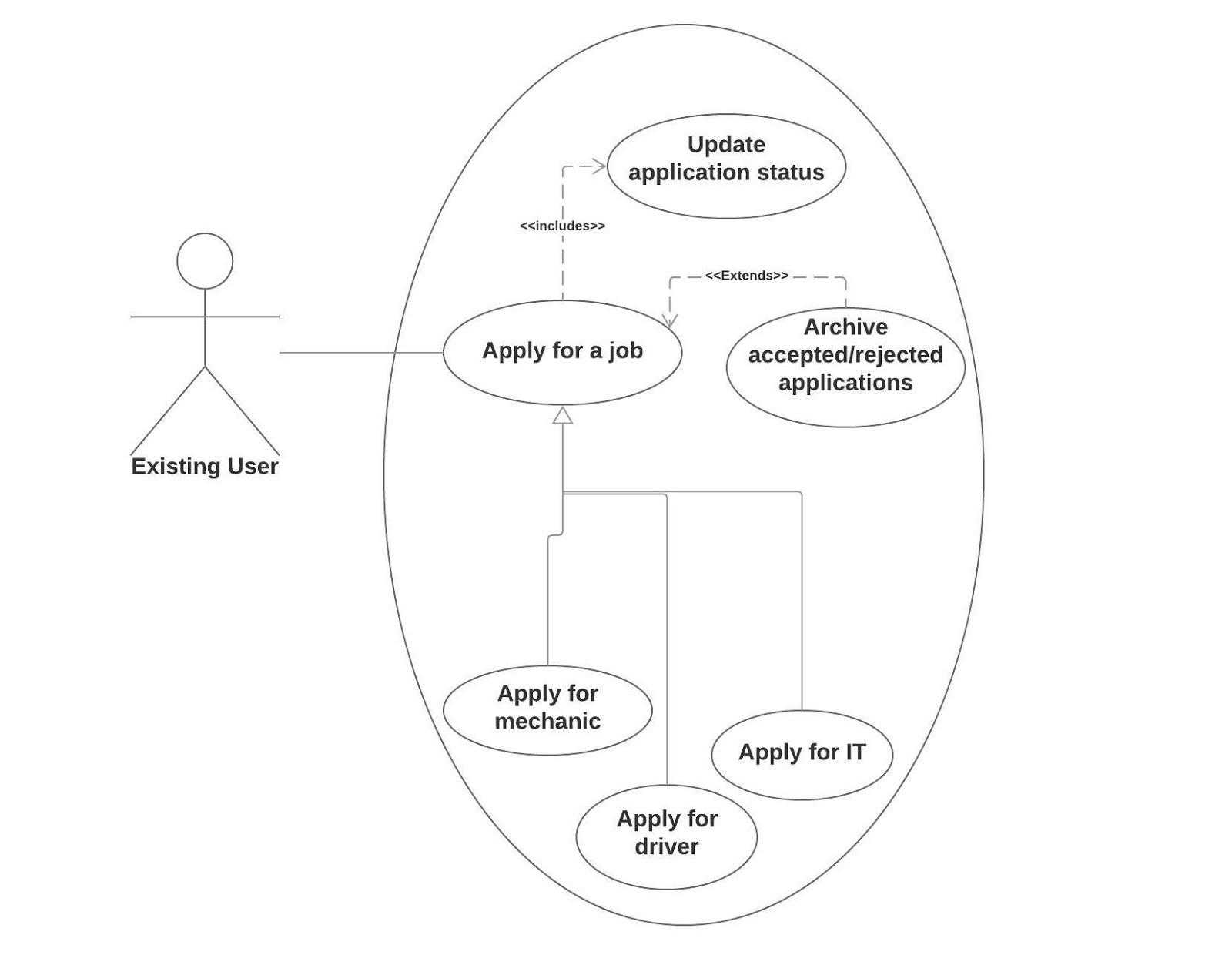
In addition to these requirements, advanced SQL must be generated, including one trigger, one stored procedure, three reports that incorporate nested queries and conditional statements and have indexes included, and one transaction. In addition to the advanced SQL, a demonstration of an Update and Delete command within the database is required. Also, it was required for a user-interface (UI) to be created that demonstrates a small portion of the database system is functional. For this project, a small portion of the database is described as a single table. The UI must allow the user to create, read, update, and delete (CRUD) information that is stored in the table. In addition, the user should be able to retrieve data from one of the reports through the UI. It was decided that the UI would be programmed using PHP.

