SOFE 3700U Data Management Systems

RFID Parking System Database

FINAL REPORT

Group 4 - Contributors

Student Name	Student ID
Scott McLean	100379538
Mohannad Abdo	100523158
Mirna Zohiry	100535658
Dominick Mancini	100517944

Abstract

The RFID parking system is an advanced parking management system designed to allow seamless entry and exit of a parking lot featuring the implementation. This parking system must keep track of all parking lot users and any information regarding them. As such, this facilitates the necessity to implement a centralized database that the parking system references for information on a given user, as well, this database will be available to access from a remote web server. This way, users of the parking lot can access their account, or register a new one, and observe details about their account. This report will focus on the design implementation, and use of the database system associated with the RFID parking system, and how it can be used to improve parking systems that are currently in public use.

1. Introduction

The RFID parking system is a next-generation parking system that uses microcontrollers, RFID technology, and a laser detection system to manage admission and exit of a parking lot. This system allows a patron to enter and exit a parking lot without the need to scan a card, or even stop for more than a second. When a user approaches the entrance to a lot, a polling laser detection system will scan for the presence of a vehicle, and a long-range RFID scanner will also poll for a valid tag in its vicinity.

When a vehicle is detected by the RFID parking system, and the RFID reader scans a tag, it will check the RFID tag against the central database. If there is a user associated with that tag, then the database will search for that user's blacklist status, and if they are entering the correct lot to which they are assigned, and allow entry if the user is not blacklisted and at the correct lot, but deny entry if

the user is blacklist (which can be set manually if the user has committed some offence against the administrator of the parking system) or at the wrong lot. If entry is permitted, a log is taken of the time entry. If entry is denied, then a log is taken of the time of attempted (but denied) entry. Upon exit, the same process is repeated, however, the event indicating time of exit will also include a balance change; a value determined relative to the time between the entry and exit of the lot. This balance change will be added to the account balance of the user in question.

System level operation only requires the above components of the database, however, the database contains further information that users will be able to access through a web server connected to the database. This information includes phone numbers, the name associated with the account, a credit card hash value, and user's vehicle(s) and license plate number(s).

2. Relation to Other Work

This parking system is similar to other systems that require user authentication through a card scan before entry is permitted, much like the parking lots on the UOIT/Durham College north Oshawa Campus, however, it is more convenient than this system as it does not require the user to stop for very long or lower their window and reach out to a machine to validate their card. As well, this system dynamically provides a fee depending on duration of parking, rather than a static fee as is normally the case with a parking pass.

The implementation of this system and database can be compared to that of the Metrolinx Presto Card. There is an account associated with the card, and a charge is applied to the account based on when they boarded a particular vehicle (train or bus), and when they departed the vehicle. This details of this account can be viewed remotely from a web server where card details and balance are provided for users to view. However, this system is different as the web server holds much more information for users, and relates to a parking lot, rather than different bus routes. As well, in practice, the system does not require a card swipe or tap, as is the case with the Presto Card.

3. Main Body of Work

3.1 Design and Implementation

3.1.1 Database Implementation

The database for the parking system is created using SQL and maintained through MySQL server and the phpMyAdmin utility. The database consists of nine relations that are all interconnected with each other and provide a number of details about users of the parking system, as well as general details about the parking configuration that the system is implemented on. The primary relation of the database is the _User relation, as it links to most other components. For details on the layout of the database system, please see section 3.3.

3.1.2 Web Server Implementation

The website for this parking system database makes use of HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) for the front end. For the purposes of ease when accessing the site from a mobile device, Twitter Bootstrap framework is employed, and this has the added benefit of providing many features that are built into this framework.

In the backend, PHP is used to connect to the SQL parking database, and is used for Database queries, such as retrieval, insertion, deletion and the update of data.

The home page of the website is a login page. Users can sign in with their first and last name, as well as their user ID. If a new user does not have an account, there is the ability to register a new account from this login page. Once logged in, the user is redirected to a profile webpage, where they can view their account balance, vehicles tied to the account, blacklist status, and other information relevant to the user. The user also has the option to add new cars to their account.

3.2 Database Relations

3.2.1 Blacklist

This relation keeps track of a user's blacklist status. It contains two attributes;

- **Primary Key** → AcctNo: A user's account number
- State: The blacklist status of the specified user

3.2.2 CreditCard

This relation contains a hashed credit card number for each user of the parking system, there are two attributes:

- **Primary Key** → AcctNo: User's account number
- **Primary Key** → CardNumHash: A hashed credit card number

3.2.3 EventLog

This relation contains all logged events of the parking system, regardless of whether they are related to a specific user account or not. There are 5 attributes:

- **Foreign Key** → AcctNo: User's account number, references AccountNumber in the _User relation, if the event concerns a particular user
- **Primary Key** → EventNum: Event number that uniquely identifies the event occurrence (ordered chronologically)
- Message: A string containing a brief description of the event
- **Primary Key** → Time: The date and time of the event's occurrence
- **Foreign Key** → LotNo: The Number of the lot where the event occurred at; references LotNo in the ParkingLot relation

3.2.4 Lot

This relations connects a user to one or more lot numbers; in this parking system, some users may have privileges to park at multiple lots. The primary lot is kept in the _User relation, however, all lots accessible to the user are kept in this relation.

