Data Design

Team Members: Yuta Kihara, Le Duy Vu

I. Introduction

Database design is the process of creating a data model by abstracting the real world so that the data can be managed by the database and make the use of it to some application. The data model defines how the database is organized. For relational databases, the work of creating data models (data modeling) is generally performed through three stages: conceptual design, logical design, and physical design. Then, at each stage, a conceptual model, a logical model, and a physical model are created as outputs.

In this project, we would like to show each process of database design and deal with the real-world problem.

[PROBLEM STATEMENT]

Our consulting company has been asked to develop a relational database design for a growing company that sells homeowners insurance.

You gather the following information:

- Each customer of the company owns at least one home.
- Each home has associated incidents that are recorded by the insurance company.
- An insurance policy can cover one or multiple homes.
- The policy defines the payments associated with the policy, and a policy that covers multiple homes will show the payments associated with each home.
- Associated with each payment is a payment due date, the time period of coverage, and a date when the payment was made.

II. ER Model

1. Identify entity sets

After doing an analysis on your company's request and data, we suggest a data design including entity sets as follows:

- Homeowner
- Home
- Incident

- Policy
- Payment

First of all, it is crucial to have an entity set that includes your customers' personal information such as name, phone number, email, address, etc. We call it **Homeowner**. Also, since each customer owns at least one home, the entity set **Home** should be identified, which includes essential information about the house like address, age, area, etc. Besides, as each home has associated incidents that are recorded by the insurance company, the entity **Incident** is also required. It stores detailed information about the incidents each home has as a recording.

It is worth noting that from the context, we are not certain about what role **Incident** plays in the big picture, so we would like to convey how we interpret the meaning. We assume Incident is simply an event happening to a **Home** and does not relate to any other entity sets.

Additionally, **Policy** is one of the important entities because an insurance policy is connected with both **Home** and **Payment**. We define **Payment** as an entity because it is necessary to hold payment information such as payment date and the amount of money involved.

2. Identify relationship sets:

- own (between Homeowner and Home)
- record (between Home and Incident)
- cover (between Policy and Home)
- define (between Policy and Payment)
- home_payment (between Payment and Home)

After identifying the entity sets in II.1., we would like to introduce the relationship sets between these entities. Each **homeowner** owns at least one **home**, so we create <u>own</u> relationship set.

When an **incident** happens to a **home**, it is recorded in the database of the company. Therefore, a relationship set <u>record</u> is needed to connect **home** and **incident**. Note that <u>record</u> is an identifying relationship as **Incident** is a weak entity set. It has no primary key and needs the address of **home** to differentiate among them.

Between **policy** and **home**, there must be a relationship set such that **home** is <u>cover</u>ed by **policy**. In addition, an insurance **policy** must be clear how much it charges each **home**, so we have the relationship <u>define</u> as a **policy** defines a **payment**. Finally, relationship <u>home_payment</u> connects **payment** and **home** since a **payment** is required to activate an insurance **policy** coverage for a **home**.

3. Identify mapping cardinalities (upper limits) and participation constraints (lower limits)

- Homeowner <= own = Home

  There is no rule regulating that a homeowner can only own 1 single home, so a

homeowner can theoretically own many homes. In contrast, we only allow a home belongs to 1 single person, and that person will be the homeowner. Therefore, the own relationship is many to one.
Based on the information we gathered, each customer of your company has to own at least 1 home, the participation of Homeowner in this relationship will be total. The same thing is true for Home itself as it has to be under the name of one and only one person.

- Home ← record = Incident
  A home in your database can have a various number of incidents attached to it, including 0. So the participation of Home in Record is partial. However, an incident can only have records about at least 1 home, so it's participation is total.
  A home is not limited to any maximum number of incidents, so it can have as many incidents as possible. However, an incident can only be connected to at most 1 home. Therefore, record relationships are many to one.

- Home = cover → Insurance Policy
  From your requirement, we have come up with a design so that a home can only be covered by one and only one single insurance policy. Meanwhile, a policy can cover any amount of homes, including 0. Therefore, the cover relationship is many to one. The participation of Home is total and participation of Policy is partial.

- Payment = define → Policy
  Policy defines some properties of Payment. A policy can define any number of existing payment. It can already define 0 (because no home is registered under that policy yet), or more payments. On the other side, a payment can only fall under 1 and only 1 policy. In our design, no payment is allowed to cover more than one policy. Therefore, the defined relationship is many to one. Policy's participation constraint is partial while Payment's is total.