The final deliverables for this project were a report that outlined the design process of the database system, a SQL file that contains the necessary statements to create and populate the tables, as well as execute the advanced SQL that was created, and a UI that satisfies the requirements mentioned before. The team is confident that an effective software solution was created for this project and hope that is eventually implemented if UNC Charlotte decides to create a golf cart service for people on campus.

**Use Case Diagram and Business Rules**

Before the actual design and development of the database can be performed, it is important to fully understand what requirements are being asked of from the user who desires the software solution. During this phase, it is necessary to understand how users will interact with the system (referred to as a use case), as well as defining the process that is used by the user for performing their work (referred to as business rules). An understanding of the process is essential since the database system must allow users to perform their work without any negative impact once implemented.

Generally speaking, a use case is a high-level overview as to how users are expected to interact with the system once it is implemented. It is common for a use case diagram to be created that displays users as “actors” and outlines what actions are possible for the user. The use case for this project is to allow users to apply for jobs that are offered from the golf cart service. Also, it is desired to allow recruiters from the golf cart service to review job applications that are submitted and decide whether or not to hire the user who submitted the application. The use case diagram that was generated and approved by the project advisor is displayed below:



**Figure 1**: Use Case Diagram

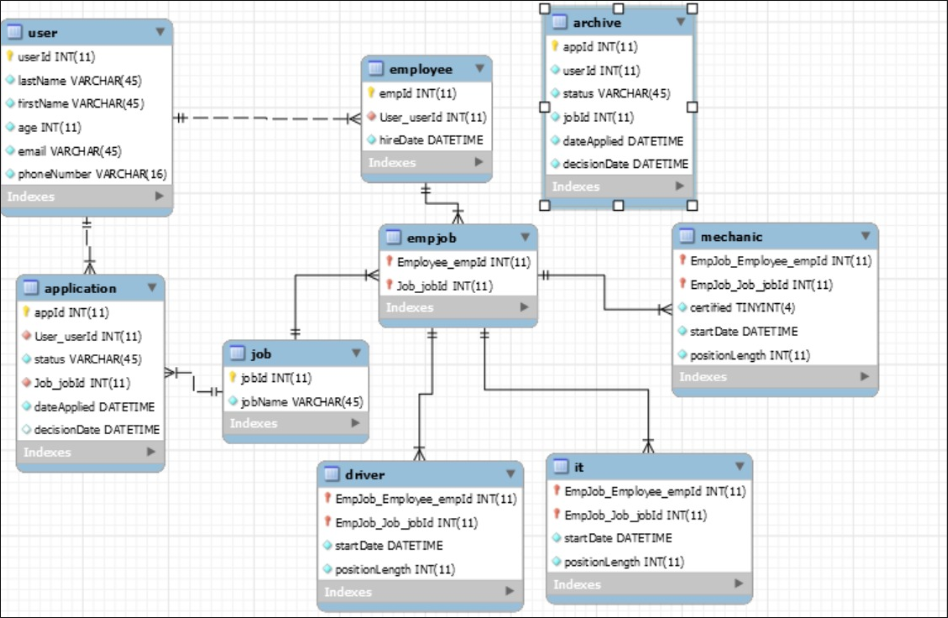
Business rules provide a high-level overview of how users desire the system to operate, which is a direct representation of a business process. They describe relationships between entities and define what attributes are needed. The business rules that were defined for this project are displayed below:

* An employee must be a user, but a user does not have to be an employee (partial participation).
* A user ID can be associated with 0 or 1 employee ID.
* A user can complete 0 or many job applications.
* There are 3 jobs users can apply for: Driver, Mechanic, and IT.
* A mechanic can either be certified or not certified.
* Each job application is associated with 1 and only 1 user.
* Each job application is associated with 1 and only 1 job.
* A job type can have 0 or many job applications associated with it.
* An employee can have 1 or many jobs.
* A job can be assigned to 1 or many employees.
* If an application is in “no longer being considered” (rejected/accepted) status, remove this job application record from the “Applications” table and insert it into the application archive table.
* The system will keep track of an initial hire date for each employee, as well as a start date for each position the employee holds.

It is assumed that only registered users of the golf cart service can apply for a job. Also, it is assumed that information filled in each job application is not stored in the database.

