CS 157A Project 2 Report

Team Members: Ahmed Abdelfaheem, Nada El Zeini, Pranika Bedi, Ying Chang Cui

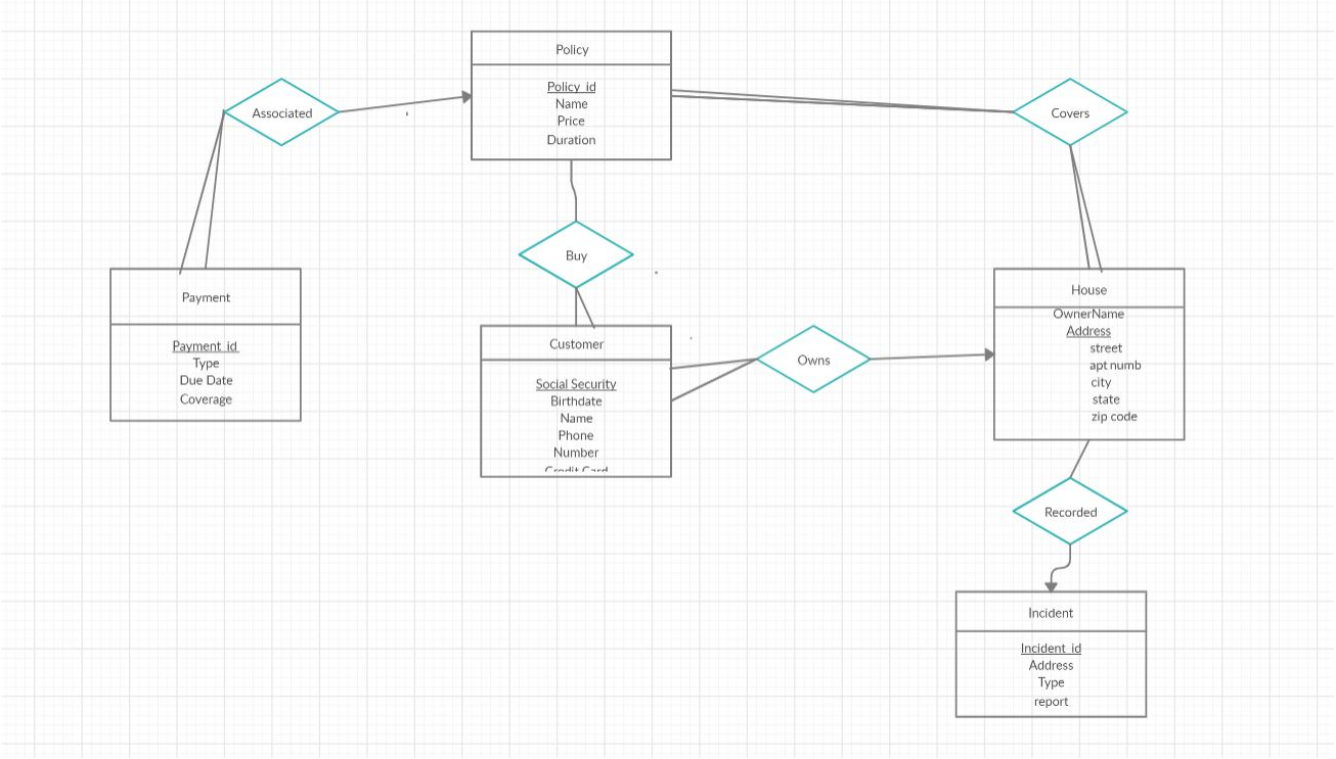
1. Introduction

The goal of this project was to develop a relational database design for a growing company that sells homeowners insurance. Using the information provided, we initially created an ER diagram. This ER diagram captured the conceptual model accurately, taking into consideration possible assumptions and further mapping cardinalities. Our next step was then to map out the diagram into a relational database schema, where we ensured all provided information was taken into account. Additionally, we normalized our relational schema and created interesting, yet important SQL queries that would answer questions regarding the company's homeowners insurance.

This project allowed us to better understand homeowners insurance and what would be required in a company that sells this insurance. We discovered important attributes and their relations to make an accurate design. Through discussion and research, we determined important steps to take regarding creating the design and normalizing our schema.

2. ER Model

When designing our ER model, we had to fully understand not only the information that was provided to us, but also the building blocks of an ER model, which includes entities, relationships, and attributes. Once we understood these concepts, we discussed how to accurately transition the information into figures and mapping cardinalities. Through trial and error, we designed the following ER model:



We tried our best to include all parts of the conceptual model into the ER diagram and believe we were pretty successful in doing so. However, our core design issues originated from accurately representing ER constraints.