- Payment = home_payment => Home
  The purpose of every payment is to cover insurance for a specific home for a limited period. Therefore, 1 payment can only pay for one and only one home. Let's say a homeowner owns 3 homes, then they have to make 3 separate payments for each of their homes. In contrast, a home can have multiple payments through its lifetime since each payment does not ensure insurance forever. Therefore, the home_payment relationship is many to one. Payment's participation is total and the same goes to Home.

4. Identify attributes of entity and relationship sets

After careful consideration, we have decided to create these attributes below:

- **Homeowner**

  SSN // VARCHAR: owner social security number = ID
  name // VARCHAR: owner name
  first_name
  middle_name
  last_name
  gender // VARCHAR: constraint only three options male, female, nonbinary
  date_of_birth // DATE: the format is 'YYYY-MM-DD'
  phone // VARCHAR: owner's phone number
  email // VARCHAR: owner's email address

- **Home**

  street // VARCHAR:
  city // VARCHAR:
  state // VARCHAR:
  zip_code // VARCHAR:
  age_year // INTEGER: how many years it passed since the house was built
  area_meter_square // INTEGER: area of the house, the unit is meter square

- **Payment**

  confirmation_number // VARCHAR: the role for primary key
  Due Date // DATE: (start date of coverage plan)
  Payment_date // DATE: the date owner paid
  Payment_amount // DECIMAL how much owner paid
  Coverage_time_day // INTEGER

- **Incident**

  remark // TEXT: the description of what happened about an incident
  date // DATE: the date when the incident happens
  damage_cost // DECIMAL: how much cost it takes to be repaired

