

Data Modeling & Use Cases

Professor Larry Heimann
Application Design & Development
Information Systems Program

Why Data Modeling?

- A bridge to convert requirements into a database
- Can be done early in the process
- Cheaper to fix errors at this stage
- Understandable to users and developers
- Data is critical!
- Entity-Relationship modeling is fairly easy to do

Class Problem

A house for sale has its address listed in the MLS directory for maximum exposure (and, in theory, a quicker sale). In addition, a MLS listing must have an asking price, listing agent, listing and expiration dates, and commissions for listing and selling agents. All listings last for 120 days and can be renewed. A house can only be listed once at any given time, but it can be relisted with a different agent once the original listing has expired. All agents must belong to a licensed real estate agency, although some agency are sole-proprietorships with one person and some are corporations with many agents. The MLS directory also has a sales section which shows the selling price, selling date and selling agent as well as the reference to the listing.

Draw out a simple ERD to capture the essential information in this example.

ERD :: MLS Example

List of all nouns

House

- address

~~MLS Directory~~

MLS Listing

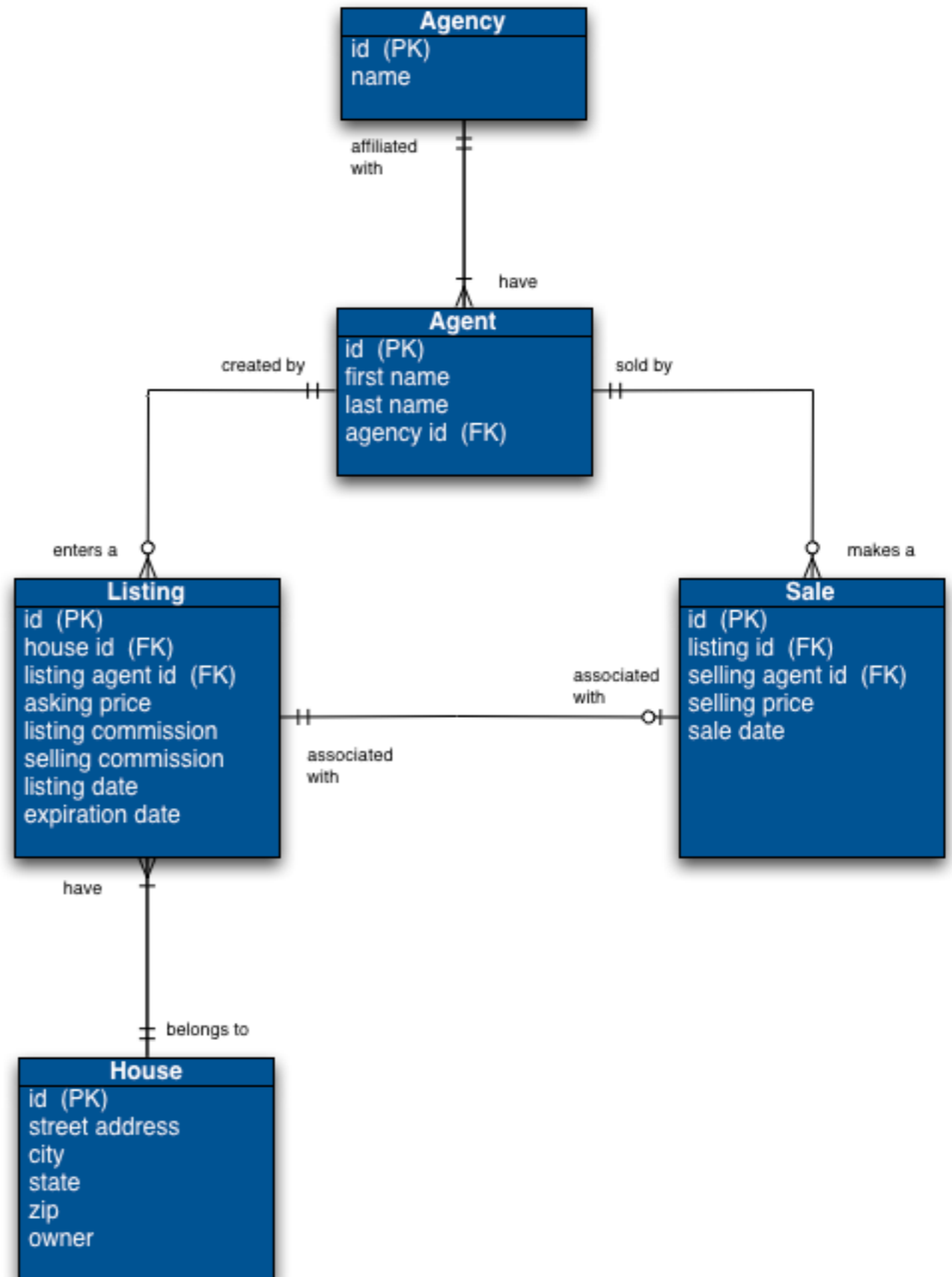
- price
- listing agent
- listing date
- expiration date
- commission - listing
- commission - selling

