

**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, a strong business need exists 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 and implemented using XAMPP Apache Server on the local host.

The final deliverables for this project consisted of 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:

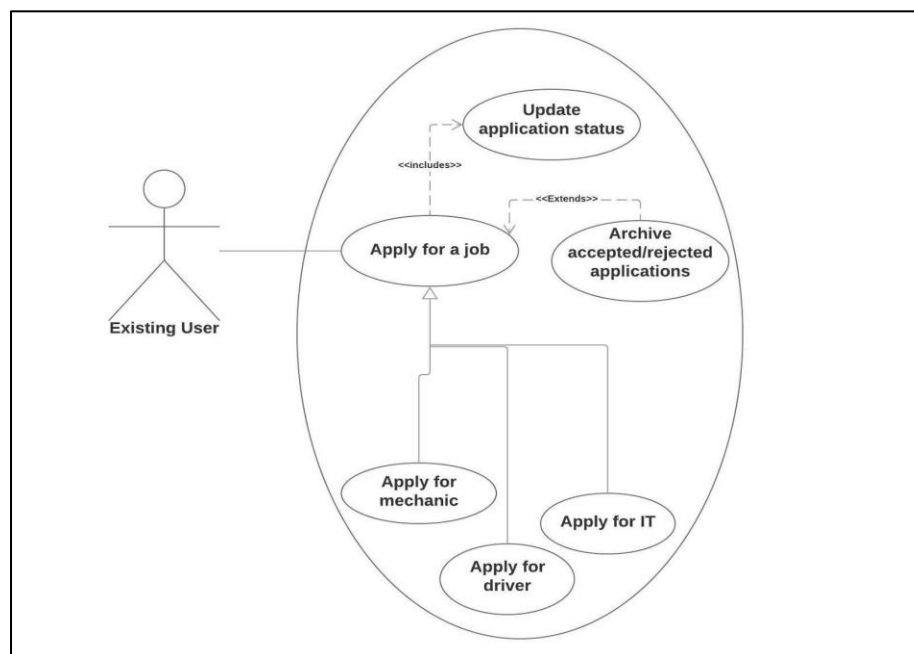


Figure 1: Use Case Diagram

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:

