EZPARK DATABASE PROJECT

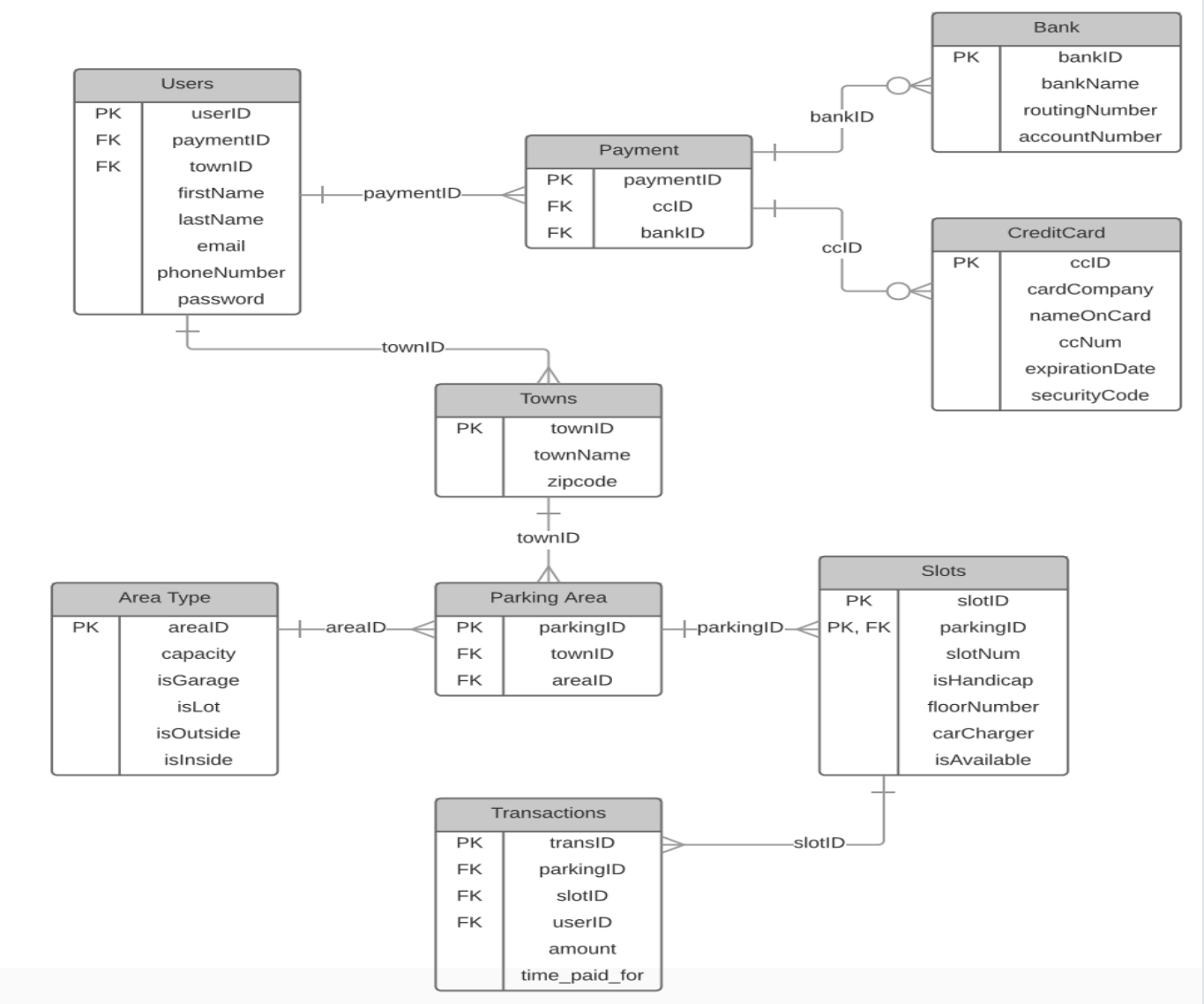
Developed by Christian Isolda

Table of Contents

Executive Summary

Through the whole project I tackled many challenges from technical to mental processes in a business perspective. EzPark is an app that is being developed that takes an annoying task and makes it into a breeze. The app's fucntionality include paying for parking through our app using a credit card or bank, notification reminders on time remaining, and option to add time to existing parking time. The document contains all information surrounding the database and will also give some insight on what the app revolves around. With the plan to expand nationwide and even worldwide a properly structured and organized database is needed to gold the information. The design of the database makes the app flow and function swiftly. The data input is dummy data, but is meant to resemble what kind of information and processes the database will be handling. This database is a fully functional database made for EzPark based around third normal form.

