

CSE532 Project Report

Credit Card Database

via the Object-Oriented Extension of SQL

Author Name, #SBU ID

October 30, 2015

1 Database Design

Something of design

1.1 Entity-Relationship Design

After a series of consideration, we decided the final Entity-Relationship Model as Figure.1.

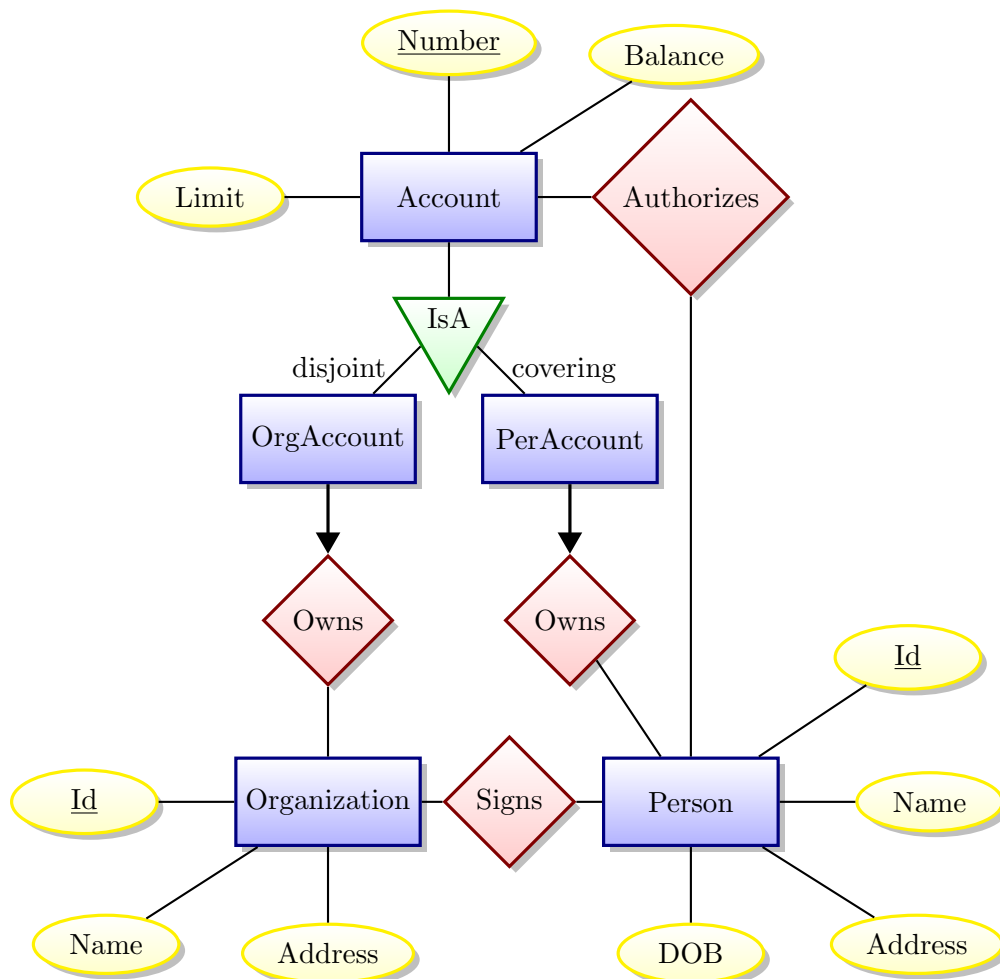


Figure 1: E-R diagram for CCDB

1.1.1 Account Type Hierarchy

The first problem we encountered is that the owner of a credit card account can be either a person or an organization. Although the sample test data has different ids between Person and Organization, we can not assume all data satisfies this condition. If we add a new enum column, labeling the owner type, into Account, then we can distinguish it. But then we have another problem: we can not define a proper foreign key constraint under owner column. And we don't want to create a trigger in order to ensure data consistency.

Therefore, we decided to split the accounts into two types: Personal Account and Organizational Account. All the attributes of Account type are applicable to subtype entities. We defined this IsA relationship that relates Account, PerAccount and OrgAccount. The role Sub refers to PerAccount and OrgAccount, while the role Super refers to Account.

In addition, our partition on Account is disjoint. A credit card account can not be a personal account and an organizational account at the same time. Thus we will not have duplicate tuples when inserting data into non-disjoint subtypes in PostgreSQL (we will discuss it later in section 1.1.2). And the partition also satisfies the covering constraint since the union of the sets of instances PerAccount and OrgAccount equals the set of instances of Account.