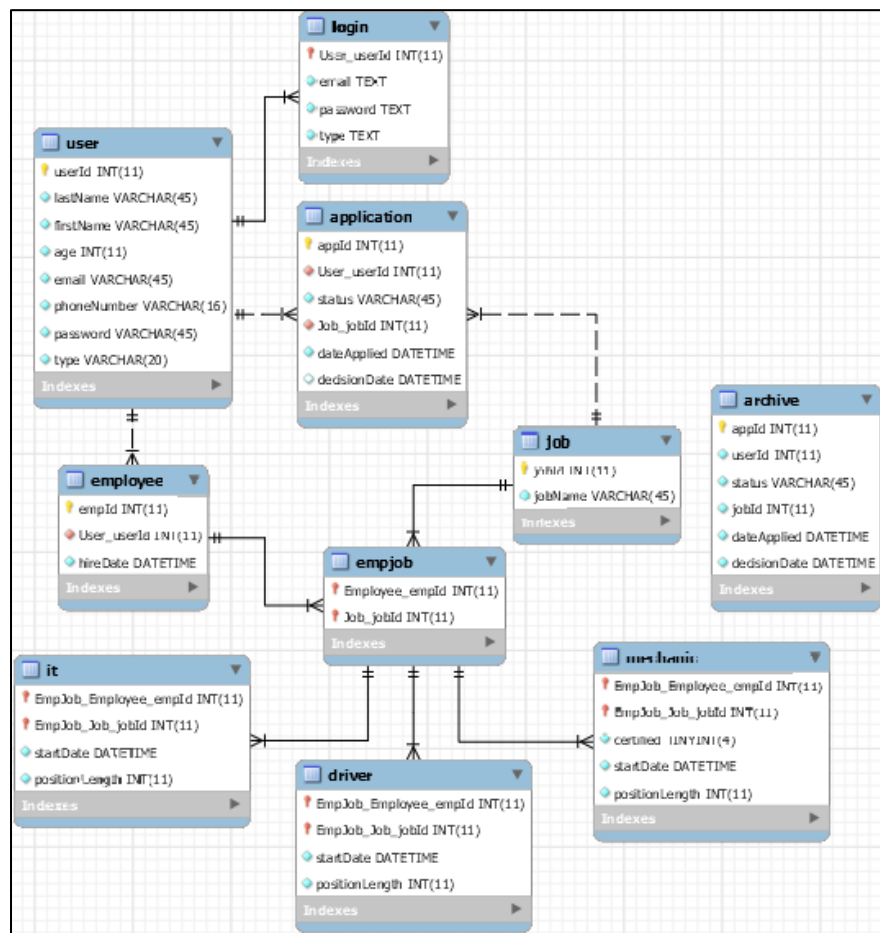


Figure 2: Enhanced Entity Relationship Diagram

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 directly related 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 “Application” table are populated into the Archive table, and then removed from the Application table. This is useful in the event a recruiter wants to retrieve an applicant’s information at a later time.

## Data Dictionary

### User table

Field	Type	Null	Key	Default	Extra
userId	int(11)	NO	PRI	NULL	auto increment
lastName	varchar(45)	NO		NULL	
firstName	varchar(45)	NO		NULL	
age	int(11)	NO		NULL	
email	varchar(45)	NO	MUL	NULL	
phoneNumber	varchar(16)	NO		NULL	
password	varchar(45)	NO		NULL	
type	varchar(20)	NO		NULL	

### Employee table

Field	Type	Null	Key	Default
empId	int(11)	NO	PRI	NULL
User userId	int(11)	NO	MUL	NULL
hireDate	datetime	NO		NULL

### Job table

Field	Type	Null	Key	Default
jobId	int(11)	NO	PRI	NULL
jobName	varchar(45)	NO		NULL

### Driver table

Field	Type	Null	Key	Default	Extra
EmpJob Employee empId	int(11)	NO	PRI	NULL	
EmpJob Job jobId	int(11)	NO	PRI	NULL	
startDate	datetime	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
positionLength	int(11)	NO		NULL	

### Mechanic table

Field	Type	Null	Key	Default	Extra
EmpJob Employee empId	int(11)	NO	PRI	NULL	
EmpJob Job iobId	int(11)	NO	PRI	NULL	
certified	tinyint(4)	NO		NULL	
startDate	datetime	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
positionLength	int(11)	NO		NULL	

### IT table

Field	Type	Null	Key	Default	Extra
EmpJob Employee empId	int(11)	NO	PRI	NULL	
EmpJob Job iobId	int(11)	NO	PRI	NULL	
startDate	datetime	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
positionLength	int(11)	NO		NULL	

### EmpJob table

Field	Type	Null	Key	Default
Employee empId	int(11)	NO	PRI	NULL
Job iobId	int(11)	NO	PRI	NULL

### Application table

Field	Type	Null	Key	Default
appId	int(11)	NO	PRI	NULL
User userId	int(11)	NO	MUL	NULL
status	varchar(45)	NO		NULL
Job iobId	int(11)	NO	MUL	NULL
dateApplied	datetime	NO		NULL
decisionDate	datetime	NO		NULL

### Archive table

Field	Type	Null	Key	Default
appId	int(11)	NO	PRI	NULL
userId	int(11)	NO		NULL
status	varchar(45)	NO		NULL
iobId	int(11)	NO		NULL
dateApplied	datetime	NO		NULL
decisionDate	datetime	NO		NULL