E/R Diagram


Bank
PK bankID
bankName
routingNumber
accountNumber
Users
PK userID
FK paymentID
FK townID
firstName
lastName
email
phoneNumber
password
paymentID
Payment
PK paymentID
FK ccID
FK bankID
bankID
ccID
CreditCard
PK ccID
cardCompany
nameOnCard
ccNum
expirationDate
securityCode
townID
Towns
PK townID
townName
zipcode
townID
Area Type
PK areaID
capacity
isGarage
isLot
isOutside
isInside
areaID
Parking Area
PK parkingID
FK townID
FK areaID
parkingID
Slots
PK slotID
PK, FK parkingID
slotNum
isHandicap
floorNumber
carCharger
isAvailable
Transactions
PK transID
FK parkingID
FK slotID
FK userID
amount
time_paid_for
slotID

Towns Table

This table represents all the available towns that a user can access with our app.

```
CREATE TABLE Towns (
townID char(6) not null,
townName text not null,
zipcode int not null,
primary key(townID));
```

Functional Dependencies:
townID→ townName, zipCode

Sample Data

townid character	townname text	zipcode integer
TN0034	Scotch Pl...	7076
TN0045	Westfield	7090
TN0002	Cranford	7080
TN0047	Summit	7680
TN0123	Fanwood	7079
TN0684	Livingston	7987
TN0007	Florham P...	7060
TN0347	Short Hills	7690
TN1234	Millburn	7567
TN0235	Wayne	7897
TN0281	Randolph	7967
TN1243	Vernon	7643
TN0225	Ridgewood	7264
TN0124	Ramsey	7827
TN0212	Trenton	7176

Bank Table

This table represents all the banks entered from the users so it can be referenced in the payment table

```
CREATE TABLE Bank (
bankID char(6) not null,
bankName text not null,
routingNumber int not null,
accountNumber int not null,
primary key(bankID));
```

Functional Dependencies:
bankID→ bankName, routingNumber, accountNumber

Sample Data

bankid character	bankname text	routingn... integer	accountn... integer
bk001	TDBANK	189879876	189876543
bk002	CHASE	189271276	1898832543
bk003	Bank of A...	199829576	129871543
bk004	Wells Fargo	182859676	129176543
bk005	TDBANK	189834576	189123543
bk006	Bank of A...	189958276	189341253
bk007	Sovereign	199119576	123371543
bk008	Hudson C...	182259676	129176543
bk009	CITI	219879876	182276543
bk010	TDBANK	189651276	1898982543
bk011	CHASE	199479576	129879143
bk012	Wells Fargo	182129676	179176543

Credit Card Table

This table represents all the credit cards entered from the users so it can be referenced in the payment table

```
CREATE TABLE CreditCard (
ccID char(6) not null,
cardCompany text not null,
nameOnCard text not null,
ccNum varchar(16) not null,
expirationDate varchar(5) not null,
securityCode int not null,
primary key(ccID));
```

Functional Dependencies:
ccID→ cardCompany, nameOnCard, ccNum, expirationDate, secuirtyCode

Sample Data

ccid character	cardcom... text	nameonc... text	ccnum character...	expiratio... character...	securityc... integer
cc001	visa	Chris Sheil	1432345...	04/19	467
cc002	master ca...	Davey Le...	2132545...	05/18	962
cc003	discover	Felix Nod...	8272345...	08/18	726
cc004	visa	Jeff Lieblich	1472341...	02/20	277
cc005	american ...	James Bo...	0072345...	07/17	7
cc006	master ca...	Vallery C...	8732195...	04/21	891
cc007	discover	Alan Labo...	2192576...	09/18	182
cc008	visa	Christian ...	7217294...	04/19	407
cc009	american ...	Christian ...	8972312...	01/20	987
cc010	visa	Michael S...	9872341...	10/18	254
cc011	master ca...	Tony Stark	9874642...	11/18	792
cc012	discover	Derek Jeter	9872342...	4/19	299
cc013	visa	Addison K...	8872341...	3/18	887

