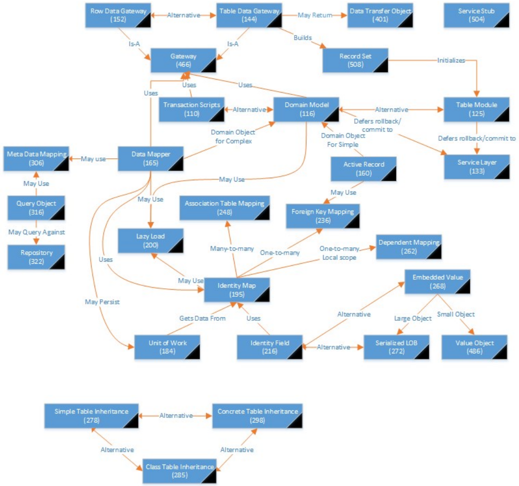
Inheritance vs. Relational Databases

Overview

Mapping to Relational Databases



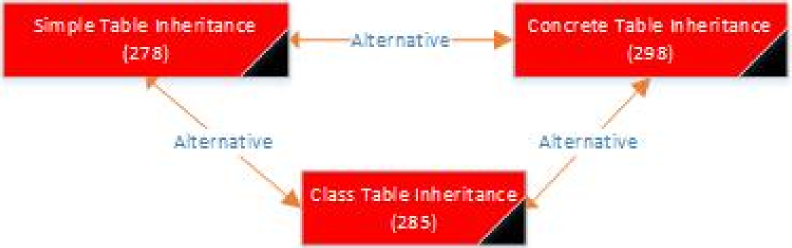
Paradigm Chasm

- Paradigm: a typical example or pattern of something; a model. (cite - google)
- Paradigm chasm: A situation where two paradigms abut and conflict
- Structure of information in objects conflicts with relational database structures
- We will have to design a translation between them

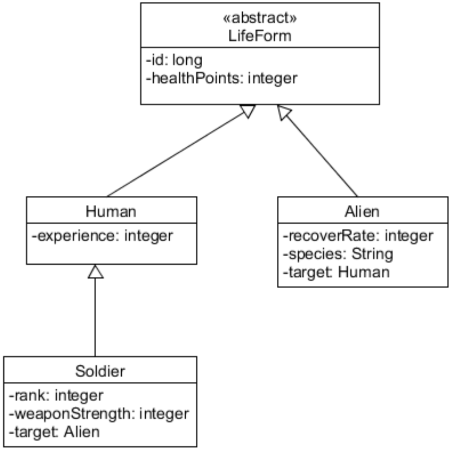
Other Paradigm Chasms?

Find other paradigm chasms in SE and in life

Where We'll Start


Simple Table Inheritance (278)
Concrete Table Inheritance (298)
Class Table Inheritance (285)
Alternative
Alternative
Alternative

Example



- What fields do objects of each type hold?
- How could we structure a db to hold them?

Make up some data

Make up some objects from this example so that we can play with how they get stored in each pattern

What do we worry about as we cross the chasm?

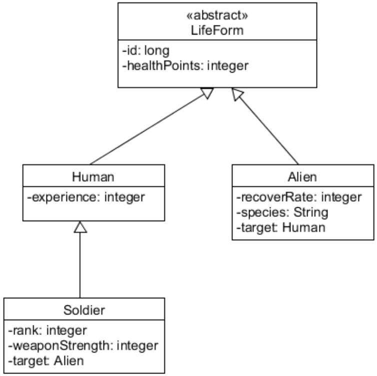
- How efficient is the storage? Will all columns be used in all rows?
- If things are spread across tables, how do we connect them and what does it cost to retrieve them?
- If something changes in the inheritance hierarchy (add, delete, move up, move down), how hard is it to adjust the DB? Remember: we already have huge amounts of data in the production DB, so any change is going to require reconfiguration of an existing DB.

Definition

One table with columns for every attribute in every class

- Need a column to tell us which type each row holds
- Need a column for every field in every class

Our Example



LifeForm:

- id:long
- type: integer (0 = human, 1 = soldier, 2 = alien)
- healthPoints: integer
- experience: integer
- species: varchar(25)
- recoverRate: integer
- rank: integer
- weaponStrength: integer
- soldierTarget: long (is a LifeForm id)
- alienTarget: long (is a LifeForm id)

Our Three Questions

- How efficient is the storage? Will all columns be used in all rows?
- If things are spread across tables, how do we connect them and what does it cost to retrieve them?
- If something changes in the inheritance hierarchy (add, delete, move up, move down), how hard is it to adjust the DB? Remember: we already have huge amounts of data in the production DB, so any change is going to require reconfiguration of an existing DB.

Bonus question: Can we have friendly fire?

Things to Think About From Reading

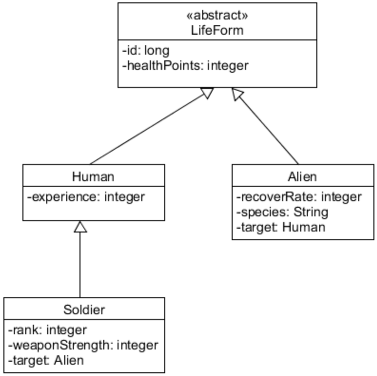
- “The single table may end up being too large, with many indexes and frequent locking, which may hurt performance. You can avoid this by having separate index tables that either list keys of rows that have a certain property or that copy a subset of fields relevant to an index”
- “You only have a single namespace for fields, so you have to be sure you don’t use the same name for different fields. Compound names with the name of the class as a prefix or suffix help here.”