## Advanced SQL Overview

### Stored Procedure

A stored procedure was created for inserting records that have a status of “accepted” or “rejected” from the Application table to the Archive table and then removing those respective records from the Application table. The Application 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 Application 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 applicant’s 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%';
```

appld	User_userId	status	Job_jobId	dateApplied	decisionMade
106	26	accepted	1	6/6/2018 0:02	6/13/2018 0:02
108	64	accepted	1	4/20/2017 18:10	4/27/2017 18:10
112	3	rejected	1	3/3/2018 10:20	3/10/2018 10:20
114	33	accepted	2	5/17/2018 23:23	5/24/2018 23:23
115	15	accepted	1	3/17/2018 15:43	3/24/2018 15:43
117	17	rejected	1	6/27/2018 23:40	7/4/2018 23:40
118	18	accepted	1	10/14/2018 12:45	10/21/2018 12:45
121	21	rejected	1	11/30/2017 1:49	12/7/2017 1:49
123	45	accepted	2	3/21/2018 20:40	3/28/2018 20:40
124	46	accepted	3	8/5/2017 22:00	8/12/2017 22:00
128	50	accepted	1	3/31/2019 19:31	4/7/2019 19:31
130	52	accepted	3	2/15/2019 19:11	2/22/2019 19:11
131	53	accepted	3	11/4/2017 8:17	11/11/2017 8:17
134	56	accepted	3	5/21/2018 3:19	5/28/2018 3:19
137	59	rejected	2	3/2/2018 19:59	3/9/2018 19:59
138	60	rejected	3	12/19/2017 10:10	12/26/2017 10:10
139	38	accepted	1	7/25/2018 0:42	8/1/2018 0:42
140	39	accepted	1	3/31/2018 20:21	4/7/2018 20:21
141	40	accepted	3	5/21/2018 19:34	5/28/2018 19:34
142	41	accepted	2	8/7/2018 14:09	8/14/2018 14:09
144	43	accepted	2	10/25/2017 9:15	11/1/2017 9:15
146	6	rejected	2	10/21/2018 3:51	10/28/2018 3:51
149	55	accepted	1	4/19/2018 17:13	4/26/2018 17:13
150	2	accepted	1	1/15/2019 5:14	1/22/2019 5:14
151	1	accepted	3	5/14/2018 2:09	5/21/2018 2:09
152	30	rejected	3	1/1/2018 14:45	1/8/2018 14:45
153	25	rejected	3	5/11/2017 7:58	5/18/2017 7:58
154	44	accepted	1	10/8/2018 15:14	10/15/2018 15:14
155	3	accepted	3	4/13/2018 14:55	4/20/2018 14:55
158	7	rejected	3	8/29/2017 16:39	9/5/2017 16:39
162	11	accepted	1	11/13/2018 17:25	11/20/2018 17:25
165	14	accepted	2	10/9/2017 17:52	10/16/2017 17:52
167	82	accepted	3	11/17/2018 4:06	11/24/2018 4:06
170	85	accepted	3	9/1/2017 21:27	9/8/2017 21:27

Every 2 days, it is desired to move these records from the Application 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 Application table:

appId	User_userId	status	Job_jobId	dateApplied	decisionDate
NULL	NULL	NULL	NULL	NULL	NULL

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 Application 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 applicant's 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’:

appId	User_userId	status	Job_jobId	dateApplied
109	21	pending	3	2018-10-15 09:01:02
121	21	pending	1	2017-11-30 01:49:39

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”

appId	User_userId	status	Job_jobId	dateApplied
109	21	pending	3	2018-10-15 09:01:02
121	21	accepted	1	2017-11-30 01:49:39

Record inserted into Employee table

empId	User_userId	hireDate
1040	21	2018-04-24 10:31:34

Record inserted into empJob table

Employee_empId	Job_jobId
1040	1

Record inserted into Mechanic table

EmpJob_Employee_empId	EmpJob_Job_jobId	certified	startDate	positionLength
1040	1	0	2018-04-24 10:31:34	0

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”

appId	User_userId	status	Job_jobId	dateApplied
109	21	accepted	3	2018-10-15 09:01:02
121	21	accepted	1	2017-11-30 01:49:39

Employee table doesn’t get updated

empId	User_userId	hireDate
1040	21	2018-04-24 10:31:34

Record inserted into empJob table

Employee_empId	Job_jobId
1040	1
1040	3

Record inserted into IT table

EmpJob_Employee_empId	EmpJob_Job_jobId	startDate	positionLength
1040	3	2018-04-24 10:36:36	0

### Update and Delete for One Table

An example for updating a record in the Application 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:

appId	User_userId	status	Job_jobId	dateApplied	decisionDate
105	90	not opened	3	2017-09-11 10:52:14	NULL

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":

appId	User_userId	status	Job_jobId	dateApplied	decisionDate
105	90	pending	3	2017-09-11 10:52:14	NULL

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:

appId	User_userId	status	Job_jobId	dateApplied	decisionDate
NULL	NULL	NULL	NULL	NULL	NULL

## 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 every day. 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) >= 0
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:

appId	lastName	firstName	jobName	status	dateApplied	NumDaysInQueue
161	Moslev	Ishmael	Driver	not opened	2017-09-24 16:02:25	212
119	Joseph	Vernon	IT	pending	2017-10-25 23:42:08	181
111	Morgan	Cole	Driver	pending	2017-11-11 21:15:18	164
127	Leach	Fallon	IT	not opened	2017-11-26 20:31:22	149
163	Trevino	Idona	Mechanic	not opened	2017-12-11 04:53:44	134
132	Brvant	Neville	Mechanic	pending	2017-12-23 16:54:01	122
147	Mcintvre	Laura	Driver	pending	2017-12-23 19:15:59	122
120	Lona	Garv	IT	not opened	2018-01-04 08:40:06	110
135	Holt	Lucian	Mechanic	not opened	2018-02-09 07:22:31	74
110	Mason	Chadwick	Driver	not opened	2018-03-19 14:25:08	36

\*\*Note that report 1 was implemented in the web application.

## 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:

```
use cartProject;

select Employee_empId, count(*) as NumJobs
from EmpJob as EmployeeJobs
group by Employee_empId;
```

Results:

	Employee_empld	NumJobs
▶	1000	2
	1001	1
	1002	1
	1003	1
	1004	1
	1005	1
	1006	1

### *Report 3*

Report 3 was designed to give the number of rejected applications for 2017 (the previous year). This will assist administrators in being able to reference the number of people that applied and were not given a position in 2017. The code and results can be seen below.

Code:

```
SELECT COUNT(*)
FROM
  (SELECT DISTINCT userId
   FROM archive
   WHERE status LIKE '%rejected%'
   AND decisionDate > 2017-01-01) AS archive;
```

Results:

	COUNT(*)
	3

### Transaction

The transactions for this project were implemented on the administrator side of the web application, in the event of update or delete. This is important because it provides for a database rollback in the event of concurrency issues during writes to the database. The code for both the update and delete transactions can be seen below. Note that the language is php/mysql using PDO.

## Update

```
// receive all input values from the form. Call the e() function
// defined below to escape form values
$last = isset($_POST['lastName']) ? e($_POST['lastName']) : "";
$first = isset($_POST['firstName']) ? e($_POST['firstName']) : "";
$appId = isset($_POST['appId']) ? e($_POST['appId']) : "";
$status = isset($_POST['taskOption']) ? e($_POST['taskOption']) : "";
$decisionDate = date("Y-m-d H:i:s", time());

//create query
$update_statement = "UPDATE Application set status = :status, decisionDate = :decisionDate WHERE appId = :id";

$update = $connection -> prepare($update_statement);
$update -> bindparam(':id', $appId);
$update -> bindparam(':status', $status);
$update -> bindparam(':decisionDate', $decisionDate);

try {
    $connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $connection->beginTransaction();
    $update->execute();
    $connection->commit();
    $_SESSION['success'] = "Application " . " for " . $first . " " . $last . " successfully updated";
    header('location: adminView.php');

} catch (Exception $e) {
    $connection->rollBack();
    echo "Failed: " . $e->getMessage();
}
```

## Delete

```
// receive all input values from the form. Call the e() function
// defined below to escape form values
$last = isset($_POST['lastName']) ? e($_POST['lastName']) : "";
$first = isset($_POST['firstName']) ? e($_POST['firstName']) : "";
$appId = isset($_POST['appId']) ? e($_POST['appId']) : "";

//create query
$delete_statement = "DELETE FROM Application WHERE appId = :id";

$update = $connection -> prepare($delete_statement);
$update -> bindparam(':id', $appId);

try {
    $connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $connection->beginTransaction();
    $update->execute();
    $connection->commit();
    $_SESSION['success'] = "Application " . " for " . $first . " " . $last . " successfully deleted";
    header('location: adminView.php');