Area Type Table

This table represents all the types of parking garages that the database holds.

```
CREATE TABLE AreaType (
areaID char(6) not null,
capacity int not null,
isGarage boolean not null,
isLot boolean not null,
isOutside boolean not null,
isInside boolean not null,
primary key(areaID));
```

Functional Dependencies:
ccID→ capacity, isGarage, isLot, isOutside, isInside

Sample Data

areaid character	capacity integer	isgarage boolean	islot boolean	isoutside boolean	isinside boolean
A001	600	true	false	false	true
A002	900	false	true	true	false
A003	100	true	true	true	true
A004	300	true	false	true	true
A005	200	false	true	true	false
A006	700	true	false	false	true
A007	150	true	true	true	false
A008	350	false	true	true	false
A009	900	true	false	false	true
A010	200	false	true	true	false
A011	900	true	false	false	true
A012	1200	true	true	true	true
A013	700	true	false	false	true
A014	150	false	true	true	false
A015	350	true	true	true	true

Payment Table

This table represents all the payment options the user has input to make it easy to pay for the user as well having multiple options.

CREATE TABLE Payment (
paymentID char(6) not null,
ccID char(6) references CreditCard(ccID),
bankID char(6) references Bank(bankID),
primary key(areaID));

Functional Dependencies:
paymentID→ ccID, bankID

Sample Data

paymentid character	ccid character	bankid character
P001	cc001	bk001
P002	cc002	bk002
P003		bk003
P004	cc003	
P005	cc004	bk005
P006		bk006
P007	cc005	
P008	cc006	
P009	cc007	bk007
P010	cc008	bk008
P011	cc009	
P012		bk009
P013	cc010	bk010
P014	cc011	bk011
P015	cc012	bk012

Users Table

This table represents all the users in the database with all their information to be able to utilize the app.

```
CREATE TABLE Users (
userID char(6) not null,
paymentID char(6) not null references Payment(paymentID),
townID char(6) not null references Towns(townID),
firstName text not null,
lastName text not null,
email text not null,
phoneNumber varchar(10) not null,
password varchar(25) not null,
primary key(userID));
```

Functional Dependencies:
userID→ paymentID, townID, firstName, lastName, email, phoneNumber, password

Sample Data

userid character	paymentid character	townid character	firstname text	lastname text	email text	phonenu... character...	password character...
U001	P001	TN0034	Chris	Sheil	c.sheil@g...	9088121...	hello123
U002	P002	TN0045	Davey	Leong	d.leong@...	9088762...	medaveyl...
U003	P003	TN0002	Ryan	Pilliego	r.pilliego...	9085746...	rbutchman
U004	P004	TN0047	Felix	Nodarse	f.nodarse...	9088122...	memer123
U005	P005	TN0123	Jeff	Lieblich	j.lieblich...	9081901...	hahmefed...
U006	P006	TN0684	Darren	Jones	d.jones@...	9088422...	idkdarren
U007	P007	TN0007	James	Bond	007@bon...	9081829...	shakeitno...
U008	P008	TN0347	Vallery	Chosen	v.chosen...	9088872...	frenchie1...
U009	P009	TN1234	Alan	Labouseur	bestprofe...	9088131...	alpaca
U010	P010	TN0235	Christian	Mastroianni	c.mastroi...	9088762...	maiden123
U011	P012	TN0281	Aria	Wester	a.wester...	9088122...	perf678
U012	P013	TN1243	Michael	Santana	m.santan...	9088561...	imaqtpie
U013	P014	TN0225	Tony	Stark	t.stark@y...	9081237...	ironmanst...
Microsoft Word	5	TN0124	Derek	Jeter	d.jeter@y...	9088761...	goat123
U015	P016	TN0212	Addison	Karambit	a.karambi...	9088572...	donplornh