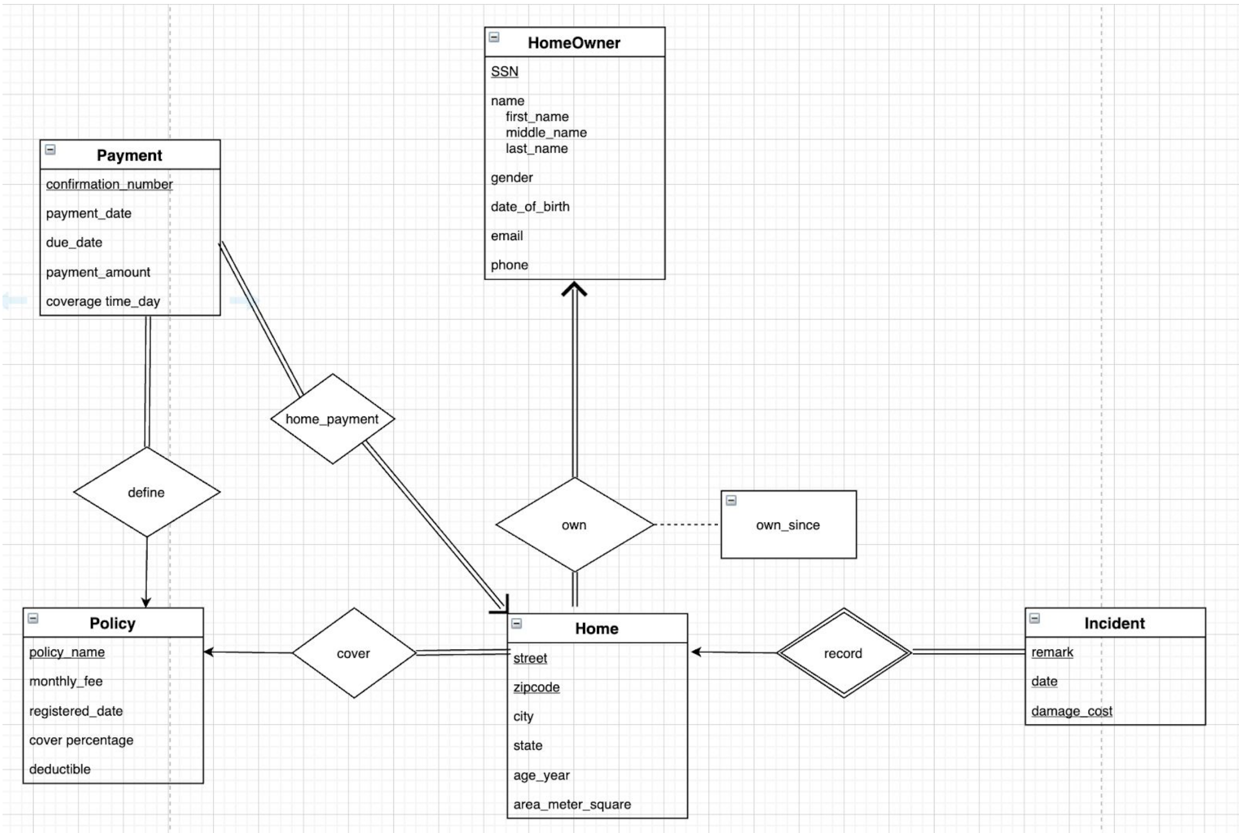
- **Policy**

  policy_name // VARCHAR: unique policy name
  monthly_fee // DECIMAL: how much an insurance policy costs per month
  registered_date // DATE: what date owner register an insurance policy
  coverage_percentage // FLOAT: the percentage that an insurance policy covers
  deductible // DECIMAL: insurance deductible

- **Own (relationship set)**

  own_since // DATE: the date since owner owns

5. ER Diagram



III. Relational Schema Derived From ER Model

```
CREATE TABLE homeowner
(
    SSN VARCHAR(9) NOT NULL PRIMARY KEY CHECK (LENGTH(SSN) = 9 AND SSN
    NOT LIKE '%[^0-9]%'),
    first_name VARCHAR(20) NOT NULL,
    middle_name VARCHAR(20),
    last_name VARCHAR(20) NOT NULL,
    gender VARCHAR(9) CHECK (gender IN ('Male', 'Female', 'Nonbinary')),
    date_of_birth DATE,
    email VARCHAR(320) NOT NULL UNIQUE,
    phone VARCHAR(10) CHECK (LENGTH(phone) = 10 AND phone NOT LIKE '%[^0-9]%')
    NOT NULL UNIQUE
) ;
```

```
CREATE TABLE home
(
    street VARCHAR(50) NOT NULL,
    city VARCHAR(50) NOT NULL,
    state VARCHAR(2) NOT NULL CHECK (LENGTH(state) = 2 AND state LIKE '%[^0-9]%'),
    zipcode VARCHAR(5) NOT NULL CHECK (LENGTH(zipcode) = 5 AND zipcode NOT LIKE
    '%[^0-9]%'),
    age_year INTEGER NOT NULL CHECK (age_year > 0),
    area_meter_square INTEGER NOT NULL CHECK (area_meter_square > 0),
    SSN VARCHAR(9) NOT NULL CHECK (LENGTH(SSN) = 9 AND SSN NOT LIKE
    '%[^0-9]%'),
    policy_name VARCHAR(255) NOT NULL,
    PRIMARY KEY (street, zipcode),
    FOREIGN KEY (SSN) REFERENCES homeowner ON DELETE CASCADE,
    FOREIGN KEY (policy_name) REFERENCES policy ON DELETE SET NULL
) ;

CREATE TABLE own
(
    SSN VARCHAR(9) NOT NULL CHECK (LENGTH(SSN) = 9 AND SSN NOT LIKE
    '%[^0-9]%'),
    street VARCHAR(50) NOT NULL,
    zipcode VARCHAR(5) NOT NULL CHECK (LENGTH(zipcode) = 5 AND zipcode NOT LIKE
    '%[^0-9]%'),
    own_since DATE NOT NULL,
    PRIMARY KEY (SSN, street, zipcode),
    FOREIGN KEY (SSN) REFERENCES homeowner ON DELETE CASCADE,
    FOREIGN KEY (street) REFERENCES home ON DELETE CASCADE,
    FOREIGN KEY (zipcode) REFERENCES home ON DELETE CASCADE
) ;

CREATE TABLE policy
(
    policy_name VARCHAR(255) NOT NULL PRIMARY KEY,
    monthly_fee DECIMAL(10, 2) NOT NULL CHECK (monthly_fee > 0),
    registered_date DATE NOT NULL DEFAULT 'now',
    cover_percentage FLOAT NOT NULL CHECK (cover_percentage > 0 AND cover_percentage
    <= 1),
    deductible DECIMAL(10, 2) NOT NULL CHECK (deductible > 0)
);

CREATE TABLE incident
```

```
(
    remark TEXT NOT NULL DEFAULT '',
    date DATE NOT NULL,
    damage_cost DECIMAL(10, 2) NOT NULL CHECK (damage_cost > 0),
    street VARCHAR(50) NOT NULL,
    zipcode VARCHAR(5) NOT NULL CHECK (LENGTH(zipcode) = 5 AND zipcode NOT LIKE
    '%[^0-9]%'),
    PRIMARY KEY (street, zipcode, date),
    FOREIGN KEY (street) REFERENCES home ON DELETE CASCADE,
    FOREIGN KEY (zipcode) REFERENCES home ON DELETE CASCADE
);

CREATE TABLE payment
(
    confirmation_number VARCHAR(15) NOT NULL PRIMARY KEY,
    payment_date DATE,
    due_date DATE NOT NULL,
    payment_amount DECIMAL(10, 2) NOT NULL CHECK (payment_amount > 0),
    coverage_time_day INTEGER NOT NULL CHECK (coverage_time_day > 0),
    street VARCHAR(50) NOT NULL,
    zipcode VARCHAR(5) NOT NULL CHECK (LENGTH(zipcode) = 5 AND zipcode NOT LIKE
    '%[^0-9]%'),
    policy_name VARCHAR(255) NOT NULL,
    FOREIGN KEY (street) REFERENCES home ON DELETE CASCADE,
    FOREIGN KEY (zipcode) REFERENCES home ON DELETE CASCADE,
    FOREIGN KEY (policy_name) REFERENCES policy ON DELETE SET NULL
);
```