} catch (Exception $e) {
    $connection->rollBack();
    echo "Failed: " . $e->getMessage();
}
```

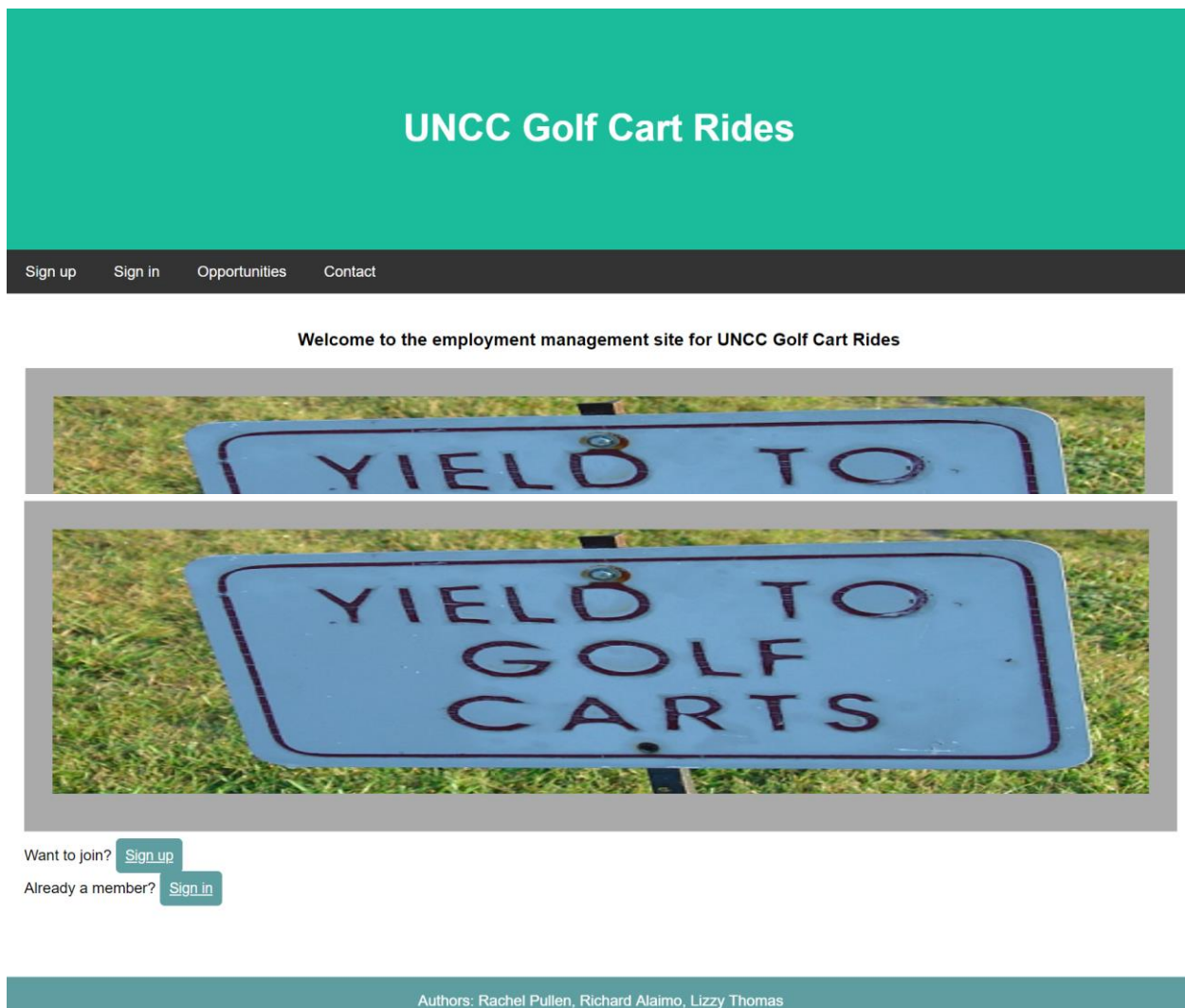
## Indexes

For the use of the web application create, read, update, and delete functionality, indexes were added to the User and Application table. In addition to having indexes on its primary and foreign keys, the Application table has indexes “app\_user” on field User\_userId and “app\_userId” on fields User\_userId and app\_appId. In addition to having indexes on its primary and foreign keys, the User table has indexes “email\_lookup” on field email and “login” on fields email, password. Indexes are generated in the SQL dump files.



## User Interface

The user interface for our project was implemented in the form of a web application, using php MySQL with PDO. The web application was implemented using XAMPP Apache Server on the local host. It is designed to be a portal for job application management. Ordinary users may register with the site, login, and view their existing application status, as well as apply for another position. They may also view descriptions for the possible jobs and a contact information page. Administrators may login and view all applications in the system. They can select to view only the applications that are still pending. Administrators may also update the status of an application, delete an application, or register a user. See screenshots below for a demonstration of the web application. Source code files are included in the project deliverables.



*Figure 3 and 4: Homepage for the web application.*

# UNCC Golf Cart Rides

[Sign up](#)[Sign in](#)[Opportunities](#)[Contact](#)

Register

First Name

Last Name

Email

Age

Phone Number (xxx) xxx-xxxx

Password

Confirm password

Register

Already a member? [Sign in](#)

Figure 5: Upon clicking “sign up,” the user can register an account with the site. Let’s assume a user “John Smith” registers here.


# UNCC Golf Cart Rides

[Opportunities](#)[Apply for a Job](#)[View Application Status](#)[User Home](#)

Welcome John Smith !

Home Page

You are now logged in

 domain@email.com (User)[logout](#)

[back to home](#)

Figure 6: After registering (or logging in) the user will be taken to his or her home page. Here shown is John Smith’s page.

[Sign up](#)[Sign in](#)[Opportunities](#)[Contact](#)

## Job Opportunities

**We have a number of opportunities available. There's a place for everyone on our team!**

### Driver

We need licensed, responsible drivers to drive our customers.