Parking Area Table

This table represents all the areas where the users can actually park.

```
CREATE TABLE ParkingArea (
parkingID char(6) not null,
townID char(6) not null references Towns(townID),
areaID char(6) not null references AreaType(areaID),
primary key(parkingID));
```

Functional Dependencies:
parkingID→ townID, areaID

Sample Data

parkingid character	townid character	areaid character
PA001	TN0034	A001
PA002	TN0045	A002
PA003	TN0002	A003
PA004	TN0047	A004
PA005	TN0123	A005
PA006	TN0684	A006
PA007	TN0007	A007
PA008	TN0347	A008
PA009	TN1234	A009
PA010	TN0235	A010
PA011	TN0281	A011
PA012	TN1243	A012
PA013	TN0225	A013
PA014	TN0124	A014
PA015	TN0212	A015

Slots Table

This table represents all the slots or spaces in each parking area that is input.

```
CREATE TABLE Slots (
slotID char(6) not null,
parkingID char(6) not null references ParkingArea(parkingID),
slotNum int not null,
isHandicap booelan not null,
floorNumber int not null,
carCharger booelan not null,
isAvailable boolean DEFAULT TRUE,
primary key(slotID));
```

Functional Dependencies:
slotID→ parkingID, slotNum, isHandicap, floorNumber, carCharger, isAvailable

Sample Data

slotid character	parkingid character	slotnum integer	ishandicap boolean	floornum... integer	carcharger boolean	isavailable boolean
S021	PA001	1	false	1	false	true
S022	PA001	2	false	1	false	true
S023	PA001	3	false	1	false	true
S024	PA001	4	false	1	false	true
S025	PA001	5	false	1	false	true
S026	PA001	6	false	1	false	true
S027	PA001	7	false	1	false	true
S028	PA001	8	false	1	false	true
S029	PA001	9	false	1	false	true
S020	PA001	10	false	1	false	true
S031	PA001	11	false	1	false	true
S032	PA001	12	false	1	false	true
S033	PA001	13	false	1	false	true
S231	PA002	1	false	1	false	true
S232	PA002	2	false	1	false	true

Transactions Table

This table represents all the transactions that are made between users and slots being taken.

```
CREATE TABLE Transactions (
transID char(6) not null,
parkingID char(6) not null references ParkingArea(parkingID),
slotID char(6) not null references Slots(slotID),
userID char(6) not null references Users(userID),
amount int not null,
time_paid_for varchar(8) not null,
primary key(transID));
```

Functional Dependencies:
transID→ parkingID, slotID, userID, amount, time_paid_for

Sample Data

transid character	parkingid character	slotid character	userid character	amount integer	time_pai... character...
T001	PA004	S060	U001	2	2 hours
T002	PA015	S139	U002	4	4 hours
T003	PA014	S134	U003	3	3 hours
T004	PA013	S138	U004	1	1 hour
T005	PA008	S104	U006	2	2 hours
T006	PA005	S070	U007	3	3 hours
T007	PA005	S074	U009	4	4 hours

Stored Procedure update_slot_status()

This stored procedure updates slots in a parking when a transaction is made. This allows the system to keep track of timings and notifications a lot easier.

```
CREATE OR REPLACE FUNCTION update_slot_status ()
RETURNS TRIGGER AS
$$
BEGIN
IF NEW.transID is NOT NULL THEN
UPDATE Slots
SET isAvailable = FALSE
WHERE NEW.slotID = Slots.slotID
END IF;
RETURN NEW;
END;
$$
LANGUAGE PLPGSQL;
```

Stored Procedure get_users_transaction()

This stored procedure can be used to check the previous transactions.

```
CREATE OR REPLACE FUNCTION get_users_transaction (char(6), REFCURSOR) returns
refcursor as
$$
DECLARE
userID    char(8)              :=$1;
results   REFCURSOR            :=$2;
BEGIN
OPEN results  FOR
SELECT tr.transID, tr.parkingID, tr.userID, tr.slotID
FROM Transactions tr INNER JOIN Users u ON tr.userID = u.userID
WHERE userID = u.userID;
RETURN results;
END;
$$
LANGUAGE PLPGSQL;
```