We represent our relational schemas in SQL as the code shown above. We add various checking constraints to ensure the data is in valid range and format. We also include foreign key attributes in entities' schema and relationship's schema to show the relationship between entity sets.

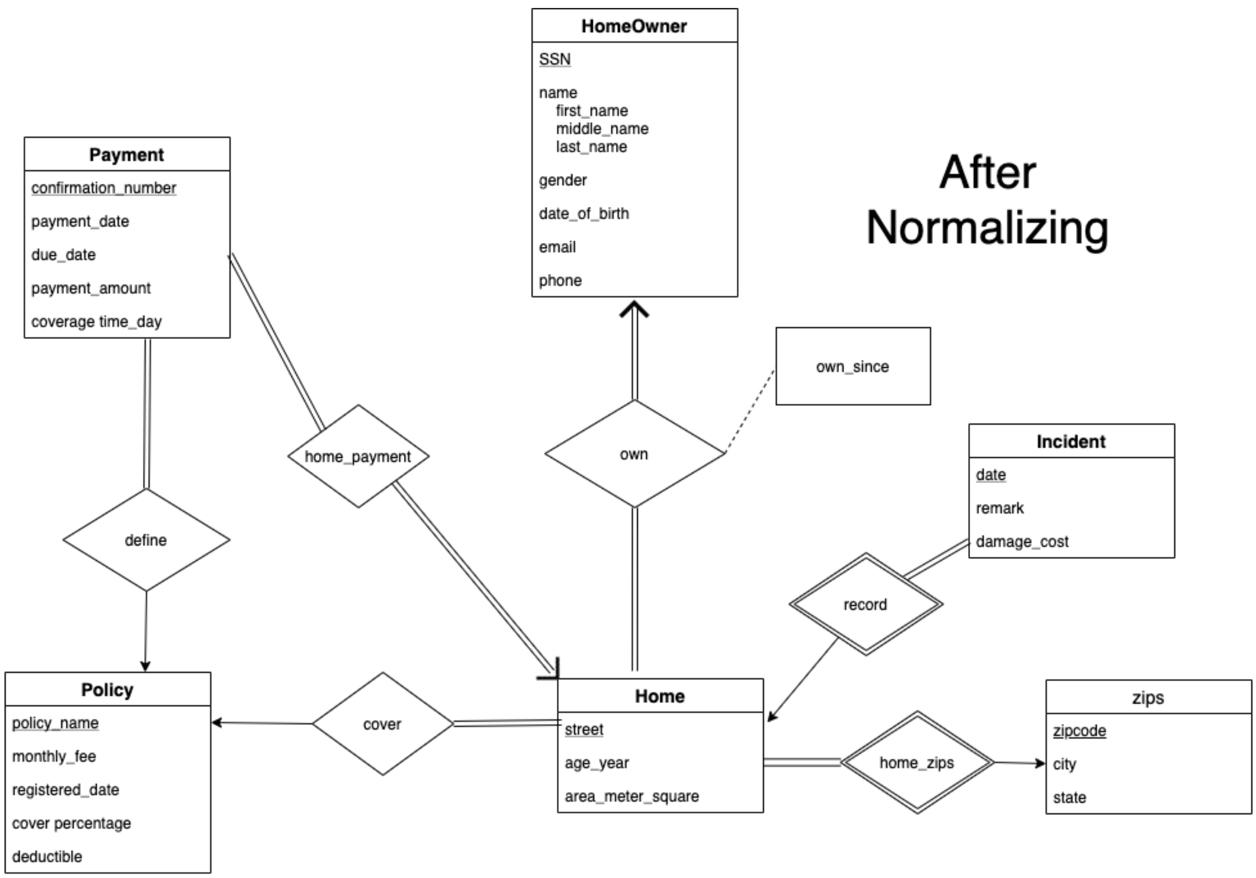
Functional Dependencies: Below are the list of our discovered non-trivial functional dependencies (also excluding the ones with primary key as left hand side):