### IT Associate

IT associates will assist in management, maintenance, and update of our technology

### Mechanic

Mechanics will provide assistance in the event of golf cart failure as well as perform regular check ups to the carts. Certification is preferred.

[Apply Now!](#)[Back to home](#)

Figure 7: Upon selecting “opportunities,” the user can see which jobs are available to apply for. He or she can also click the link to go directly to the apply page.

## UNCC Golf Cart Rides

[Opportunities](#)[Apply for a Job](#)[View Application Status](#)[User Home](#)

### Apply for a job:

We're delighted you want to join our team! Please select the job for which you would like to apply.

Mechanic ▼ Submit

[Back to user home](#)

Authors: Rachel Pullen, Richard Alaimo, Lizzy Thomas  
(For educational purposes) University of North Carolina at Charlotte, Spring 2018  
Contact Us: [Contact Information](#)

Figure 8: Applying for a job. Since the user is already logged in, the only information he or she must select is the type of job. Let's assume that John Smith applies for “mechanic.”

Application status for John Smith :

Results

First Name	Last Name	Date Applied	Job Name	Current Status
John	Smith	2018-04-30 23:43:51	Mechanic	not opened

[Back to user home](#)

Figure 9: Upon visiting the application status page, John Smith can now view his application, which has not yet been opened by the administration.

Login

Email

ethoma53@uncc.edu

Password

...

Login

Not yet a member? [Sign up](#)

Figure 10: Administrator login

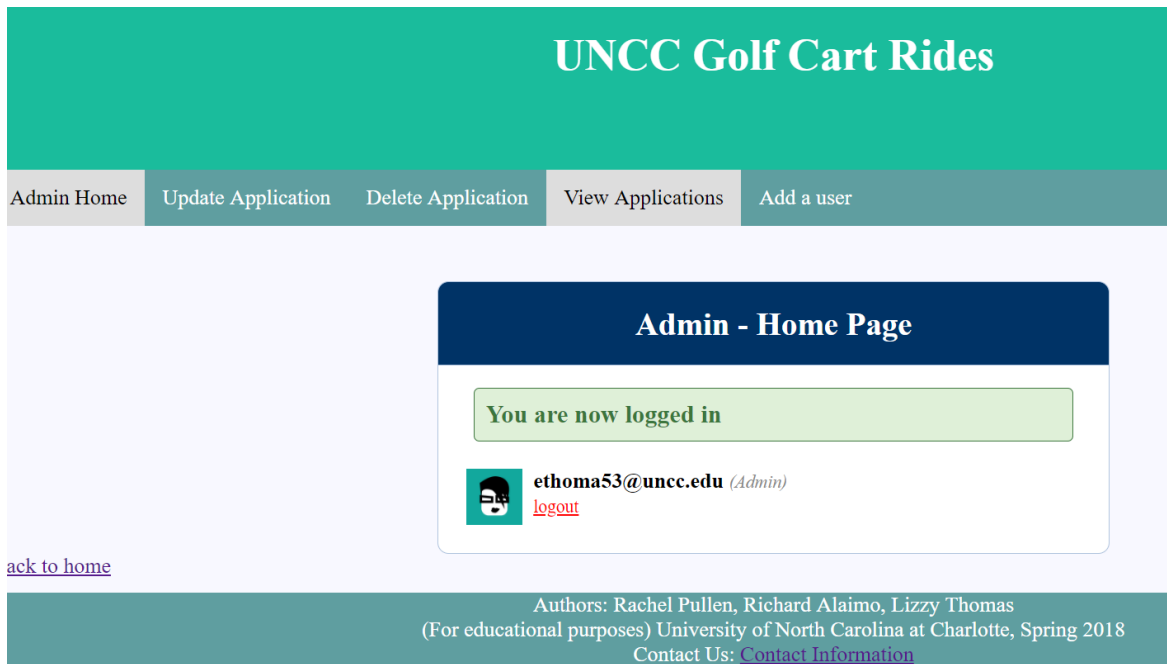


Figure 11: After logging in, the administrator will see his or her admin page.

Application #	First Name	Last Name	Date Applied	Job Name	Current Status	Decision Date
101	Zachery	Dillard	2017-08-16 06:15:26	Mechanic	pending	N/A
105	Angelica	French	2017-09-11 10:52:14	IT	not opened	N/A
106	Aladdin	Waller	2018-06-06 00:02:19	Mechanic	accepted	2018-06-13 00:02:19
107	Micah	Richmond	2018-09-04 21:34:35	Driver	pending	N/A
108	Vance	Rivas	2017-04-20 18:10:51	Mechanic	accepted	2017-04-27 18:10:51
109	Vladimir	Velasquez	2018-10-15 09:01:02	IT	pending	N/A
110	Chadwick	Mason	2018-03-19 14:25:08	Driver	not opened	N/A
111	Cole	Morgan	2017-11-11 21:15:18	Driver	pending	N/A
112	Zachery	Bruce	2018-03-03 10:20:48	Mechanic	rejected	2018-03-10 10:20:48
113	Cruz	Dawson	2019-04-02 20:01:34	Mechanic	pending	N/A
114	Clementine	Meyer	2018-05-17 23:23:43	Driver	accepted	2018-05-24 23:23:43
115	Kevin	Warren	2018-03-17 15:43:28	Mechanic	accepted	2018-03-24 15:43:28