- **Primary Key** → AcctNo: User's account number
- **Primary/Foreign Key** → LotNo: The parking lot number; references LotNo in the ParkingLot relation.

3.2.5 ParkingLot

This relation contains information on all parking lots that the RFID implementation is managing, there are three attributes:

- **Primary Key** → LotNo: The number of the lot (for internal database use)
- Location: The location of the lot on the premises, where applicable (i.e., North Side)
- LotName: The name of the lot, where applicable (i.e, Hafeez Lot)

3.2.6 Tag

This relation contains all RFID tags that the parking system is currently in possession of. If they are assigned a user, then the user account number is paired with the tag in this relation. There are two attributes.

- **Primary Key** → TagNo: The RFID Tag's unique tag ID
- **Foreign Key** → AcctNo: User's account number, if there is a user assigned to that tag. References AccountNumber in the _User relation.

3.2.7 TransactionLog

This relation can be considered a subset of sorts of the EventLog relation; all events stored in this relation pertain to balance changes for users of the parking system when they exit the lot. There are four attributes:

- **Primary Key** → TransNum: Unique transaction ID, generated when a transaction occurs.
- **Foreign Key** → acctNum: User's account ID, references AccountNumber in the _User relation
- Balance Change: A change in balance as a result of the transaction
- **Foreign Key** → EventNum: Event number, references EventNum in the EventLog relation, and uniquely identifies transaction in that relation, but also links it to the TransactionLog relation

3.2.8 Vehicle

This relation lists data about a user's vehicle(s). There are three relations:

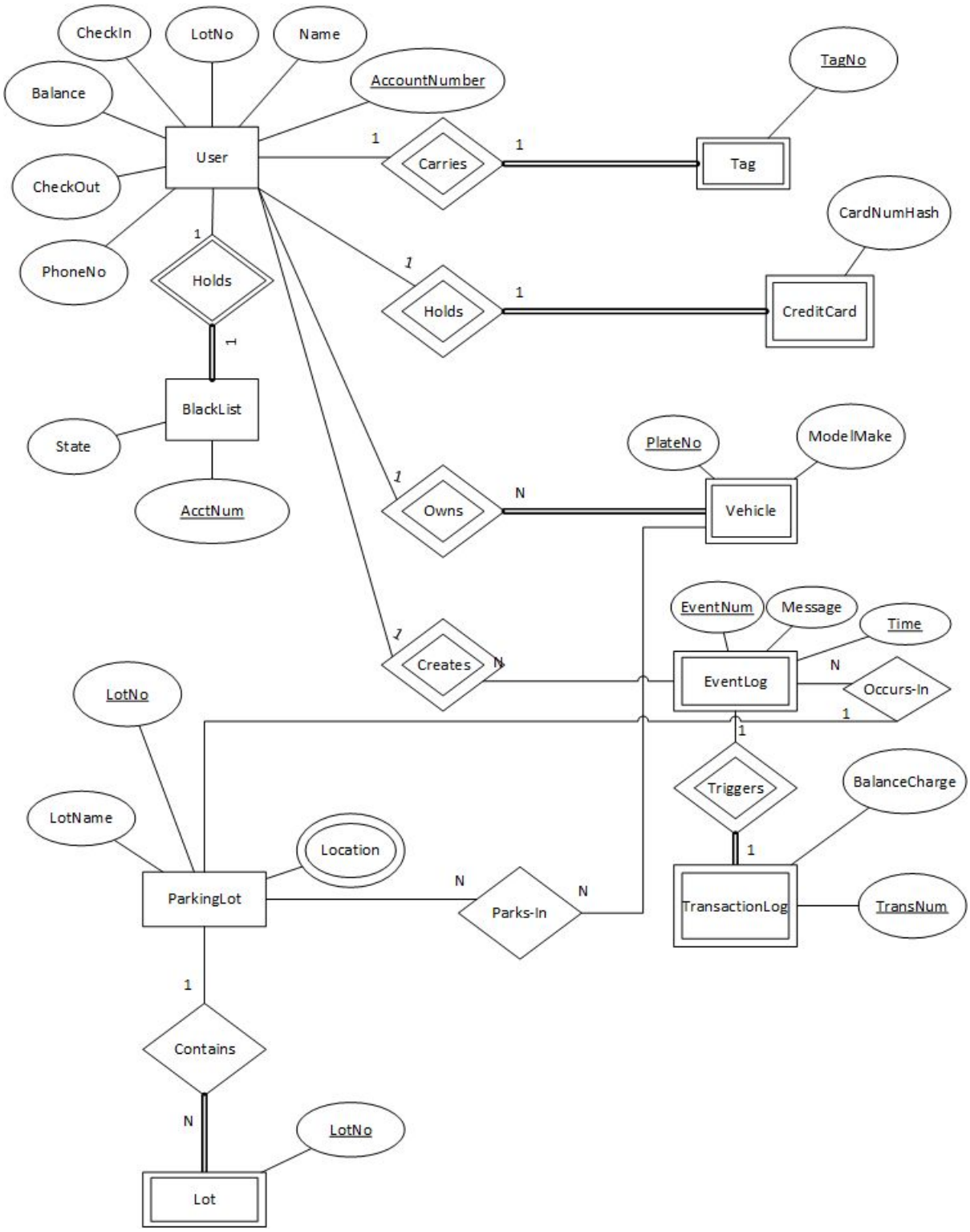
- **Primary Key** → PlateNo: The vehicle's registration/plate number
- ModelMake: The make and model of the vehicle
- **Primary/Foreign Key** → AcctNum: User's account number, references AccountNumber in the _User relation.