Agent (general)

Agency

MLS Sale

- selling price
- selling date
- selling agent
- listing reference



From the makers of PostgreSQL...

Don't use table inheritance



Don't use [table inheritance](#) . If you think you want to, use foreign keys instead.

Why not?

Table inheritance was a part of a fad wherein the database was closely coupled to object-oriented code. It turned out that coupling things that closely didn't actually produce the desired results.

When should you?

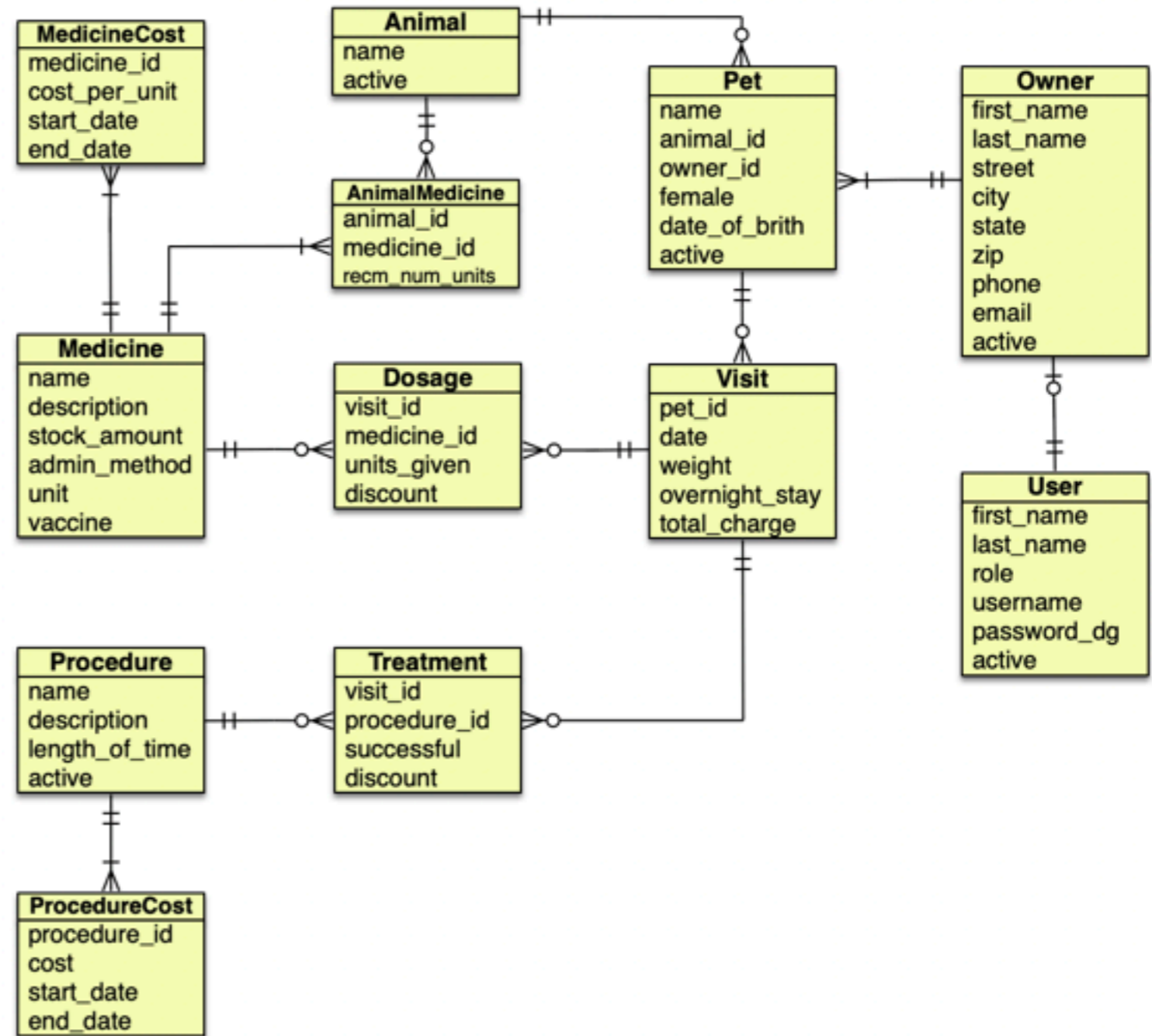
Never ...almost. Now that table partitioning is done natively, that common use case for table inheritance has been replaced by a native feature that handles tuple routing, etc., without bespoke code.