**Enhanced Entity Relationship Diagram**

For this project, it was necessary to create an enhanced entity relationship diagram (EERD). Within the EERD, entities, or tables, are displayed with associated attributes that help structure the data that is to be populated. Relationships between tables are also displayed. Within an EERD, it is possible to display tables that have a many-to-many relationship associated with it, as well as generalization/specialization requirements for any table. The EERD that was created for this project is displayed below:

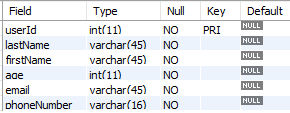


**Figure 2**: Enhanced Entity Relationship Diagram

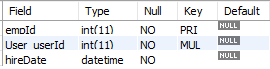
For this EERD, there are nine tables. The information that each table contains is self-explanatory by looking at their attributes. A many-to-many relationship is considered for the “Employee” and “Job” table, where an employee can have 1 or many jobs, and a job can have 1 or more employees. In order to implement the many-to-many relationship, an additional table was created called “empJob”, which consists of the primary keys from the Employee and Job tables. This table is used to list all of the jobs that an employee has within the database, for all employees. In addition to a many-to-many relationship, the empJob table has generalization/specialization associated with it. There are three job types that an employee can have, so the team thought it would helpful to separate this information into three tables called “Driver”, “Mechanic”, and “IT”. There also exists the “Archive” table that is not associated with any of the other tables. This table is used to store application information that are in the “accepted” or “rejected” status, which states that a recruiter has reviewed the application and made a decision. The Archive table is populated using a stored procedure that is executed where applications that have been reviewed in the “Applications” table are populated into the Archive table, and then removed from the Applications table. This is useful in the event a recruiter wants to retrieve an applicant’s information at a later date.

**Data Dictionary**

User table



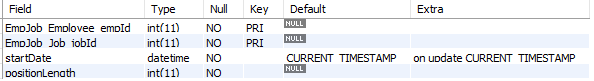
Employee table



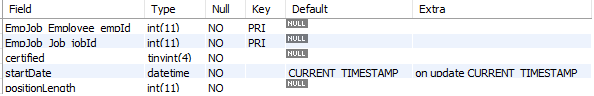
Job table



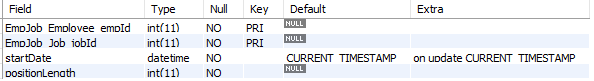
Driver table



Mechanic table



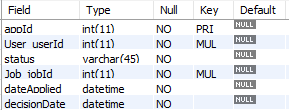
IT table



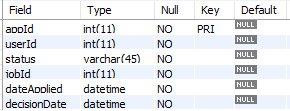
EmpJob table



Applications table



Archive table



**Advanced SQL Overview**

Stored Procedure

A stored procedure was created for inserting records that have a status of “accepted” or “rejected” from the Applications table to the Archive table and then removing those respective records from the Applications table. The Applications table is where a recruiter will see a list of applications that have been already reviewed or still waiting for a final decision to be made. The stored procedure is executed on a schedule of every 2 days. It makes sense to remove records from the Applications table to help make it easier for the recruiter to see what applications need to be considered. It also makes sense to insert records into the Archive in the event a recruiter wants to reference an applicants information at a later time. The SQL code that was created for this stored procedure is displayed below:

|  |
| --- |
| use cartProject;  DELIMITER $$  CREATE PROCEDURE archiveApplications()  BEGIN  DECLARE done INT DEFAULT 0;  DECLARE application\_appId INT;  DECLARE application\_User\_userId INT;  DECLARE application\_status VARCHAR(45);  DECLARE application\_Job\_jobId INT;  DECLARE application\_dateApplied DATETIME;  DECLARE application\_decisionDate DATETIME;  DECLARE applicationRec cursor for select appId, User\_userId, status, Job\_jobId, dateApplied, decisionDate FROM application;  DECLARE continue handler FOR NOT FOUND SET done = 1;    SET SQL\_SAFE\_UPDATES=0;  OPEN applicationRec;  REPEAT  FETCH applicationRec INTO application\_appId, application\_User\_userId, application\_status, application\_Job\_jobId, application\_dateApplied, application\_decisionDate;  IF application\_appId NOT IN (SELECT appId FROM archive) and (application\_status LIKE ('%accepted%') OR application\_status LIKE ('%rejected%'))  THEN  INSERT INTO archive  VALUES(application\_appId, application\_User\_userId, application\_status, application\_Job\_jobId, application\_dateApplied, application\_decisionDate);  END IF;  UNTIL done  END REPEAT;  DELETE FROM application  WHERE status LIKE ('% rejected%')  OR status LIKE ('% accepted %');  SET SQL\_SAFE\_UPDATES=1;  END$$ |

