OBJECT-RELATIONAL MAPPING

1/28/2025

SANDUNI AALOKA

CONTENT

OOP AND RDMS

• The object-oriented paradigm features concepts such as objects, classes, attributes, specialization or inheritance, and associations.
• The relational database paradigm involves tables consisting of records, columns with data, primary keys and foreign keys.
• When objects are stored in a relational database, these object-oriented concepts have to be translated into the database paradigm.
• This is not straightforward due to the mismatch between the object model and the relational database which is called **impedance mismatch**.

LAYERED ARCHITECTURE

• The layered(n-tier) architecture is a solution for data persistence.
• Needs to
  • build SQL statements for CRUD operations
  • handle data types in objects for data fields in tables
  • handle data values for special cases (ex:- empty strings, date formats, null values, etc.)
  • handle IDs, keys,..

LAYERED ARCHITECTURE

- Problems with traditional approaches,
  - Tedious and requires lots of code.
  - Extremely error-prone.
  - Non-standard SQL ties the application to specific databases.
  - Vulnerable to changes in the object model.
  - Difficult to represent associations between objects.

ORM

- Converts data between relational databases and object-oriented programming languages.
- It creates a model of the object-oriented program with a high level of abstraction. The mapping describes the relationship between an object and data without knowing how the data is structured.
- It eliminates the need to create a data layer tier (data layer is implicit).

ex: Hibernate
Prisma
Sequelize
TypeORM

TYPES OF ORM

- **Active Record Pattern**
  - Maps data within the structure of objects in the code. (e.g:- Ruby on rails, Laravel's Eloquent).
  - Pros:
    - Simple
    - Easy to learn and understand
  - Cons:
    - High database coupling (and testing)
    - Performance bottlenecks

TYPES OF ORM

- **Data-mapper Pattern**
  - Decouple the business logic in the objects from the database. (e.g:- Java Hibernate, Doctrine-Symfony)
  - Pros:
    - Greater flexibility between domain and database.
    - More performant(compared to AR).
  - Cons:
    - Hard to set-up.

ORM VS SQL

- Most RDs support SQL to build data interfaces and applications.
- Need a lot of works, but it is more flexible and detailed than an ORM abstraction.
- Native Querying with SQL
  - Developer highly responsible for safety and security.
- SQL Query Builders
  - Add a layer of abstraction over the raw SQL without masking all of the underlying details.
  - Still developer needs to understand the database structure.

ADVANTAGES OF ORM

- Productivity - Eliminate repetitive code, Fast development of application.
- Maintainability - Few lines of code.
- Performance - Minimize row reads and joins.
- Database vendor independence.
- Transaction management.
- Less error prone.
- Code reuse.
- Reduced testing.
- Lets business code to access objects rather than database tables.
- Hides details of SQL queries from OO logic.
- No need to deal with database implementation, only deal with domain objects.

DISADVANTAGES OF ORM

• Performance issues due to extra-generated code.

• Developer needs to know SQL - High-level abstractions don't always generate the best SQL code.

• Sometimes create a poor/incorrect data mapping.

• A poorly-written ORM layer effects on schema and database migrations.

ORM CONCEPTS

Relation/Table - Class

Record/Row/Tuple - Object

Attribute/Column - Member/Field

Relationship - Composition/ Aggregation

Hierarchy(is-a) - Inheritance

ORM ENTITIES

- Model collections of real-world objects of interest to the application.
- Have properties/attributes of database data types.
- Participate in relationships.
- Have unique ids consisting of one or more properties.
- Are persistent objects of persistent classes.
- Correspond to database rows of matching unique id.

VALUE OBJECTS

- Persistent objects can be entities or value objects.
- Value objects can represent E/R composite attributes and multivalued attributes.

ex:

  - One address consisting of several address attributes for a customer.
  - Programmers want an object for the whole address, hanging off the customer object.

- Value objects provide details about some entity, have lifetime tied to their entity, and don't need own unique id.

CREATING UNIQUE IDS

• A new entity object needs a new id, and the database is holding all the old rows, so it is the proper agent to assign it.

• This can't be done with standard SQL insert, which needs predetermined values for all columns.