- In **homeowner** schema: <u>email</u> -> everything else, <u>phone</u> -> everything else
  These functional dependencies will put **homeowner** in 3NF because both <u>email</u> and <u>phone</u> are candidate keys. We set both of them as unique so there will be no duplication in these columns.
- In **home** schema: zipcode -> state, zipcode -> city
  These functional dependencies will make **home** fall out of 3NF to 2NF. This will be the motivation for the normalization process in the next part.

IV. Normalized Relational Schema

From the functional dependencies mentioned above in **Home** schema, we have decided to normalize **Home** schema to move **Home** become 3NF, effectively making all schemas in BCNF/3NF. To tackle this, we create a new schema **Zips** and move zipcode, city, and state to the new schema. Now **Zips** will have 3 attributes and zipcode is its primary key. Back in **Home**, we only have street, age_year, and area_meter_square left as its indeginous attributes. It is worth noting that this modification makes **Home** a weak entity set since one of its 2 primary keys, zipcode, now belongs to another schema. Therefore, the new relationship **home_zips** between **Home** and **Zips** is an identifying relationship. **home_zips** is many to one, with **Home**'s participation being total and **Zips**'s participation can be partial.

The finaled, normalized diagram is shown below. The final version of relation schemas is saved in the attached SQL file.



V. Sample Data and SQL Queries

To demonstrate how our design and product can be applied in the real life world, we have come up with several 1 million dollar questions and queries that your company may be interested in. The data these queries provide can play a big role in your company's analysis and strategic decision. Hereby, we supply a small set of data samples and show how these above queries and questions are answered with this data set.

INSERT INTO homeowner (SSN, first_name, middle_name, last_name, gender, date_of_birth, email, phone) VALUES
('493829382', 'James', '', 'Born', 'Male', '1975-03-12', 'jamesborn@gmail.com', '3859374394'),
('294857204', 'Abbey', '', 'Edward', 'Nonbinary', '1964-09-21', 'abbeyedward@gmail.com', '9348509340'),
('934857920', 'Agena', '', 'Keiko', 'Female', '1993-05-21', 'agenakeiko@gmail.com', '9435864130'),
('849348571', 'Alba', '', 'Jessica', 'Female', '1979-12-21', 'albajessica@gmail.com', '9483758202') ;

INSERT INTO home VALUES
('90 Hello Ave', '96801', 23, 100, '934857920', 'policy-2'),
('43 Goodbye Road', '72201', 100, 200, '493829382', 'policy-2'),
('11 Volcano Court', '06101', 1, 250, '849348571', 'policy-1'),
('375 Corona Street', '95148', 68, 50, '493829382', 'policy-4'),
('666 Alien Blvd', '99501', 12, 643, '294857204', 'policy-2'),
('1024 Milkyway Ave', '30301', 53, 369, '849348571', 'policy-1'),
('9099 Polaris Court', '20001', 71, 876, '493829382', 'policy-4') ;

INSERT INTO policy (policy_name, monthly_fee, registered_date, cover_percentage, deductible) VALUES
('policy-1', 100.00, '2020-01-01', 0.8, 3000.00),
('policy-2', 120.00, '2020-01-01', 0.85, 4000.00),
('policy-3', 150.00, '2018-12-12', 0.90, 5000.00),
('policy-4', 175.00, '2019-03-22', 0.95, 7500.00) ;