116	Cameran	Wolfe	2018-11-04 09:50:48	IT	not opened	N/A
117	Octavia	Soto	2018-06-27 23:40:56	Mechanic	rejected	2018-07-04 23:40:56
118	Chadwick	Mason	2018-10-14 12:45:50	Mechanic	accepted	2018-10-21 12:45:50

Figure 12: If the administrator selects “View applications,” he or she can see all applications in the system.

169	Wayne	Barnett	2017-11-30 05:06:50	Driver	pending	N/A
170	Alden	King	2017-09-01 21:27:56	IT	accepted	2017-09-08 21:27:56
171	John	Smith	2018-04-30 23:43:51	Mechanic	not opened	N/A

Figure 13: Scrolling down the list, John Smith's application is viewable.

Admin Home
Update Application
Delete Application
View Applications
Add a user

Click to see only pending applications

Click to see all applications

## Results

Application #	First Name	Last Name	Date Applied	Job Name	Current Status	Decision Date	Number of days in the Queue
156	Reuben	Matthews	2017-04-06 11:05:07	Mechanic	not opened	N/A	389
148	Kevin	Warren	2017-04-06 21:54:00	IT	pending	N/A	389
122	Echo	Aguirre	2017-04-16 00:54:40	Driver	not opened	N/A	379
129	Emerson	Hurley	2017-06-11 07:36:44	Driver	not opened	N/A	323
101	Zachery	Dillard	2017-08-16 06:15:26	Mechanic	pending	N/A	257
161	Ishmael	Mosley	2017-09-24 16:02:25	Driver	not opened	N/A	218
119	Vernon	Joseph	2017-10-25 23:42:08	IT	pending	N/A	187
111	Cole	Morgan	2017-11-11 21:15:18	Driver	pending	N/A	170
127	Fallon	Leach	2017-11-26 20:31:22	IT	not opened	N/A	155
163	Idona	Trevino	2017-12-11 04:53:44	Mechanic	not opened	N/A	140

Figure 14: Administrators can also select "click to see only pending applications, which will display only the applications that have not been decided upon yet (that is, "not opened" and "pending").

Please enter the information for the application you wish to update, then choose the status you wish to input.

Admin - Update Application Status

First Name

Last Name

Application Number

Application Status

Figure 15: Upon selecting "update application," an administrator can fill out the form to change the status of a given application. Let's suppose the administrator wants to update John Smith's application to "rejected."

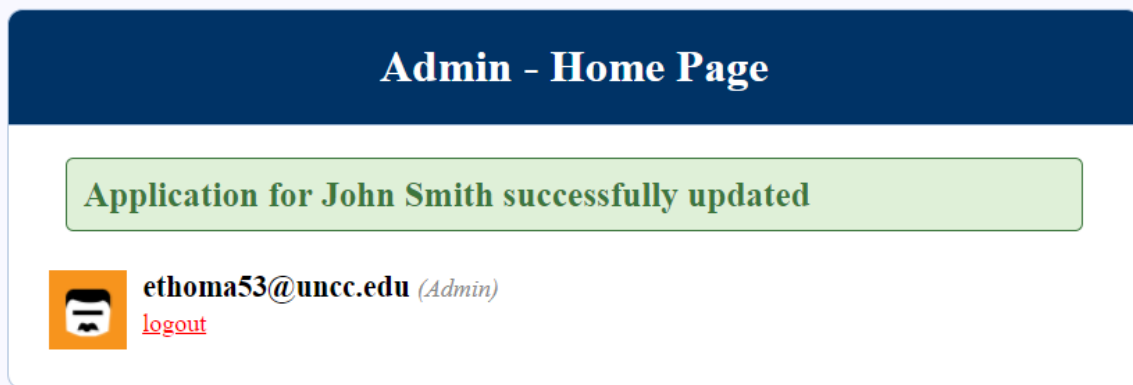
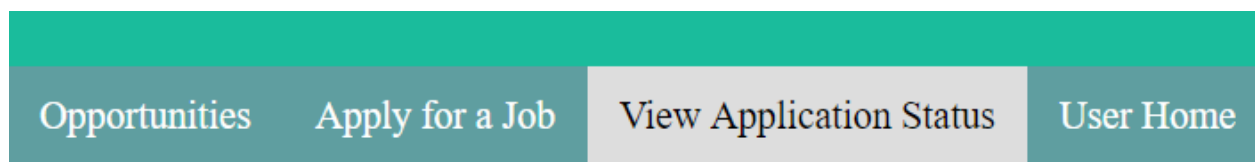


Figure 16: After the administrator selects “update status,” the page will return to his or her home screen and a message will be displayed to confirm that the status has been updated.



## Application status for John Smith :

### Results

First Name	Last Name	Date Applied	Job Name	Current Status