We also noticed that Person type and Organization type have many common attributes like Id, Name, Address and they can both be the owner of a credit card account. A supertype, let us call it Owner, is not made for Person and Organization not only because we think those two type is essentially different but also they can not satisfy the covering constraint. If we create the Owner type for the convenience of the foreign key constraint of Account, then we can find some instances like 'p2- John' in Person doesn't own any card.

1.1.2 Person Type Integrity

In our CCDB, a person can participate in many relationship. For example, a person can sign for an organization, can be authorized to use a card, and can also own some credit cards. Naturally, we can divide Person to Signer, AuthorizedUser and so on although the partition is not disjoint. At the beginning we accept the Person Type Hierarchy and it seemed to have no big problem. But soon when we used PostgreSQL to insert data into tables, the trouble rose up.

We used the concept of inheritance from object-oriented databases in PostgreSQL. Two tables, Signer and AuthorizedUser, inherit from Person. If we insert data into supertable Person, the subtables will have no data. That's understandable since the subtable has more attributes and constraints so that it can not extract data directly from supertable. But one person can be both a signer and an authorized user. If we insert data into subtables, we will have duplicate tuples in the view of supertable. For example 'p4 - May' is a signer of organization 'Acme', and also an authorized user of credit card c10. We insert the data of May into Signer and AuthorizedUser then we have two "identical" tuples in supertable Person.

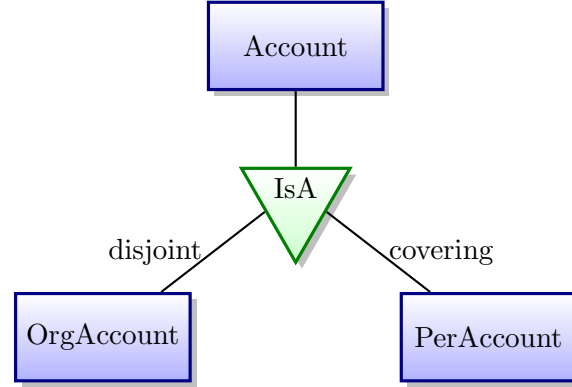


Figure 2: Hierarchy of Account Type

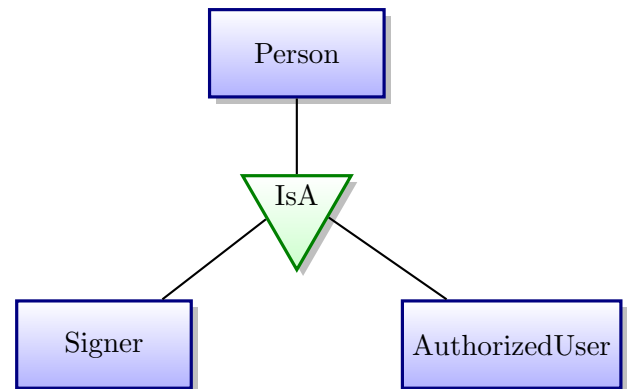


Figure 3: Early partition on Person

We quote “identical” because they are actually different tuples in PostgreSQL database. In PostgreSQL documents, the behavior is described as *“INSERT always inserts into exactly the table specified.”* Therefore, we have two different tuples with same values in table Signer and AuthorizedUser. When we run SELECT query on Person, PostgreSQL returns all data from tables inherited from Person.

To show that, we modified the table to explicitly store OIDs. Then we ran the following SQL command and got the result as shown in Figure.4.

```
SELECT oid, * FROM "Person";
```

Data Output						Explain	Messages	History
	oid oid	Id text	Name text	Address text	DOB date			
1	17087	p4	May	141 George Street Knoxville, TN 37918	1990-08-02			
2	17088	p4	May	141 George Street Knoxville, TN 37918	1990-08-02			

Figure 4: Return query on Person

The tuples were not identical because they came from different tables and they had different OID. The same-value tuples did not violate the primary key constraint because table Person was empty. We can verify that by add ONLY keyword to the query and the query will return 0 result.

```
SELECT oid, * FROM ONLY "Person";
```

In fact, that is not the only problem that inheritance of PostgreSQL rises. When we create a foreign key that points to a supertable, we will encounter exception saying that foreign key does not exist. It is because the supertable is usually empty and PostgreSQL does not extract data from subtables when handle foreign key constraints. This type of behavior has been described in the document:

All check constraints and not-null constraints on a parent table are automatically inherited by its children. Other types of constraints (unique, primary key, and foreign key constraints) are not inherited.