As shown in the ER model above, the relationships between each entity have various meanings. The relation, Customer-Owns-House, is represented with a double arrow transitioning into a single, solid arrow emphasizing that a customer can own at least one home. As we continued to map out the diagram, it became more difficult to specifically determine the cardinalities in relation to the entity sets.

It was important for us to understand what we had to implement. For example, an insurance policy can cover one or more homes, but it also defines the payments associated with not only each home, but also each policy. Therefore, we realized we had to create a relation with the entity sets, Payment, Policy, and House. This is where the bulk of the design issues came from. At this point, we had identified the entity and relationship sets needed. The next steps were to identify the mapping cardinalities and participation constraints. We were able to figure this out by discussing among each other and reviewing the lecture slides. We thus decided to make a weak relationship set between Payment and Policy. Payment-Associated-Policy is designed with a double to single arrow to show that a payment can have at least one policy associated with it. However, a policy can cover multiple homes, and a home can have many policies. Policy-Covers-House then holds a many-to-many relationship, identified by the double lines towards each entity set.

It was fairly easy determining the attributes of each set, but it was difficult determining how the primary keys of each entity set would map out, due to a weak relationship set existing. However, after discussing and examining lecture notes, we concluded and found the primary keys of the relationship and entity sets.

3. Relational Schema derived from the ER Model

Entity Sets:

[Customer, House, Policy, Incident, Payment]

Relations:

Customer	Owns	House	one to many	total participation from the customer entity set
Customer	Buy	Policy	many to many	total participation from the customer entity set
policy	Associated	Payment	many to one	weak relation
policy	Covers	House	many to many	total participation from the policy and payment entity set
House	Recorded	Incident	one to many	

Converting ER Model to Relation Schema:

Converting the ER model to relational schema is an important step in the logical design process. It was important to follow the rules concerning strong entity sets, weak entity sets, and relationship sets. As a result, we had to correctly determine the mapping cardinalities in order to determine the right case for the relational schema. As seen above in the table and below, the relational schema varies depending on the mapping cardinalities. A many-to-many relationship uses the primary keys of the entity set and the attributes of the relationship set. A many-to-one relationship affects what attribute will be the primary key. These cases are thus similar for the rest of mapping cardinalities and are demonstrated below.

Customer (social_security, name, creditnum, phonenum, birthdate)
House (ownername, street,apt_num,city, state, zip_code)
Owns(social_security,street,apt_num,city, state, zip_code)
+foreign key (social_security) references customer
+foreign key (street,apt_num,city, state, zip_code) references house

Policy(policy_id, name, price, duration)
Payment(payment_id, type, due_date, coverage) red is partial key
Associated(payment_id, policy_id,type, due_date, coverage)

House (ownername, street,apt_num,city, state, zip_code)
Policy(policy_id, name, price, duration)
Covers(policy_id,street,apt_num,city, state, zip_code)
+foreign key (policy_id) references policy
+foreign key (street,apt_num,city, state, zip_code) references house
Customer (social_security, name, creditnum, phonenum, birthdate)
Policy(policy_id, name, price, duration)
Buy(social_security, policy_id)
+foreign key (social_security) references customer
+foreign key (policy_id) references policy

incident(incident_id, address, type,report)
House (ownername, street,apt_num,city, state, zip_code)
recorded(incident_id, street,apt_num,city, state, zip_code)
+foreign key (incident) references incident
+foreign key (street, apt_num,city, state, zip_code) references house

The Functional Dependencies are already stated in the normalization

4. Normalized relational schema

To finalize our relational schema, an important step is to normalize to make sure the initial relational design is refined to a maximum. To improve a schema we need to enumerate and

analyze the functional dependencies trying to identify some redundancy. Redundancy can exist if two things are present:

1. a functional dependency
2. the possibility of multiple rows involving the source of the multiple dependencies

In this section, we check whether or not our suggested tables are in BNCF. A table will be normalized if it is not in BCNF.