• Every production database has a SQL extension to do this.

ex:

- Oracle's sequences
- SQL Server's auto-increment data type

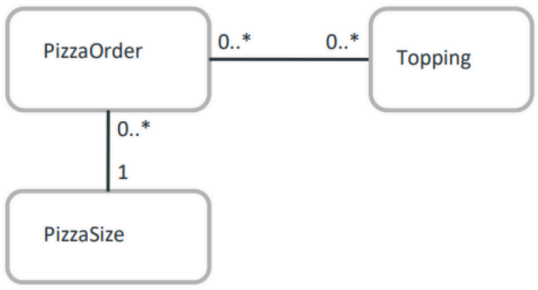
• The ORM system coordinates with the database to assign the id, in effect standardizing an extension of SQL.

ENTITY MODEL

• Uses UML-like diagrams to express object models that can be handled by this ORM methodology.

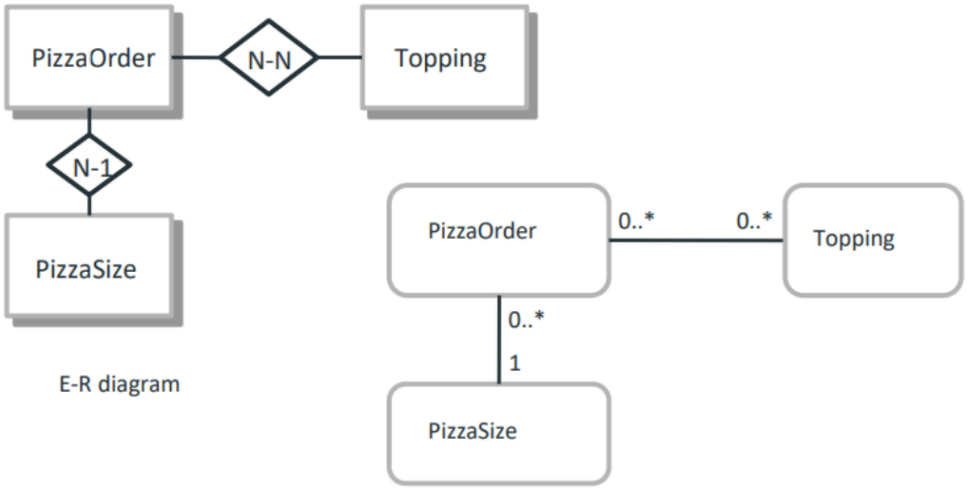
• Currently handles only binary relationships between entities, expects foreign keys for them in database schema.

• Supports updates and transactions.

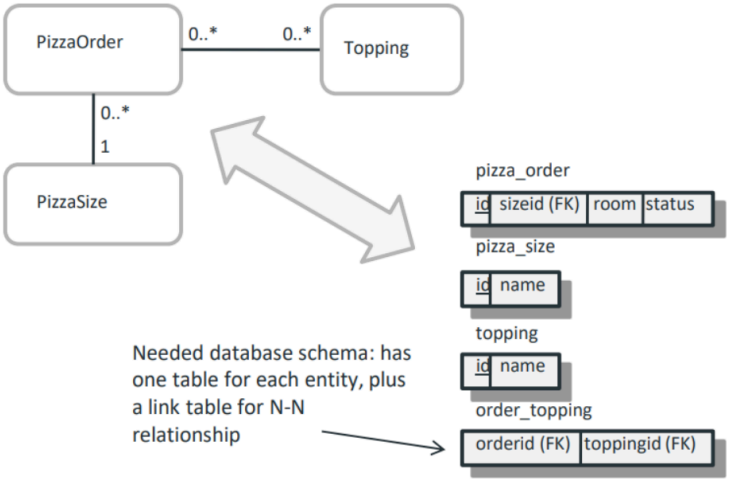


CLASS RELATIONSHIPS

UML class diagram/entity model: no big diamonds, type of relationship is infered from cardinality markings.