As a result, we need to handle the foreign key constraints which point to a supertable very carefully. The best way is to avoid using that. But we can also create some triggers to ensure the data consistency. Based on the complexity of implementation and the performance on PostgreSQL, we decided not to partition the Person type.

1.2 UML Design

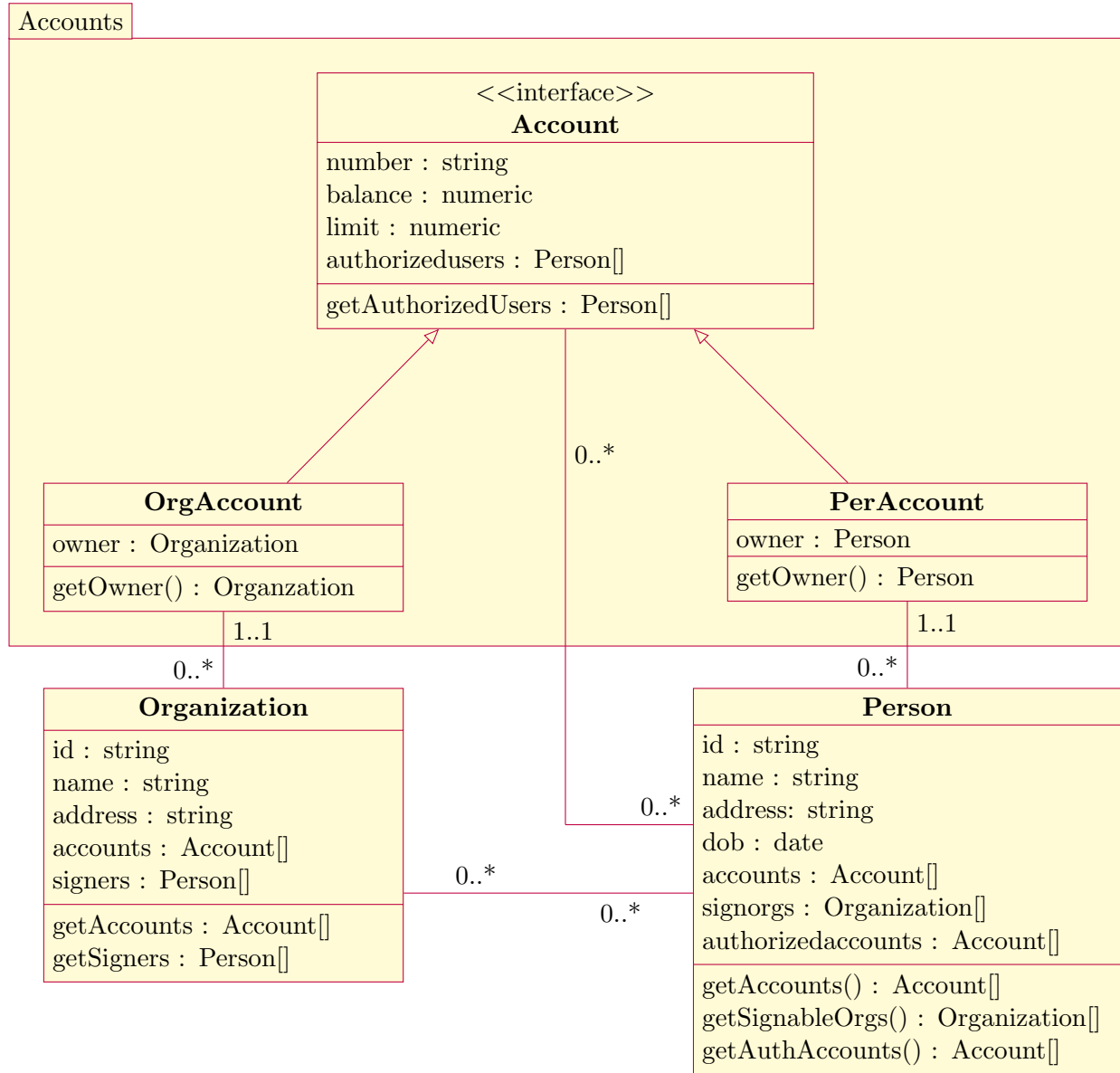


Figure 5: UML diagram for CCDB