The table below shows the functional dependencies for each

Schema	Functional Dependencies	Need to Normalize	BCNF
policy(policy_id, name,price, duration)	policy_id → name, price, duration	No need, policy_id is a superkey and is on the left side. There are no other functional dependencies where the left side is not a superkey.	No other FDs, the schema is already in BCNF

payment(*payment_id*, type, due_date, coverage)	payment_id → type, due_date, coverage	No need, payment_id is a partial key since payment is a weak entity set	The schema is in BCNF
associated(payment_id, policy_id,type, due_date, coverage)	1) payment_id,policy_id → type, due_date, coverage 2) payment_id →policy_id	Split: 1. {payment_id,policy_id} 2. {payment_id,type, due_date, coverage} Since the second schema is actually the payment schema above we just reduce the associated schema to associated{payment_id, policy_id} instead	associated{payment_id,policy_id} now in BCNF with payment_id as primary key
customer(social_security, name, birthdate, phone_num, credit_card_num)	1) social_security → name, birthdate, phone_num,credit_card_num	No need. We assume that the credit_card_num and the phone_num can be changed and altered, therefore no other functional dependencies are here	Already in BCNF, the left hand side of the FD is a superkey

house(owner_name, street,apt_num,city, state, zip_code)	1) street,apt_num,city, state, zip_code → owner_name 2) zip_code →state	zip_code can also indicate the state, and zip_code is not a superkey so we have to further normalize	Split: {zip_code,state} {street,apt_num,city, state,zip_code,owner_name}
incident(incident_id, street,apt_num,city, state, zip_code, type, report) + foreign key street,apt_num,city, state, zip_code references house	incident_id → address, type, report	incident_id is a primary key and so a superkey so no need to normalize	Already in BCNF
Buy(social_security, policy_id) +foreign key (social_security) references customer +foreign key (policy_id) references policy	social_security → policy_id	No need to normalize, we assume every customer would buy one policy	Already in BCNF

Owns(social_security,street,apt_num,city, state, zip_code) +foreign key (social_security) references customer +foreign key (street,apt_num,city, state, zip_code) references house	social_security → street, apt_num,city, state, zip_code	No need to normalize, social_security is unique and each customer can own a house	Already in BCNF
Covers(policy_id,street,apt_num,city, state, zip_code) +foreign key (policy_id) references policy +foreign key (street,apt_num,city, state, zip_code) references house	policy_id → street,apt_num,city, state, zip_code	In this schema all the attributes form the primary key, no need to normalize. This relationship involves the policy and the house it covers	Already in BCNF
recorded(incident_id, street,apt_num,city, state, zip_code)	incident_id → street, apt_num, city, state, zip_cde	The FD doesn't have any redundancy. The left hand side is a superkey so no need to normalize	Already in BCNF

Here is the final relational schema in SQL documentation:

```sql
create table policy
(
   policy_id VARCHAR(8) not null,
   name VARCHAR(12) not null,
   price NUMERIC(6,2) not null check (price > 0),
   duration INTEGER not null,
   PRIMARY KEY(policy_id)
);

create table payment
(
   payment_id VARCHAR(8) not null,
   type VARCHAR(12) not null,
   due_date DATE not null,
   coverage VARCHAR(50) not null,
   PRIMARY KEY(payment_id)
);

create table customer
(
   social_security NUMERIC(9,0) not null unique,
   name VARCHAR(12) not null,
   credit_card_num NUMERIC(16,0) not null,
   phone_num NUMERIC(10,0) not null,
   birthdate DATE not null,
   PRIMARY KEY(social_security)
);

create table house
(
   owner_name VARCHAR(12) not null,
   street VARCHAR(24) not null,
   apt_num NUMERIC(4,0),
   city VARCHAR(12) not null,
   state CHAR(2) not null,
   zip_code NUMERIC(5,0) not null,
   PRIMARY KEY(street,apt_num,city,state,zip_code)
);

create table associated
(
   payment_id VARCHAR(8) not null,
   policy_id VARCHAR(8) not null,
   primary key(payment_id),
   foreign key(payment_id) references payment,
   foreign key(policy_id) references policy
);

create table incident
```

```
(
   incident_id VARCHAR(8) not null,
   street VARCHAR(24) not null,
   apt_num NUMERIC(4,0),
   city VARCHAR(12) not null,
   state CHAR(2) not null,
   zip_code NUMERIC(5,0) not null,
   type VARCHAR(12) not null,
   report VARCHAR(32) not null,
   primary key(incident_id),
   foreign key(street,apt_num,city,state,zip_code) references house
);
create table buy
(
   social_security NUMERIC(9,0) not null unique,
   policy_id VARCHAR(8) not null,
   primary key(social_security),
   foreign key(social_security) references owner,
   foreign key(policy_id) references policy
);

create table owns
(
   social_security NUMERIC(9,0) not null unique,
   street VARCHAR(24) not null,
   apt_num NUMERIC(4,0),
   city VARCHAR(12) not null,
   state VARCHAR(12) not null,
   zip_code NUMERIC(5,0) not null,
   primary key(social_security),
   foreign key(social_security) references customer,
   foreign key(street,apt_num,city,state,zip_code) references house

);

create table recorded
(
  incident_id VARCHAR(8) not null,
  street VARCHAR(24) not null,
  apt_num NUMERIC(4,0),
  city VARCHAR(12) not null,
  state VARCHAR(12) not null,
  zip_code NUMERIC(5,0) not null,
  primary key (incident_id)

);

create table covers
(
```

```
    policy_id VARCHAR(8) not null,
    street VARCHAR(24) not null,
    apt_num NUMERIC(4,0),
    city VARCHAR(12) not null,
    state VARCHAR(12) not null,
    zip_code NUMERIC(5,0) not null,
    primary key(policy_id, street, apt_num, city, state, zip_code),
    foreign key(policy_id) references policy,
    foreign key(street,apt_num,city,state,zip_code) references house
);
```