CLASS RELATIONSHIPS



INHERITENCE

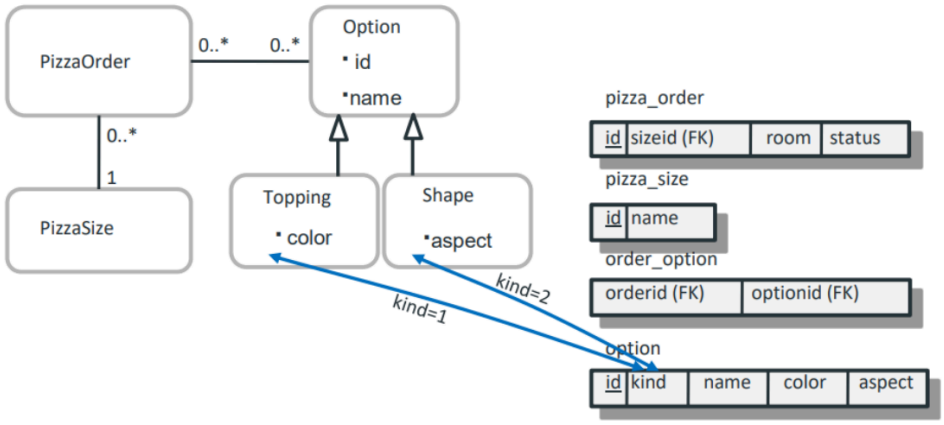
- E.g.:- generalize Topping to PizzaOption, to allow other options in the future.
  - Topping IS A PizzaOption
  - Shape IS A PizzaOption, ...
- Then a PizzaOrder can have a collection of PizzaOptions.
- We can process the PizzaOptions generically, but when necessary, be sensitive to their subtype: Topping or Shape
  - Inheritance is supported directly in Java, C#, etc., IS A “relationship”.
  - Inheritance is not native to RDBs, but part of EER, extended entity-relationship modeling, long-known schema-mapping problem.

INHERITENCE

- Hibernate can handle inheritance hierarchies and polymorphic associations to them.
- Hibernate provide single-table and multiple-tables per hierarchy solutions.
  - Single-table: columns for all subtypes, null values if not appropriate to row’s subtype.
  - Multiple-table: table for common (superclass) properties, table for each subclass for its specific properties, foreign key to top table.
  - Also hybrid: common table plus separate tables for some subclasses.

INHERITANCE MAPPING (SINGLE TABLE)

Discriminator column to specify subtype(not seen in object properties).



INHERITANCE MAPPING (SINGLE TABLE)

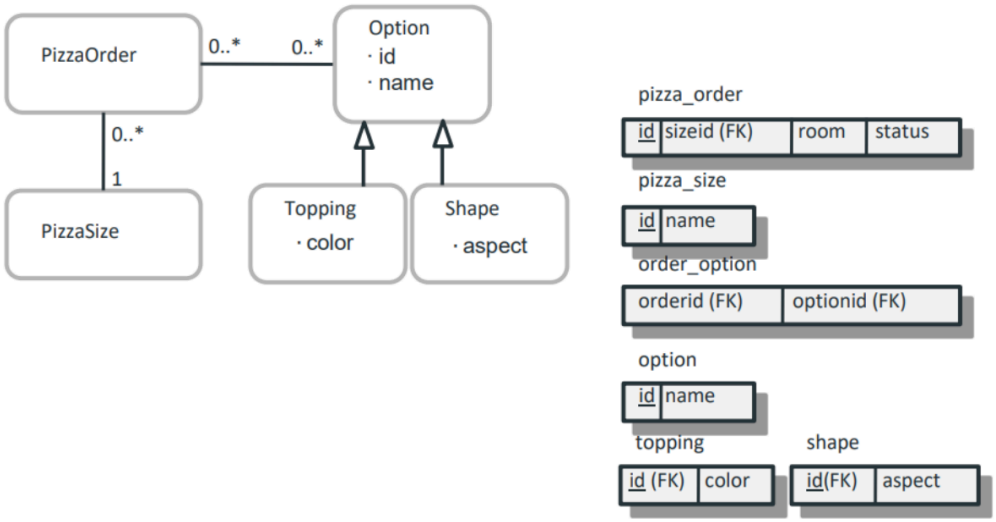
• The discriminator column (here “kind”) is handled by the O/R layer and does not show in the object properties.

• The hierarchy can have multiple levels.

