ECON485 Introduction to Database Systems

Lecture 03 – Relational Database Design and Normalization

Relational Database Design

The idea behind relational databases is that we can model relations between business objects (in real life and/or in a software running in memory) on the data storage medium.

A relation by definition needs at least two participants. In our view a relation exists between two records representing two business objects.

Since we expect multiple business objects of the same kind (ie. orders, customers, addresses, etc) we use tables to store the records. Each row in a table represents one record.

The relations between individual records are tracked by using the data available.

Relational Database Design

In our initial design we will model the relations between automobile tires and their owners in a tire storage facility.

The business story goes as follows.

In the tire storage facility, customers (ie. tire owners) check-in and check-out their tires. This means they leave a set of tires, and pick-up another set.

This usually happens in order to change tires due to winter. At the beginning of winter, owners switch to their winter tires. This means they check-out the winter tires and they check-in the summer tires. At the end of winter, the reverse happens.

A tire can last for about 6-7 years. So sometimes, tire owners ask for the disposal of their old tire set. This means they are not checked-in at the storage facility but processed elsewhere. However, the same tire owners will probably come back next season and check-in their tires. This time they will not have any tires in storage, so they will have to buy new tires.

So at any time, a tire owner needs to have at least one set of tires on the vehicle, and at most one set of tires in storage.

We usually deal with individuals, so car fleets, etc who have multiple vehicles are of no concern to us.

Relational Database Design

Relational database design begins by identifying the business objects.

We identify business objects by looking at nouns and pronouns, as well as adjectives.

Sometimes these are representing qualities of business objects, so even adjectives can lead to discovery of nouns.

- Example. “A <u>large</u> house” implies houses have a quality related to size. By asking further questions such as “how do you define a large house” you reveal more about the business objects and learn that houses have units (ie. rooms, quantified in whole numbers) and a total usable area (quantified in square meters).
- These will become the properties of the business object “house.”

Relational Database Design

Getting back to our example.

There are three different business objects we are interested in.

Tire Owner

- Owns tires, assembled into tire sets.
- Checks-in and checks-out tire sets.

Tire Set

- Is a combination of tires.
- Should include at least one tire, usually 4.
- Gets checked-in and checked-out.
- Because the user might replace individual tires within season (ie. replace flats), we cannot assume tires in a tire set (on the vehicle) remain the same.
- However, tires in a tire set in storage should remain the same.

Tire

- They have distinguishing properties such as size, brand, and model.
- Type (ie. winter, summer) can be deduced from tire model.
- Normally they have no unique identifier. We may print an ID on the tire surface or stick some identifier while in storage.

Relational Database Design

Relational database design begins by identifying the business objects.

We identify business objects by looking at nouns and pronouns, as well as adjectives.

Sometimes these are representing qualities of business objects, so even adjectives can lead to discovery of nouns.

Example. “A large house” implies houses have a quality related to size. By asking further questions such as “how do you define a large house” you reveal more about the business objects and learn that houses have units (ie. rooms, quantified in whole numbers) and a total usable area (quantified in square meters).

These will become the properties of the business object “house.”

Relational Database Design

Getting back to our example.

- We identify three types of business objects.
  - Tire Owners.
  - Vehicles.
  - Tire Sets.
  - Tires.
- Note that tire owners own both the vehicles and the tire sets, but the tire sets are installed on vehicles.
- Therefore vehicles are present always during check-in and check-out.

Tire Owner

- Owns vehicle(s).
- Owns tire set(s) installed on each vehicle.
- Owns tire set(s) in storage.

Notes:

- Owners can replace vehicles (ie. buy a new car) and use the old tires on the vehicle.
- Therefore tire-sets and their relation with vehicles are **temporary**.

Relational Database Design

So how do we model this relation?

- First we try to design an individual record for each business object.
- A table will be just a collection of these records. So designing a record is equivalent to designing a table.
- This type of model is often called the logical model.