John	Smith	2018-04-30 23:54:03	Mechanic	rejected

[Back to user home](#)

Figure 17: If we return to John Smith’s account, we can now see that his “view application status” reflects the change made by the administrator.

Please enter the information for the application you wish to delete.

### Admin - Delete Application

First Name

John

Last Name

Smith

Application Number

171

Delete Application

Figure 18: Returning to the administrator account, let's suppose that the administrator navigates to "delete application" and fills out the form to delete John Smith's application.

## Admin - Home Page

Application for John Smith successfully deleted


 **ethoma53@uncc.edu** (Admin)  
[logout](#)

Figure 19: After selecting "delete application," the administrator will be returned to his or her home screen, and a message of confirmation will be displayed.

Opportunities

Apply for a Job

View Application Status

User Home

## Application status for John Smith :

No application results found.

[Back to user home](#)

Figure 20: Returning to John Smith's account, we can see that the change is reflected on the "view application status page."



Admin HomeUpdate ApplicationDelete ApplicationView ApplicationsAdd a user

Admin - create user

First Name

Last Name

Email

User type

User

Age

Phone Number (xxx) xxx-xxxx

Password

Confirm password

Create user

Figure 21: The administrator can also create a user for the website and select whether that user is administrator or user.

UNCC Golf Cart Rides

Sign upSign inOpportunitiesContact

Contact Us

Rachel Pullen

Some text...

Richard Alaimo

Some text...

Lizzy Thomas

Some text...

[Back to home](#)

Figure 22: Contact us page (to be completed).

## **Future Work**

Although all requirements were satisfied for this project, there is always the opportunity to perform additional work. 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. Additionally, the look and feel of the website could be improved. For example, the contact page could be updated with photos and contact information. A form could be provided where users could submit “contact us” messages directly on the site, which could be saved and sent to the administrator. Also, the website could be integrated with the site for the actual golf cart rides, so that applying for a job could take place on the same website as booking a ride. These are all potential futures for our work. Finally, having the entire database system would be ideal in the future, so planning how to implement it in stages could be considered.