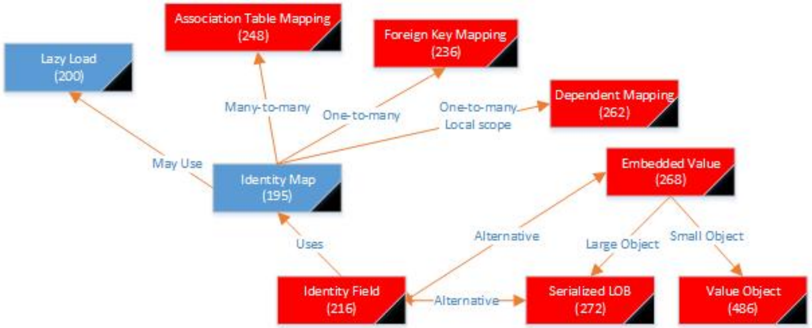
08/19/2020

Overview

Where are we?

Patterns for mapping relationships between objects into a relational database



Storing Relationships

	Objects	**Relational DB**
one-to-one representation	run-time memory reference (containment)	key into another table
one-to-many representation	Object holds pointer to list (containment of collection)	Each item in the collection stores the key of the object containing it
many-to-many representation	Each object holds a pointer to a list (in both directions)	Separate table with row containing pairs of keys that are related

Identity Field

Saves a database ID field in an object to maintain identity between an in-memory reference object and a database row

Key Choices

Meaningful Key	A key that is an attribute of the object; sometimes called a natural key.
Meaningless Key	Randomly generated - hidden from user

- Meaningful is risky because keys should be unique and immutable

Simple Key	Uses only one database field
Compound Key	Combination of values from multiple fields

- Compound can make sense when one table makes sense in the context of another (orders and line items)

Ponder . . .

How do compound keys work if we do Concrete Table Inheritance?

Key Uniqueness

Database Unique Unique across every row of every table

Table Unique Unique across every row of a single table (minimum requirement)

- Table unique is usually sufficient
- Choice really isn't binary - can be unique across a set of tables
- Scope of uniqueness defines the types that can be managed by a single Identity Map

Ponder . . .

How is this choice affects by our inheritance mapping pattern choice?

Representing Identity Field in an Object

- Simple key - Just a key field is fine
- Compound key
  - A key class can contain logic for building, comparing, sequencing, etc.
  - Layer Supertype is a good idea if the compound key structure spans multiple classes. It can hold the common behavior with subclasses for unique combinations
- If the db had multiple instances, have to worry about key collisions

Key Collisions

What could cause them? What strategies could you use to prevent them?

Getting a new key (part 1)

Autogenerate easy for simple, table unique keys

- Usually, db can do it for you
- Foreign keys will require separate inserts because you don't get the key until you insert the row - but want them all inserted before you commit

Database counter DB gives sequence independent of insertions

Globally Unique ID (GUID) number generated on one machine that is unique across all machines in space and time.

- Usually combination of things like ethernet card addresses, time of day in nanoseconds, chip ID numbers, etc.
- Makes for BIG keys: hard to type and may cause performance problems on retrieval via indexes

Getting a new key (part 2)

Table Scan Search the table for available key

- Max key + 1
- Read locks the entire table
- Only good when inserts are rare

Key Table Keep next available key in a table

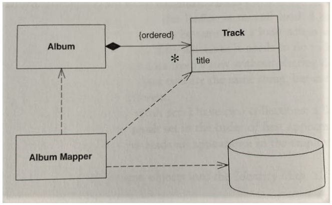
- Two columns: context and next key
- Table unique keys will have one row for each table

Ponder . . .

Suppose that we are using a Key Table and we make getting a new key a separate transaction from the insertion of a new row. What could go wrong? What did it cost us?

Dependent Mapping

"Has one class perform the database mapping for a child class"



- I don't think he means "child class" like inheritance. I think he means "contained"
- Note that the dependent objects can have only one owner
- Mapping can be done by the owner or by a separate Data Mapper class
- Dependent object doesn't have an identify field and can't be in an Identity Map
- Key in db is often a compound key with the owner's id included because that must be there as a foreign key.

Foreign Key Mapping

Maps an association between objects to a foreign key reference between tables

One-To-One Put the ID of the reference in the referring table

One-To-Many Put ID of the containing object in the contained rows

How do we code that?

Look at the RelationshipsInDBs.pdf I gave you and figure out how that SQL matches these descriptions

One-To-Many Updates (part 1)

How do we update the DB when the contained list gets updated?

Delete and Insert Delete all contained objects and reinsert them

- Only works if the contained objects are Dependent Mappings

Adding a Back Pointer Making the relationship by-directional in the object model

- adding a backwards one-to-one relationship to the object model

Ponder . . .

How does that back pointer help?

One-To-Many Updates (part 2)

Diff the Collection When we go to persist the update, diff the list in the object to see what needs to change

- Compare with current DB
- Compare with copy of what was originally retrieved

What if there are multiple copies in memory?

How would these two options work if there could be multiple copies of the owning object in memory?

Association Table Mapping

“Saves an association as a table with foreign keys to the tables that are linked by the association”

- This is the way you were taught to story many-to-many relationships in CSC371
- Retrieving the list one item at a time is very expensive!
- Joining the tables lets you get all of the data with one query
- Link table is essentially like a Dependent Mapping, so updates are easy.