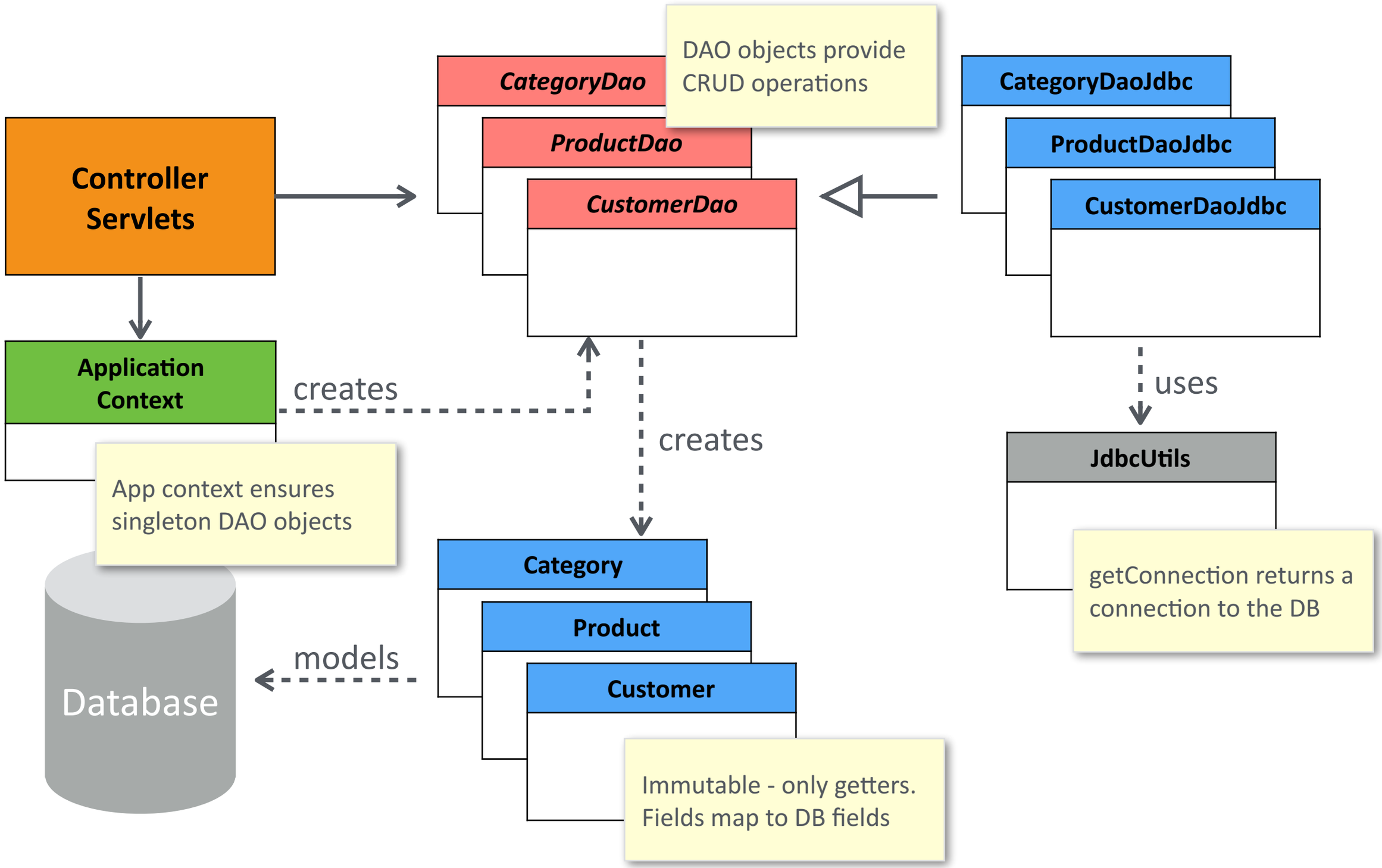