Trigger update_slot_status_trigger()

This trigger links to the stored procedure in the previous slide, which triggers the stored procedure whenever an INSERT statement is input into Transactions

```
CREATE TRIGGER update_slot_status_trigger
BEFORE INSERT ON Transactions
FOR EACH ROW
EXECUTE PROCEDURE update_slot_status();
```

View availableSlots

This view shows a table of available slots that are not being used by users. This can be used for checking the amount of slots open around or even in a parking area

```
DROP VIEW IF EXISTS availableSLots;
CREATE VIEW availableSlots as (
SELECT pa.parkingID,
        slotID
FROM parkingArea pa INNER JOIN Slots s
        ON pa.parkingID = s.parkingID
WHERE s.isAvailable = TRUE
);
```

```
SELECT *
FROM availableSlots
WHERE parkingID = 'PA001'
```

Sample Data

parkingid character	slotid character
PA001	S021
PA001	S022
PA001	S023
PA001	S024
PA001	S025
PA001	S026
PA001	S027
PA001	S028
PA001	S029
PA001	S020
PA001	S031
PA001	S032
PA001	S033

View townParkingAreas

This view shows a table of parking areas that a town has to offer. This can help the company expand into different areas as well as help towns understand their needs.

```
DROP VIEW IF EXISTS townParkingAreas;
CREATE VIEW townParkingAreas as (
SELECT t.townName,
        pa.parkingID
FROM parkingArea pa INNER JOIN Towns t
        ON pa.townID = t.townID
);
```

Sample Data

townname text	parkingid character
Scotch Plains	PA001
Scotch Plains	PA4313

```
SELECT *
FROM townParkingAreas
WHERE townName = 'Scotch Plains'
```

Report / Interesting Query 1

This report searches for the parking ID provided and sums the total amount generated from a specified parking area. This is useful for analytics as well as which types of areas are more used.

```
SELECT SUM(amount) AS totAmountUSD
FROM transactions
WHERE parkingID = 'PA004'
```

Sample Data

totamountusd bigint
2

Report / Interesting Query 2

This report queries all the users and shows their total amount spent on parking. This is very useful for analytics as well as pricing and marketing to grow the app.

```
SELECT u.userID,
        SUM(amount) AS userTotalUSD
FROM Users u INNER JOIN Transactions t
        ON u.userID = t.userID
GROUP BY u.userID
ORDER BY userTotalUSD DESC;
```

Sample Data

userid character	usertotalusd bigint
U002	4
U009	4
U003	3
U007	3
U006	2
U001	2
U004	1

Report / Interesting Query 3

This report counts the amount of bank accounts and credit cards in use. This is very useful for determining which payment method is being used more frequently as well as what more types of options we should offer.

```
SELECT COUNT(ccID) AS totalCC,
            COUNT(bankID) as totalBank
FROM Payment;
```

Sample Data

totalcc bigint	totalbank bigint
13	11

Security

There are two roles in EzPark: Admin and Users
Admin: Have control over DB and can monitor and update as needed.
Users: Users have privileges to insert and update their user information as well as their payment information.

ADMIN:
GRANT ALL ON ALL TABLES IN SCHEMA PUBLIC TO ADMIN;

USERS:
REVOKE ALL ON ALL TABLES IN SCHEMA PUBLIC FROM USERS;
GRANT INSERT ON Users, Payment, CreditCard, Bank TO USERS;
GRANT UPDATE ON Users, Payment, CreditCard, Bank TO USERS;

Implementation – Known Problems – Future Enhancements

- Implementation:
  - If the app was to grow to full scale there would need to be more character values to implement towns into the app.
  - The app would need to have functionality to communicate with parking station hardware which means software and hardware changes would need to be done to make it communicate.
- Known Problems
  - The second stored procedure is very close to working and can not get it to output correct data.
  - There will need to be many more options for IDs as well as general information categorizing data in the database, the inputs now are just dummy data and should be fixed before implementation.
- Future Enhancements
  - More payment options such as PayPal would be added to enhance the experience as well provide more options for uses to use the app.
  - Future development for the app would have a feature that shows a map of the parking area and indicates where there are open slots so it makes parking that much faster.