Tire Owner

- OwnerID (unique, created with record)
- Name
- Contact Info
- VehicleID (may be multiple)
- TireSetID (may be multiple)

Vehicle

- VehicleID (unique, created with record)
- License Plate number

Relational Database Design

Managing the multiplicities is an important task.

- Since an owner is associated with many vehicles. Should we store the data of ownership within the owner. If yes how?
- Tables are required to have predefined columns.
- We should not arbitrarily add columns.

Tire Owner

- OwnerID (unique, created with record)
- Name
- Contact Info
- ~~VehicleID (may be multiple)~~
- VehicleOneID
- VehicleTwoID (how about three vehicles?)
- TireSetID (may be multiple)

Relational Database Design

Using relationships, we have two alternatives.

Since all vehicles have one owner, let the vehicles store the owners' info, so that the relationship is formed by using the info in the vehicle.

Finding all vehicles associated with an owner becomes looking for and matching vehicle records (table rows) that **store the owner ID of that particular owner**.

So there should be **a search** through the Vehicle table.

Tire Owner

OwnerID (unique, created with record)

… (All else)

~~VehicleID (may be multiple)~~

Vehicle

VehicleID (unique, created with record)

OwnerID (non-unique, created with record)

License Plate number

Relational Database Design

Using relationships, we have two alternatives.

- Note that, since multiple vehicles may be owned by the same owner, multiple records may contain the same OwnerID value.
- Therefore, the property (field, column, etc) OwnerID will be
  - Unique in the Tire Owner table (because owners are unique)
  - Non-Unique in the Vehicle table (because the references to owners may not be unique)

Tire Owner

- OwnerID (**unique**, created with record)
- … (All else)
- ~~VehicleID (may be multiple)~~

Vehicle

- VehicleID (unique, created with record)
- OwnerID (**non-unique**, created with record)
- License Plate number

Relational Database Design

Using relationships, we have two alternatives.

- The relationships are usually classified with their multiplicities.
- 1-1 relationships.
- 1-many relationships. Example tire owner(1) – vehicle (many).
- Many-many relationships.

Note that many to many relationships are harder to model.

Tire Owner

- OwnerID (unique, **primary key**)
- … (All else)
- ~~VehicleID (may be multiple)~~

Vehicle

- VehicleID (unique, **primary key**)
- OwnerID (non-unique, **foreign key**)
- License Plate number

Relational Database Design

Using relationships, we have two alternatives.

- The first alternative we covered so far requires you to store foreign keys at both end of the relation.
- Suppose vehicles may have multiple owners…
  - We add a foreign key as vehicleID to owners' record.
  - As a result we will have multiple tire owner records per vehicle.

Tire Owner

- OwnerID (**unique**, **primary key**)
- … (All else)
- VehicleID (non-unique, **foreign key**)

Vehicle

- VehicleID (unique, **primary key**)
- OwnerID (non-unique, **foreign key**)
- License Plate number

Relational Database Design

Using relationships, we have two alternatives.

- The first alternative we covered so far requires you to store foreign keys at both end of the relation.
- Suppose vehicles may have multiple owners…
  - We add a foreign key as vehicleID to owners' record.
  - As a result we will have multiple tire owner records per vehicle.
  - OwnerID will not be sufficient to be a primary key.
  - We solve the primary key problem by **adding a new column to identify rows**, and we will auto-increment its value as we add new rows.
  - This new column has no place in the business model, but has become a necessity for relational database model.

Owner

Owner ID	VehicleID	Name	**RowID (auto)**
1	1	Bora	**1**
1	2	Bora	**2**
2	2	Tufan	**3**

Vehicle

VehicleID	Owner ID	LicensePlate	**RowID (auto)**
1	1	06A01	**1**
2	1	06A02	**2**
2	2	06A02	**3**

Relational Database Design

Using relationships, we have two alternatives.

- The first alternative we covered so far requires you to store foreign keys at both end of the relation.
- Suppose vehicles may have multiple owners…
  - We add a foreign key as vehicleID to owners' record.
  - As a result we will have multiple tire owner records per vehicle.