For each attribute, we carefully decided on the type based on realistic expectations. For instance,the social security number is a 9-digit number so we chose NUMERIC(9,0) to indicate that. In addition, the social security number is unique which explains adding UNIQUE as a type. Also, including most of the other attributes in the complete schema, NOT NULL is important to make sure there are no inconsitie in the data. An attribute that didn't need NOT NULL was the apartment number for the address where in realistic measure it's not always required to define an address in the United States. The zip code needed is of 5 digits which explains the type NUMERIC(5,0). In all of the tables created above, the primary key is a unique id identifying the set except for the house table where it's the full address of the house. Other attributes are VARCHAR and the size specified is based on what the company would expect from customers.

5. Sample data and SQL queries

Think of at least 5 questions about the data that would be interesting to a company that sells Homeowners Insurance. Write each of them in SQL. Do not use only questions that are simple to express in SQL.

We started off thinking about the type of data a company that sells homeowners insurance would want such as their customer age demographic and what policies are attracting the most customers. There were a few questions that were obvious for us to think of such as "What is the policy id with the most customers". We figured that most companies would want this data because it's important to know what is working for the company. All of our questions were based on "what data will help the insurance companies save more money" because that would be the only reason to develop a database design to sell homeowners insurance. Question number 4, "Which type of payment is most commonly used" will make the company be more aware of that transaction between that type of payment than any other because of how important it is compared to the others.

The second concept our team wanted our questions to focus on was to use "group by" because it provides a lot of data for each category of interest. One question that uses "group

by" is "For each city, what is the most bought policy?". We believe this is an important question because companies can base their funding/policies on each specific area which will save them money. It can also be used as an advertisement strategy for the insurance company to reassure interested buyers.

question 1: what is the policy id with the most customers?
SQL answer: select policy_id from (select policy_id, count(payment_id) from associated group by policy_id limit 1);
SQL result: 10000000

question 2: For each city what is the most bought policy?
SQL answer: select city, count(policy_id) from covers group by city;
SQL result:

city	count(policy_id)
Las Vegas	1
Los Angeles	1
New York C	1
Phoenix	2
San Franci	1

question 3: What type of incident is most frequent in Arizona?
SQL answer: select type from (select type, count(type) from incident where state = 'AZ' group by type);
SQL result:
Fire and smoke damage
Roof collapse

question 4: which type of payment is most commonly used?
SQL answer: select type from payment group by type order by -count(type) limit 1;
Result: credit

question 5: what is the average age of customers?
SQL answer: select round(avg(2020-substr(birthdate,0,5))) from customer;
SQL result: 64

6. Conclusions

We started with the ER model connecting each relationship with the gathered information in the problem statement. All the entity and relationship sets were made with details such as mapping cardinalities. After approval, we map the ER diagram to a relational

database schema using the procedure we have learned in class. Then we discussed if we needed to normalize for each schema by listing out all the functional dependencies. Our team had a lot of questions related to normalization that we wouldn't have thought about if we didn't have this project. We realized that each assumption we needed to make would change the entire look of the database. We also agreed that some schemas didn't need to be normalized because of little use of would bring to the database.

After we finished creating the normalized relational schemas, we made the sql statements such as tables and inserts. Our related data needs to be connected so the relationships would make sense. Then we thought of questions the insurance company would most likely be interested in and answered them in sql. After completing this project, we realized how much thought it takes to make a database. From creating an ER model from given information to making sql queries of the finished product, all of the work needed to be interconnected and built upon from start to finish.