SQL Basics

Dr. Isaac Griffith Idaho State University

Outcomes

Idaho State University | Computer Science

After today's lecture you will be able to:

- Describe the basics of ORM
- Use ActiveJDBC in your daily development

ORM

CS 3321

What is an ORM?

Idaho State University | Computer Science

- Object-Relational Mapping (ORM) is a technique to map OO constructs such as classes to tables of a relational database

- ORM frameworks typically provide an API which allows for the augmenting of existing classes with the necessary capabilities to store and retrieve instances from the DB
  - Typically this is done using features of the underlying language
  - Usually implemented using the \textcolor{Roarange}{**Data Access Object (DAO) design pattern**

Advantages and Disadvantages

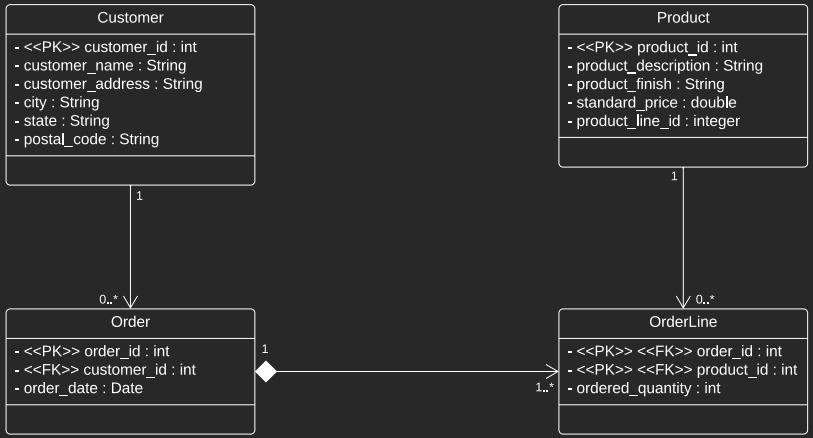
- ORM frameworks reduce the amount of code needed to perform the basic DB operations
- ORM frameworks also abstract away the details, making understanding the code more difficult (potentially)
- ORM frameworks are not magic and do not replace good OO design and good Database design
  - When using ORM your database will directly reflect your OO system
  - Garbage In == Garbage Out

ActiveJDBC

CS 3321

Our Model (again)

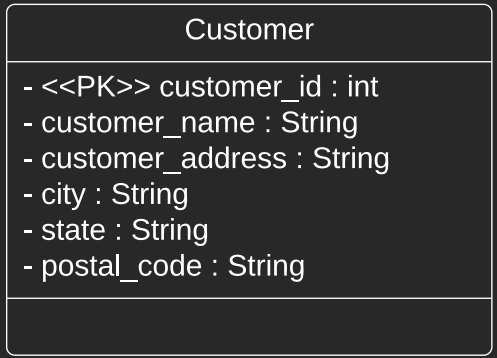
Idaho State University | Computer Science



Customer

Idaho State University | Computer Science

UML



SQL (MySQL)

```
CREATE TABLE customers (
  id           INTEGER NOT NULL PRIMARY KEY Auto_Increment,
  name         VARCHAR(25) NOT NULL,
  address      VARCHAR(30),
  city         VARCHAR(20),
  state        VARCHAR(2),
  postal_code  VARCHAR(9),
  created_at   DATETIME,
  updated_at   DATETIME
);
```

Java Code

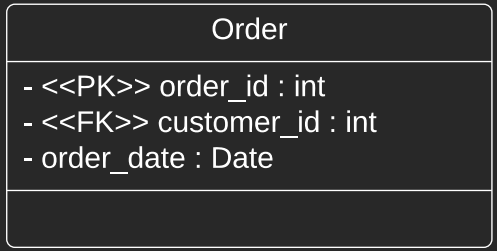
```
public class Customer extends Model {

}
```

Order

Idaho State University | Computer Science

UML



SQL (MySQL)

```
CREATE TABLE orders (
  id           INTEGER NOT NULL PRIMARY KEY Auto_Increment,
  order_date   DATETIME NOT NULL,
  customer_id  INTEGER REFERENCES customers (id),
  created_at   DATETIME,
  updated_at   DATETIME
);
```

Java Code

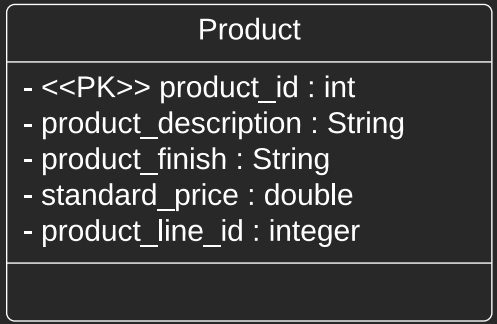
```
@BelongsTo(parent = Customer.class,
    foreignKeyName = "customer_id")
public class Order extends Model {

}
```

Product

Idaho State University | Computer Science

UML



SQL (MySQL)

```
CREATE TABLE products (
  id               INTEGER NOT NULL PRIMARY KEY Auto_Increment,
  description      VARCHAR(50),
  finish           VARCHAR(20),
  price            DECIMAL(6,2),
  product_line_id  INTEGER,
  created_at       DATETIME,
  updated_at       DATETIME
);
```