Tire Owner

- OwnerID (**unique**, **primary key**)
- … (All else)
- VehicleID (non-unique, **foreign key**)

Vehicle

- VehicleID (unique, **primary key**)
- OwnerID (non-unique, **foreign key**)
- License Plate number

Relational Database Design

Using relationships, we have two alternatives.

The second alternative is to model the relationship as a separate entity.

The relationship object will have references to both sides of the relationship.

Owner		
OwnerID (**PK**)	Name	
1	Bora	
2	Tufan	

Vehicle		
VehicleID (**PK**)	License Plate	
1	06A01	
2	06A02	

Vehicle Ownership		
RelationID (**PK**)	OwnerID (**FK**)	VehicleID (**FK**)
1	1	1
2	1	2
3	2	2

Relational Database Design

Owner

Vehicle

TireSet

VehicleOwnership

TireSetStorage

- Check-in vehicle and time
- Check-out vehicle and time

Tire

Normalization

In the older times, disk storage space and memory was really (really) scarce.

- Computers with **4 kilobytes** of memory were used to send a shuttle to the moon (and back) in 1969.
- Disks capacities of **20 megabytes** were common to banks in 70s, to personal computers in 90s.

So everyone tried to minimize the memory footprint.

Normalization

Normalization is a technique to minimize the memory footprint of data stored in tables by eliminating repeating data.

- Normalization has a cost in the complexity (ie. number of tables) you end up with.
- There are multiple levels of normalization. The further you go, the more complex your models become.

Levels of normalization

Normalization

An entity is in First Normal Form (1NF) when all tables are two-dimensional with no repeating groups.

- A row is in first normal form (1NF) if all underlying domains contain atomic values only.
- 1NF eliminates repeating groups by putting each into a separate table and connecting them with a one-to-many relationship.

For 1NF, make a separate table for each set of related attributes and uniquely identify each record with a primary key.

- Eliminate duplicative columns from the same table.
- Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

Normalization

An entity is in Second Normal Form (2NF) when it meets the requirement of being in First Normal Form (1NF) and additionally:

- Does not have a composite primary key. Meaning that the primary key can not be subdivided into separate logical entities.
- All the non-key columns are functionally dependent on the entire primary key.
- A row is in second normal form if, and only if, it is in first normal form and every non-key attribute is fully dependent on the key.
- 2NF eliminates functional dependencies on a partial key by putting the fields in a separate table from those that are dependent on the whole key.

For 2NF, we handle many to many relationships using the second alternative (ie. create relationship tables).

Normalization

An entity is in Third Normal Form (3NF) when it meets the requirement of being in Second Normal Form (2NF) and additionally:

- Functional dependencies on non-key fields are eliminated by putting them in a separate table. At this level, all non-key fields are dependent on the primary key.
- A row is in third normal form if and only if it is in second normal form and if attributes that do not contribute to a description of the primary key are moved into a separate table.

To achieve 3NF, for large tables with rows containing a lot of values (columns), we separate them into two tables.

- Queries to identify the exact row is carried out in the **lookup table**. The index table has only columns containing keys (primary or foreign).
- The query on the index table gives us a foreign key that matches the primary key value for the remaining of the row in the **data table**.

Normalization

Achieving 1NF is obvious.

- From a junior business analyst’s point of view 1NF is good enough, 2NF is better.
- A senior business analyst will model objects immediately at 2NF.

Achieving 2NF will help you model your queries because you will only have 1-1 and 1-many relations in 2NF.

- From a programmer’s point of view 2NF matches most data structures she uses, and is good enough. On this level 3NF is better but not necessary.

Achieving 3NF will help you in search performance, managing locks during row updates, etc.

- From a database administrator’s point of view 3NF is good enough.

There are even further levels of normalization, but we will not cover them.

- In your exams, homework assignments, and team project you will be usually **required to achieve 2NF**, but not 3NF.