3.2.9 _User

This relation is by far the most important relation of the database; it links all other relations together, and includes important information about every user of the database. There are seven attributes:

- **Primary Key** → AccountNumber: The user's account number
- Name: The user's first and last name
- PhoneNo: The user's primary telephone number
- **Foreign Key** → LotNo: The number of the user's primary parking lot (the user may use other lots), references LotNo in the ParkingLot relation
- Balance: User's account balance. May be positive if account is pre-loaded, or negative, if there is an amount owing.
- CheckIn: The time and date of the last entry into a lot
- CheckOut: The time and date of the last exit from a lot

3.3 Schematic and Design Diagrams



3.4 SQL Queries and Views

```
--1. Show the name of the account holder, the plate number and make of the car, and
the lot paired to that account holder. (View 1)
DROP VIEW uservehiclelot;
CREATE VIEW uservehiclelot
AS SELECT u.Name, v.PlateNo, v.ModelMake, l.LotName
FROM _User u, Vehicle v, parkinglot l, lot o
WHERE l.LotNo = u.LotNo AND o.LotNo = l.LotNo
AND u.AccountNumber = o.AcctNo
AND v.AcctNum = u.AccountNumber;
```

+ Options

	Name	PlateNo	ModelMake	LotName
Edit Copy Delete	Dominick Mancini	BHYL 435	Nova LF-S	Greenview Lot A
Edit Copy Delete	Mirna Zohiry	BBCX 858	Chevrolet Impala	Greenview Lot A
Edit Copy Delete	Mohannad Abdo	BBCD	Chevrolet Malibu	Greenview Lot B
Edit Copy Delete	Mohannad Abdo	BXWK 040	GMC Acadia	Greenview Lot B
Edit Copy Delete	Joseph Evelyn	AFDF 222	Porsche Carrera GT	Greenview Lot B
Edit Copy Delete	Joseph Evelyn	ASDF 123	Lamborghini Huracan	Greenview Lot B
Edit Copy Delete	Khalid Hafeez	BBBX 859	Lada Niva	Hafeez Lot
Edit Copy Delete	Scott McLean	ARCW 866	Honda Civic	Hafeez Lot
Edit Copy Delete	Jon An	ASHL 888	Nissan Altima	That Lot
Edit Copy Delete	Troy Scrivens	2V2982	Chevrolet Uplander	Quoderatdemonstrandum
Edit Copy Delete	Troy Scrivens	659 4YH	Bugatti Veyron 16.4	Quoderatdemonstrandum

```
--2. Get a list of all accounts whose phone number is not in the 905 area code
(long-distance commuters).
CREATE VIEW LONGDISTANCE
AS SELECT u.Name, u.PhoneNo, v.PlateNo, v.ModelMake
FROM _User u, Vehicle v
WHERE u.AccountNumber = v.AcctNum AND u.PhoneNo NOT LIKE '%905%';
```

Name	PhoneNo	PlateNo	ModelMake
Nick Me	90123124	Jokq 123	*NULL*
Troy Scrivens	613-539-0000	2V2982	Chevrolet Uplander
Troy Scrivens	613-539-0000	659 4YH	Bugatti Veyron 16.4
Dominick Mancini	289-675-0000	BHYL 435	Nova LF-S
Jon An	705-313-8888	ASHL 888	Nissan Altima
Joseph Evelyn	705-652-1111	AFDF 222	Porsche Carrera GT
Joseph Evelyn	705-652-1111	ASDF 123	Lamborghini Huracan

```
--3. FULL JOIN : Vehicles and _User (View 4)
CREATE VIEW JOINUSERVEHICLE
AS SELECT *
FROM _User
FULL JOIN Vehicle
ON _User.AccountNumber = Vehicle.AcctNum
GROUP BY _User.AccountNumber;
```