Java Code

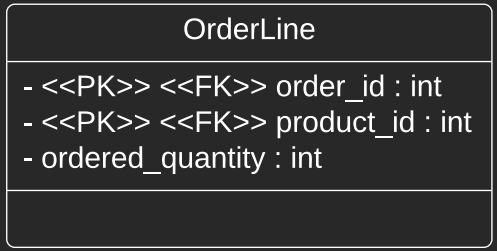
```
@Many2Many(other = Order.class, join = "order_lines",
    sourceFKName = "product_id", targetFKName = "order_id")
public class Product extends Model {

}
```

OrderLine

Idaho State University | Computer Science

UML



SQL (MySQL)

```
CREATE TABLE order_lines (
  id                INTEGER NOT NULL PRIMARY KEY Auto_Increment,
  order_id          INTEGER NOT NULL REFERENCES orders (id),
  product_id        INTEGER NOT NULL REFERENCES products (id),
          quantity  INTEGER,
  created_at        DATETIME,
  updated_at        DATETIME
);
```

Java Code

```
@Table("order_lines")
public class OrderLine extends Model {

}
```

Connections

Idaho State University | Computer Science

- The most basic connection is on a single thread and can be opened using the `DB` class:

```
new DB("default").open("com.mysql.cj.jdbc.Driver", // DB Connection Class
    "jdbc:mysql://localhost/test_db?serverTimezone=America/Denver", // DB URL
    "user", // User Name
    "password"); // password
```

  - MySQL Driver: `com.mysql.cj.jdbc.Driver`
  - SQLite Driver: `org.sqlite.JDBC`

- You need to have an open connection to do any real work with the model classes

- Once you are done you need to close the connection

```
new DB("default").close();
```

Creating Instances

Idaho State University | Computer Science

- You can create a new instance just like any other class, with the `new` operator.
  - You then need to provide the new instance with its data and save it.

```
Customer bob = new Customer();
bob.set("name", "Bob Sampson");
bob.set("address", "123 Some St.");
bob.set("city", "Pocatello");
bob.set("state", "ID");
bob.set("postal_code", "83201");
bob.saveIt();
```

- You can also do this in a single line with the `createIt` method:

```
Customer bob = Customer.createIt("name", "Bob Sampson", "address", "123 Some St.",
                        "city", "Pocatello", "state", "ID", "postal_code", "83201");
```

- Additionally, if you believe some record may exist, you could use the `findOrCreateIt` method in the same way as `createIt`.
  - This will attempt to find a match given the fields, and if one is found, will return that rather than create if, otherwise it creates it.

Accessing Data

Idaho State University | Computer Science

- You can access all of the data using a variety of `get` methods.
  - `get("fieldname")` returns an `Object` or throws an error if no such field exists
  - `getString(...)` just like get but for Strings
  - `getInteger(...)`
  - `getDate(...)`
  - and many more

- Additionally, there are `set` methods which take two parameters
  - First is the name of the field to set
  - Second is the new value of the field
  - **Remember to save after setting...**

Working with Relationships

Idaho State University | Computer Science

- If there is a relationship between two model objects, we can work with as follows
  - To connect two objects, we simply use the `add(...)` method to add a child to the parent
  - To retrieve a list of children from an object we simply use `getAll(ClassName.class)`
    - Where `ClassName.class` is the class object of the child class
- If we wish to know what the parent of a model class is we can use the
  - `getParent(ClassName.class)` which retrieve the parent instance for that class type
- We can also set the parent, though this is typically automatically done for us, using the
  - `setParent(...)` method

Deleting Instances

- Just as we can `save()` an instance to the database, we can also delete one as well.

- This is done by calling the `delete()` method on the object reference.
  - Only removes it from the database, not from memory
  - Places the object into a `frozen` state, which you can `unfreeze()` the object, and `saveIt` again.

Gradle

CS 3321

Connecting with Gradle

Idaho State University | Computer Science

- In order to include activejdbc into your gradle projects, you need to:
  1. Add the activejdbc-gradle-plugin:

```
plugins {
  ...
  id "de.schablinski.activejdbc-gradle-plugin" version "2.0.1"
}
```

  2. Add dependencies (if your model classes use them) to the `activejdbc` configuration

```
dependencies {
  ...
  activejdbc 'org.scala-lang:scala-library:2.12.6'
  ...
}
```

3. Create your database

Connecting with Gradle

Idaho State University | Computer Science

- In order to include activejdbc into your gradle projects, you need to:
  4. Setup your classes
  5. Connect your project to your database
  6. You can then instrument your classes as follows:

```
$ gradle instrumentModels
```

  7. Test your setup

- https://github.com/cschabl/activejdbc-gradle-plugin

Gradle Database Dependencies

Idaho State University | Computer Science

```
// SQLite
implementation 'org.xerial:sqlite-jdbc:3.25.2'

// MariaDborg.sqlite.JDBC
implementation 'org.mariadb.jdbc:mariadb-java-client:2.6.0'

// MySQL
implementation 'mysql:mysql-connector-java:8.0.20'

// JavaLite
implementation "org.javalite:activejdbc:3.0-j11"
implementation "org.javalite:javalite-common:3.0-j11"
```

For Next Time

- Review this Lecture
- Come to Class

Are there any questions?