Definition

One table for each concrete class

- Substitution principle:
  - Formally: Let $\phi(x)$ by a property provable for objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T
  - Informally: An instance of the subclass must be able to be substituted for any use of the superclass. Subclasses must enforce an "is-a" relationship with the superclass?
  - What does this do to id numbers if each concrete class has a table?

Our Example



Human:

- lifeFormId:long
- healthPoints: integer
- experience: integer

Alien:

- lifeFormId: long
- healthPoints: integer
- recoverRate: integer
- species: varchar(25)
- target: long (is a lifeForm id)

Soldier:

- lifeFormId:long
- healthPoints: integer
- experience: integer
- rank: integer
- weaponStrength: integer
- target: long (is a lifeForm id)

Our Three Questions

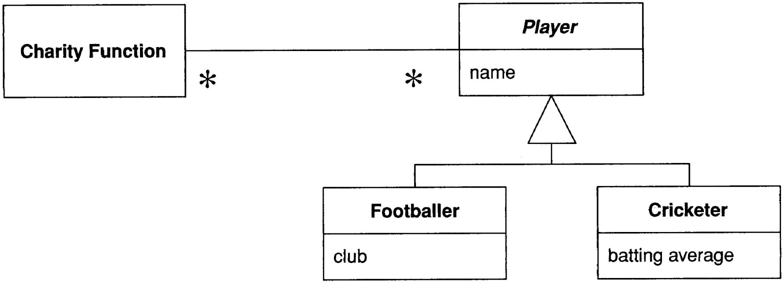
- How efficient is the storage? Will all columns be used in all rows?
- If things are spread across tables, how do we connect them and what does it cost to retrieve them?
- If something changes in the inheritance hierarchy (add, delete, move up, move down), how hard is it to adjust the DB? Remember: we already have huge amounts of data in the production DB, so any change is going to require reconfiguration of an existing DB.

Bonus question: Can we have friendly fire?

Things to Think About From Reading

- "Each table is self-contained and has no irrelevant fields. As a result it makes good sense when used by other applications that aren't using the objects."
- When talking about keys:
    - "A classic example of where you need this is if you have a collection of players and you're using Identity Field (216) with table-wide keys. If keys can be duplicated between the tables that map the concrete classes, you'll get multiple rows for a particular key value."
    - "You need a key allocation system that keeps track of key usage across tables; "
    - "also, you can't rely on the database's primary key uniqueness mechanism."
    - "You can get around some of this by not having fields that are typed to the superclass, but obviously that compromises the object model"
    - "For compound keys you can use a special key object as your ID field for Identity Field (216). This key uses both the primary key of the table and the table name to determine uniqueness."

Things to Think About From Reading



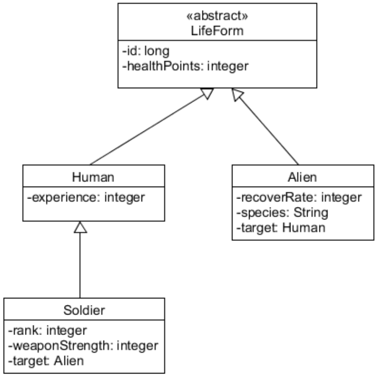
- “To implement referential integrity you need a link table that contains foreign key columns for the charity functions and for the player.”
- “The problem is that there’s no table for the player, so you can’t put together a referential integrity constraint for the foreign key field that takes either footballers or cricketers.”
- “Your choice is to ignore referential integrity or must multiple link tables, one for each of the actual tables in the database.”

Definition

One table for each class in the hierarchy

- What does this do to id numbers if each class has a table?

Our Example



LifeForm:

- id: Long
- healthPoints: Integer

Human:

- id:long
- experience: integer

Alien:

- id: long
- species: varchar(25)
- target: long (is a LifeForm ID)

Soldier:

- id:long
- rank: integer
- weaponStrength: integer
- target: long (is a LifeForm id)

Our Three Questions

- How efficient is the storage? Will all columns be used in all rows?
- If things are spread across tables, how do we connect them and what does it cost to retrieve them?
- If something changes in the inheritance hierarchy (add, delete, move up, move down), how hard is it to adjust the DB? Remember: we already have huge amounts of data in the production DB, so any change is going to require reconfiguration of an existing DB.

Bonus question: Can we have friendly fire?

Things to Think About From Reading

- "How to link the corresponding rows of the database tables"
- "Use a common primary key value. Since the superclass table has a row for each row in the other tables, the primary keys are going to be unique across the tables."
- "An alternative is to let each table have its own primary keys and use foreign keys into the superclass table to tie the rows together. This strategy would let the dbms enforce referential integrity."

Definition

Tradeoffs are all about duplication of data, wasted space, and speed of access So, we have to think about:

- What is required to access an object?
- What is required to store an object?
- How is the database affected by changes to the classes?
- How is the database affected by changes to the structure of the hierarchy?
- How is the database affected by moving instance variables up or down the hierarchy?

Single Table Inheritance

- One table for access and updates
- That one table will be locked for updates - lock contention
- Robust to changes in class design and hierarchy
- Wasted space
- Logic for figuring out which class something belongs to is extra

Concrete Table Inheritance

- No joins on access if you know the class and only one table updated on writes
- Searching for all members of a superclass requires a join with all descendent classes
- No superclass lock contention
- Brittle to changes in the classes
- Change in a superclass can affect multiple tables
- Robust to changes in hierarchy

Class Table Inheritance

- Has a simple relation to the structure of the code
- Is brittle to changes in the hierarchy, but not to changes in the classes
- Requires joins on access which is a performance issue
- Updates affect multiple tables
- Lock contention on abstract superclass tables