In order for the stored procedure to be executed automatically every 2 days, it is necessary to create an event. The SQL code that was created for creating an event is displayed below:

|  |
| --- |
| SET GLOBAL event\_scheduler = ON;  CREATE EVENT updateApplications  ON SCHEDULE EVERY 2 DAY  DO  CALL archiveApplications(); |

To demonstrate the functionality of the stored procedure, an example was created.

Consider the set of applications that are currently in “accepted” or “rejected” status. These records can be retrieved using the following query:

|  |
| --- |
| SELECT \*  FROM application  WHERE status LIKE '%rejected%' OR status LIKE '%accepted%'; |



Every 2 days, it is desired to move these records from the Applications table to the Archive table. After running the stored procedure, the same query from above can be executed to see that these records are no longer in the Applications table:



In order to verify that the records were successfully inserted into the Archive table, the following query can be executed:

|  |
| --- |
| SELECT \*  FROM archive  WHERE appId >= 106; |

After executing this query, the same records that were originally in the Applications table now exist in the Archive table.

Trigger

A trigger was created for adding a user who has been employed by the recruiter to the Employee table, the empJob table, as well as the respective job type table (i.e., Mechanic, Driver, , IT). Whenever a recruiter sets an applicants status to “accepted” for a particular job application that is associated with any job type, the user is immediately added to the Employee table if they are not already an employee. If an employee has been accepted for additional job, then they are not added to the Employee table again since there is already a record of their employment by the golf cart service. Following this, a record is inserted into the empJob table which allows for the database to keep a record of which jobs each employee has assigned to them. Finally, the user/employee is added to the correct job type table depending on some conditional statements. The SQL code that was created for this trigger is displayed below:

|  |
| --- |
| DELIMITER $$  CREATE TRIGGER acceptedStatusUpdate  AFTER UPDATE ON cartproject.application  FOR EACH ROW  BEGIN  declare maxEmpId INT;  set maxEmpId = (SELECT max(empId) from employee) + 1;  IF new.status LIKE '%accepted%'  THEN  IF old.User\_userId NOT IN (SELECT User\_userID from employee)  THEN  -- add to employee table  INSERT INTO cartproject.employee(empId, User\_userID, hireDate)  VALUES(maxEmpId, old.User\_userId, NOW());  END IF;    -- add to empjob table  INSERT INTO cartproject.empjob  VALUES((SELECT empId FROM cartproject.employee WHERE User\_userId = old.User\_userId),old.Job\_jobId);    -- add to mechanic table  IF old.Job\_jobId = 1  THEN  INSERT INTO cartproject.mechanic(EmpJob\_Employee\_empId, EmpJob\_Job\_jobId, certified, positionLength)  VALUES((SELECT empId FROM cartproject.employee WHERE User\_userId = old.User\_userId), 1, 0, 0);  END IF;  -- add to driver table  IF old.Job\_jobId = 2  THEN  INSERT INTO cartproject.driver(EmpJob\_Employee\_empId, EmpJob\_Job\_jobId, positionLength)  VALUES((SELECT empId FROM cartproject.employee WHERE User\_userId = old.User\_userId), 2, 0);  END IF;    -- add to it table  IF old.Job\_jobId = 3  THEN  INSERT INTO cartproject.it(EmpJob\_Employee\_empId, EmpJob\_Job\_jobId, positionLength)  VALUES((SELECT empId FROM cartproject.employee WHERE User\_userId = old.User\_userId), 3, 0);  END IF;  END IF;  END $$ |

For demonstrating the functionality of the trigger, an example was created. Suppose there is a user with userId of 21 who applied for two jobs, Mechanic and IT (jobId of 1 and 3, respectively). Since there is a conditional statement in the trigger that requires the application to have an “accepted” status, this trigger will not execute if the status is set to “rejected”. There exists no employee in the database with a userId of 21. Currently, their application status is in ‘pending’:



If we change the status for appId of 121 to “accepted”, then it is expected that this user will be added to the Employee table, empJob table, and the Mechanic table. After updating the status to “accepted”, the result is displayed:

Updated status to “accepted”



Record inserted into Employee table



Record inserted into empJob table



Record inserted into Mechanic table



If the same user is also accepted for the other job that they applied to (appId of 109 for jobId 3), then a similar process would be executed, except for the user being added to the employee table:

Updated status to “accepted”



Employee table doesn’t get updated



Record inserted into empJob table



Record inserted into IT table



Update and Delete for One Table

An example for updating a record in the Applications table will be presented. It is simple to write a query for updating an application status. Suppose there is a user with a userId of 90 and submitted an application:



Initially, the application status is set to “not opened”, which means that a recruiter has not opened the particular application yet. Once a recruiter does open this application, it is the recruiter’s responsibility to set the application status to “pending” if a decision isn’t made after opening it for the first time. A query that can be used to update this application’s status to “pending” is below:

|  |
| --- |
| UPDATE application  SET status = "pending "  WHERE appId = 105; |

After executing this query, the application status is set to “pending”:



It is possible for a user to delete their application if they no longer want to be considered for a job. Suppose it is desired to delete the application with appId of 105. The query that can be used to execute this query can be found below:

|  |
| --- |
| DELETE FROM application  WHERE appId = 105; |

After executing this query, this record will no longer exist in the database. The query that can be used to verify that this record is removed from the Application table is below:

|  |
| --- |
| SELECT \*  FROM application  WHERE appId = 105 |

After executing this query, the following result is returned:



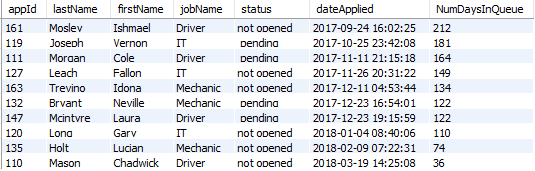
SQL Views for Reports

*Report 1*

The first report that was created can be used by recruiters to determine which applications need to be considered. Within the report, the recruiter can see the application number (appId), name of the applicant (lastName and firstName), the job type (jobName), current status of the application (status), the submission date for the application (dateApplied), and the age of the application (NumDaysInQueue). This is useful so recruiters can better prioritize their work. Conditional statements are included in the SQL that only retrieve applications in “not opened” and “pending” statuses, as well as applications that have an age greater than 30 days. Although there is a stored procedure that removes applications that have already been reviewed, it is still necessary to apply the WHERE clause since the stored procedure isn’t executed everyday. A nested query is used to retrieve records for applications that are completed by users who are not already an employee. This can help new employees get hired by the golf cart service instead of assigning an existing employee an additional job. In order to generate this report, inner joins are necessary to retrieve information that exists across a set of multiple tables. The SQL code that retrieves this information is displayed below:

|  |
| --- |
| SELECT a.appId, b.lastName, b.firstName, c.jobName, a.status, a.dateApplied, datediff(NOW(), a.dateApplied) as NumDaysInQueue  FROM cartproject.application as a  INNER JOIN cartproject.user as b  ON a.User\_userId = b.userId  INNER JOIN cartproject.job as c  ON a.Job\_jobId = c.jobId  WHERE (a.status LIKE ('%not opened%') OR a.status LIKE ('%pending%'))  AND datediff(NOW(), a.dateApplied) >= 30  AND a.User\_userId NOT IN (SELECT User\_userId  FROM employee)  ORDER BY dateApplied; |

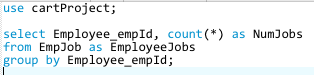
An example as to what the report looks like is displayed below:



*Report 2*

The second report created is one for the employers. They can use it to keep track of all of their employees and how many jobs each employee has. In the report, the employer can see the employee IDs as well as the number of jobs each person has. This can be useful for employers to keep track of how much of a workload each of their employees is taking on. The code and results of the report can be seen below.

Code:



Results:



*Report 3*

Transaction

Indexes

User Interface

**Future Work**

Although all requirements were satisfied for this project, there is always the opportunity to perform additional work at a later date. One way this project can be extended is to allow for files to be saved for each application, which contains more information about the applicant. Information can include a resume, college transcript, references, etc. Another way this project can be extended is to allow users to . A final thought for future work is to