INSERT INTO incident (remark, date, damage_cost, street, zipcode) VALUES
('The earthquake happened and the house has been gone', '2020-02-21', 2000.00, '90 Hello Ave', '96801'),
('The big typhoon came to my city and the roof of my house is gone.', '2020-02-22' , 4050.00, '90 Hello Ave', '96801'),
('Tsunami has eaten everything of my house.', '2020-03-11', 10000.00, '43 Goodbye Road', '72201'),
('Tiger ate my house.', '2020-02-23', 6000.00, '11 Volcano Court', '06101'),
('Thunder destroyed my house', '2019-12-31', 4000.00, '375 Corona Street', '95148'),
('Fire Fire Fire Fire Fire', '2020-04-30', 4500.00, '666 Alien Blvd', '99501') ;

INSERT INTO zips (zipcode, city, state) VALUES
('95148', 'San Jose', 'CA'),
('99501', 'Anchorage', 'AK'),

('85001', 'Phoenix', 'AZ'),
('72201', 'Little Rock', 'AR'),
('80201', 'Denver', 'CO'),
('06101', 'Hartford', 'CT'),
('19901', 'Dover', 'DE'),
('20001', 'Washington', 'DC'),
('30301', 'Atlanta', 'GA'),
('96801', 'Honolulu', 'HI') ;

INSERT INTO own (SSN, street, zipcode, own_since) VALUES
('493829382', '90 Hello Ave', '96801', '2012-02-21'),
('294857204', '43 Goodbye Road', '72201', '2013-07-21'),
('934857920', '11 Volcano Court', '06101', '2014-09-21'),
('849348571', '375 Corona Street', '95148', '2015-01-21'),
('493829382', '666 Alien Blvd', '99501', '2013-02-21'),
('294857204', '1024 Milkyway Ave', '30301', '2014-11-21'),
('493829382', '9099 Polaris Court', '20001', '2016-09-21') ;

- How many payments and how much money did each homeowner spend?
  SELECT first_name, last_name, total_payment, payment_amount FROM homeowner NATURAL JOIN (SELECT SSN, COUNT(confirmation_number) AS total_payment, SUM(payment_amount) AS total_amount FROM homeowner NATURAL JOIN home NATURAL JOIN payment WHERE payment_date IS NOT NULL GROUP BY SSN) ;

first_name	last_name	total_payment	total_amount
Abbey	Edward	1	10000
James	Born	2	52500
Alba	Jessica	1	4500
Agena	Keiko	1	3000

- How many incidents are covered by each policy? That is, the damage cost of these incidents is larger than the deductible amount of the insurance policies.
  SELECT policy_name, COUNT(*) AS incident_num FROM policy NATURAL JOIN home NATURAL JOIN incident WHERE deductible < damage_cost GROUP BY policy_name ;

policy_name	incident_num
policy-1	1
policy-2	3

- Which policy is the most popular among homeowners?
  WITH T(policy_name, homeowner_num) AS (SELECT policy_name, COUNT(DISTINCT SSN) FROM homeowner NATURAL JOIN home NATURAL JOIN policy GROUP BY policy_name) SELECT policy_name FROM T WHERE homeowner_num = (SELECT MAX(homeowner_num) AS homeowner_num FROM T) ;

policy_name
policy-2

- Which homeowners are not making a payment yet?
  SELECT first_name, last_name FROM homeowner WHERE SSN IN (SELECT DISTINCT SSN FROM homeowner NATURAL JOIN home NATURAL JOIN payment WHERE payment_date IS NULL) ;

first_name	last_name
James	Born
Agena	Keiko

- How much damage_cost did each homeowner suffer from incidents without insurance? Sorted by largest to smallest amount.
  SELECT first_name, last_name, total_cost FROM homeowner NATURAL JOIN (SELECT SSN, SUM(damage_cost) AS total_cost FROM homeowner NATURAL JOIN home NATURAL JOIN incident GROUP BY SSN) ORDER BY total_cost DESC ;

first_name	last_name	total_cost
James	Born	14000
Agena	Keiko	6050
Alba	Jessica	6000
Abbey	Edward	4500

VI. Conclusion

In conclusion, the process of logical design plays an important role for an entire data design compared with conceptual design and physical design: analyzing the SQL and creating the index to an appropriate place while avoiding the normalization as much as possible. Alternatively, the optimization is performed by breaking the normalization. Also, the process of identifying all the required attributes for each entity took us some time. After finalizing the entity and identifying all the attributes, we present it in a logical ER diagram. From a security point of view, the business and users are associated and the necessary privilege management is designed.