One of the very few exceptions would be [temporal_tables](#)  extension if you are in a pinch and want to use that for row versioning in place of a lacking SQL 2011 support. Table inheritance will provide a small shortcut instead of using `UNION ALL` to get both historical as well as current rows. Even then you ought to be wary of [caveats](#)  while working with parent table.

Summary :: ERD

- Identify all entities and attributes
- Define relationships between entities
- Determine connectivity and transform many-to-many relationships
- Ascertain whether required/optional
- Recognize that data modeling is usually iterative process

A project we
are studying



Pittsburgh Animal Treatment Services ERD

Converting ERD to database design

Database Design in 3NF

owners (id, first_name, last_name, street, city, state, zip, phone, email, active)

pets (id, name, animal_id, owner_id, female, date_of_birth, active)

visits (id, pet_id, date, weight, overnight_stay, total_charge)

animals (id, name, active)

medicines (id, name, description, stock_amount, method, unit, vaccine)

medicine_costs (id, medicine_id, cost_per_unit, start_date, end_date)

animal_medicines (id, animal_id, medicine_id, recommended_num_of_units)

visit_medicines (id, visit_id, medicine_id, units_given, discount)

procedures (id, name, description, length_of_time, active)

treatments (id, visit_id, procedure_id, successful, discount)

procedure_costs (id, procedure_id, cost, start_date, end_date)

notes (id, notable_type, notable_id, title, content, user_id, date)

users (id, first_name, last_name, role, username, password_digest, active)

Underlines:

Solid underlined fields are primary keys;

Dotted underlined fields are foreign keys;

Double underlined fields are composite keys that are both primary and foreign keys.

Database Design Notes:

1. Strictly speaking, having zip code in the owners table creates a transitive dependency, but given the limited size of the system (the greater Pittsburgh area) there is no need to normalize and move zip code and primary city & state into its own table.
2. To minimize the impact of rounding errors associated with floats, we will follow the industry norm of recording all costs as integers with the base monetary unit being cents, not dollars.
3. In a similar vein, length_of_time in the procedures table is recorded as an integer representing the number of minutes the procedure is expected to take.
4. A medicine's current cost is determined by finding the one medicine_cost record that has a NULL value in end_date. Similarly a procedure's current cost is found by identifying the one record that has a null end_date. Triggers will need to set up to automatically add the end date to the record prior to adding a new record to these tables.
5. Since usernames are used for uniquely identifying users during login, the values of username must be unique even though it is not a primary key.

Prof. H's first rule of software development:

It always takes longer than you think to develop software.

Corollary to the first rule:

Start your work early!

Prof. H's second rule of software development:

Add safeguards whenever you can. You can't imagine all the ways users will try to use and abuse your software.

Corollary to the second rule:

Add safeguards to the database because you can't assume it will always be coupled with your software.