MariaDB (MySQL) refuses to accept this query.

```
--4. Select a list of all blacklisted
CREATE VIEW VIEWBLACKLISTED
AS SELECT u.AccountNumber, u.Name, u.PhoneNo, b.State
FROM _User u, Blacklist b
WHERE u.AccountNumber = b.AcctNo AND b.State = 2;
```

AccountNumber	Name	PhoneNo	State
100379538	Scott McLean	905-809-0000	2
100517944	Dominick Mancini	289-675-0000	2
100528555	Joseph Evelyn	705-652-1111	2

```
--5. Select all events except where account number not linked with Khalid Hafeez.
CREATE VIEW EVENTSNOTKHALID
AS SELECT e.AcctNo, e.EventNum, e.Message, e.Time, pl.LotName
FROM EventLog e, ParkingLot pl, _User u
WHERE e.LotNo = pl.LotNo AND u.Name NOT IN (SELECT usr.Name FROM _User usr);
```

THIS VIEW RETURNED NULL FOR THE CURRENT DATABASE POPULATION

```
--6. Select the _User with the greatest charge (charges are negative). >= ALL (View
2)
CREATE VIEW GREATESTCHARGE
AS SELECT u.Name, t.acctNum, t.BalanceChange
FROM TransactionLog t, _User u
WHERE t.BalanceChange <= ALL (SELECT ts.BalanceChange
FROM TransactionLog ts) AND t.acctNum = u.AccountNumber;
```

Name	acctNum	BalanceChange
Mohannad Abdo	100523158	-15.45

```
--7. Select the _User with the most number of unauthorized accesses on Lot number
3. (View 3)
CREATE VIEW UNAUTHORIZED3
AS SELECT u.Name, e.AcctNo, e.EventNum, e.Time, e.LotNo, COUNT(e.eventNum) AS
`Unauthorized Attempts`
FROM _User u, EventLog e, EventLog se
WHERE u.AccountNumber = e.AcctNo AND e.Message LIKE '%Unauthorized Access%';
```

Name	AcctNo	EventNum	Time	LotNo	Unauthorized Attempts
Dominick Mancini	100517944	2	2015-10-25 10:22:00	3	10

```
--8. Select transactions corresponding to a User
CREATE VIEW TRANSACTIONSUSER
AS SELECT tl.TransNum, tl.BalanceChange, SUM(tl.BalanceChange) AS `Total`
FROM EventLog e, TransactionLog tl, _User u
WHERE u.AccountNumber = e.AcctNo AND e.EventNum = tl.EventNum;
```

TransNum	BalanceChange	Total
1	-7.68	-44.24999952316284

```
--9. Select an account number and corresponding Credit Card Hash
CREATE VIEW CREDITHASH
AS SELECT u.AccountNumber, cc.CardNumHash
FROM CreditCard cc, _User u
WHERE u.AccountNumber = cc.AcctNo;
```

AccountNumber	CardNumHash
100517944	#12345456788
100517944	#12345456789
100535658	#580004585
100535658	#580004586
100535658	#580004587
100523158	#580004321
100528555	#852741963
100000000	#159357468
100379538	#987654321
100525222	#457812177
100515638	#013794682
100515638	#013794688

```
--10. Get a list of _User account numbers who have not yet used the parking
system/created any events
CREATE VIEW SELECTNEWUSERS
AS      SELECT u.AccountNumber
        FROM _User u
        WHERE u.AccountNumber NOT IN (SELECT e.AcctNo FROM EventLog e);
```

THIS VIEW RETURNED NULL FOR THE CURRENT DATABASE POPULATION

All SQL used to actually create the original database can be found in this ZIP archive in the file “parkingsystem.sql”

3.5 Goals of Project

The goal of the project is to design an automated parking control system which can capture user data. The data itself is auditable by the company implementing the system and may be available to clients depending on the company’s policy. The software system can easily be extended for data-mining and to provide heuristics.

3.6 Relationship to the Course

This database system utilizes PHP, CSS, and a MySQL database, which uses SQL to retrieve records from the database file. The SQL queries which can be performed on the database are sufficiently complex, though complex queries are not necessarily required for the records to be retrieved. Specific views have been created to control what data a user can query.

4 Conclusion

In conclusion, we were able to get the database configured and running successfully. Some of the SQL queries submitted in Phase II were not really relevant for our specific database. If we had a credit card hashing algorithm and ran the server from a dedicated host machine and configured security, our software could be a complete solution.