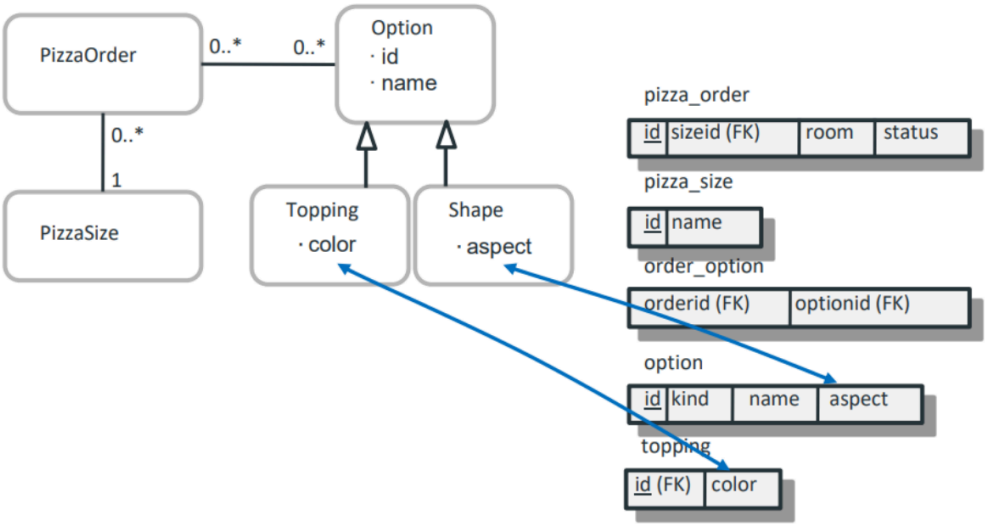
• Single-table approach is usually the best performing way.

• But we have to give up non-null DB constraints for subtype specific properties.

INHERITANCE MAPPING (MULTIPLE TABLES)



INHERITANCE MAPPING (HYBRID)



OBJECT AND OBJECT RELATIONAL TABLE

Relational Table

```
CREATE TABLE people (
name VARCHAR (30),
NIC Varchar (10) primary key,
phone VARCHAR (20) );
```

Object-relational Table

```
CREATE TYPE person AS OBJECT (
NIC VARCHAR(10),
name VARCHAR(30),
phone VARCHAR(20) );

CREATE TABLE person_table OF person(
NIC primary key);
```


OBJECT RELATIONAL TABLE

CREATE TABLE person_table OF person;

- You can view this table in two ways:
  - As a single-column table, in which each row is a person object, allowing you to perform object-oriented operations.
  - As a multi-column table, in which each attribute of the object type person such as idno, first_name, last_name, and so on, occupies a column, allowing you to perform relational operations.

OBJECT RELATIONAL TABLE

- **Method 1**
  - INSERT INTO person_table VALUES (“Sheela”, “123141”);
- **Method 2**
  - INSERT INTO person_table VALUES (person(“Sheela”, “123141”));

ACTIVITY 1

Consider the following schema:

employee (eno, ename, hireDate, salary)
project (projID, projName, budget)
emp_Proj (eno, projID, assignedDate)

1. Map the above schema to the tables using Object Relational Mapping.
2. Using the queries insert values to the created tables.

JPA & HIBERNATE

- **Java Persistence API**
  - A Java specification that gives some functionality and standard to ORM tools.
  - It is used to examine, control, and persist data between Java objects and relational databases.
  - JPA does not conduct any functioning by itself.
- **Hibernate**
  - A Java framework which is used to store the Java objects in the relational database system.
  - Open-source, lightweight, ORM tool.
  - Hibernate is an implementation of JPA. So, it follows the common standards provided by the JPA.

HIBERNATE QUERY LANGUAGE (HQL)

- It is a tool used in object relational mapping for Java environments.
- Hibernate supports many different relational databases.
  - Uses objects and their properties.
  - Keywords are not case sensitive but table, column names are case sensitive.
- Hibernate supports almost all the major RDBMS.
- Following is list of few of the database engines supported by Hibernate.
  - HSQL Database Engine, DB2, MySQL, PostgreSQL, FrontBase, Oracle, Microsoft SQL sever, Sybase SQL Server

HQL PROS AND CONS

- **Advantages**
  - Support Inheritance, associations, polymorphism
  - Generate primary keys automatically
  - Even the database changes HQL is independent
  - If we try to insert data to non existing table, HQL will create a table and insert values
- **Disadvantages**
  - Generate many SQL statements in run time
  - Same code need to be written in several files in the same application

ADVANTAGES OF HQL

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.

ADVANTAGES OF HQL

- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides Simple querying of data.