Creating a data dictionary

| pets | | | |
|----------------|-----------|--|--------------|
| Field | Data Type | Description | Example Data |
| id (PK) | INT | The ID for this table (auto-increment) | 3801 |
| name* | STRING | The first name of the owner | Dusty |
| animal_id (FK) | INT | A foreign key indicating the animal type of the pet | 2 |
| owner_id (FK) | INT | A foreign key indicating the owner of the pet. | 243 |
| female | BOOLEAN | A boolean that is true if the pet is femal and false otherwise | FALSE |
| date_of_birth | DATE | Date when pet was born, if known | 1999-10-19 |
| active | BOOLEAN | 1 if active; 0 otherwise | TRUE |

| visits | | | |
|----------------|-----------|--|--------------|
| Field | Data Type | Description | Example Data |
| id (PK) | INT | The ID for this table (auto-increment) | 2 |
| date* | DATE | The date of the pet's visit | 2020-01-31 |
| pet_id (FK) | INT | A foreign key linking a pet to a particular visit | 17 |
| weight | FLOAT | The weight of the pet at the time of visit, measured in pounds | 12.0 |
| overnight_stay | BOOLEAN | A boolean that is true if the pet is femal and false otherwise | TRUE |
| total_charge | FLOAT | Amount in dollars the owner was charged for the visit | 116.40 |

* Indicates a non-key attribute that is required. All primary and foreign keys are required.

Dealing with primary keys

- Purpose of primary keys
- Problems with Rails PKs
 - duplicate entries sneak by
 - generic name of *id* is meaningless
 - existence of redundant keys
 - can't use `USING` in joins or natural joins
- Composite keys

Use cases defined

“A use case is a methodology used in system analysis to **identify, clarify, and organize system requirements.**

The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

It consists of a group of elements (for example, classes and interfaces) that can be used together in a way that will have an effect larger than the sum of the separate elements combined. The use case should contain all system activities that have significance to the users. A use case can be thought of as a collection of possible scenarios related to a particular goal, indeed, the use case and goal are sometimes considered to be synonymous.”

Use cases characteristics

- Organizes functional requirements
- Models the goals of system/actor (user) interactions
- Records paths (called scenarios) from trigger events to goals
- Describes one main flow of events (also called a basic course of action), and possibly other ones, called exceptional flows of events (also called alternate courses of action)
- Is multi-level, so one use case can use the functionality of another.

Use case actors

- Any human actor interacting with a graphical user interface is almost always considered a complex actor
- Humans limited to read-only access may be considered average actors
- If the system is interacting with an external API (such as Google Calendar, in this case), then it can be considered a simple actor.

| Actor List | | |
|----------------------------|---------|--|
| Actor Name | Level | Description |
| Vet | Complex | This user administers the site and has complete control over all operations using an online graphical user interface. |
| Assistant | Complex | This user has read access to all aspects of the PATS system and has write access to owners, pets and visits, but not medicines or procedures |
| Owner | Complex | This user can log in to see a list of pets, view and edit some of the details of their pets as well as their personal information |
| Guest | Average | This user has read-only access and is limited to semi-static content, such as 'About PATS' and 'Privacy Policy' pages |
| Google Calendar API | Simple | System is registered with, and interacts with, the Google Calendar API to handle scheduling of visits |

Use case levels

- A-level use cases are “must haves” — without it the system is of little or no value
- B-level use cases are “want to have” — they make life much easier, but system is still usable
- C-level use cases are “like to have” — nice, but could easily live without
- Build functionality in order of priority; i.e., build out A-level functionality before working on B- or C-level cases

| A-Level Use Cases | | |
|---------------------------|-----------------------------|--|
| Use Case Name | Actor(s) | Description |
| Add Owner | Vet, Assistant | The user can add a new owner to the system. |
| Edit Owner | Vet, Assistant, Owner | The user can edit an existing owner in the system. Owners can only edit their own personal data; vets can edit everyone. |
| Delete Owner | Vet | The user can delete an owner from the system. Used sparingly as most will simply be deactivated and the record retained. |
| List All Owners | Vet, Assistant | The user can view a list of all the owners in the system (active and inactive lists separated out). |
| View Owner Details | Vet, Assistant, Owner | The user can view the details of the owner's record in the system along with list of the owner's pets and visit history. Owners can only see their own data. |
| Add Pet | Vet, Assistant | The user can add a new pet to the system. |
| Edit Pet | Vet, Assistant | The user can edit an existing pet in the system. |
| Delete Pet | Vet | The user can delete a pet from the system. Used sparingly as most will simply be deactivated and the record retained. |
| List All Pets | Vet, Assistant | The user can list all pets in the system, including both active and inactive pets (but in separate lists). |
| View Pet Details | Vet, Assistant, Owner | The user can view the details of a pet's record in the system, along with a list of all the pet's prior visits to PATS. |

CRUD operations

Create

Read

Update

Delete