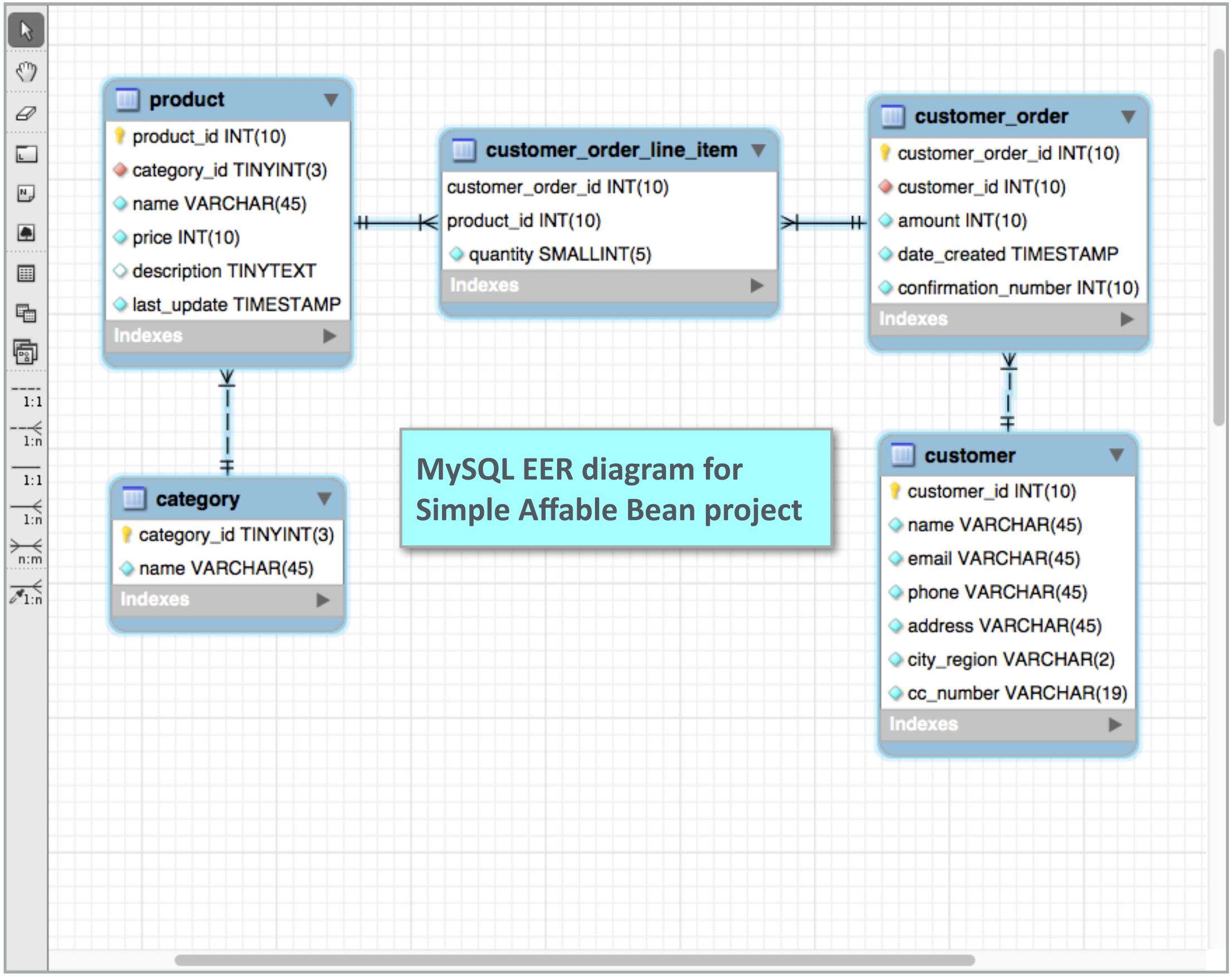
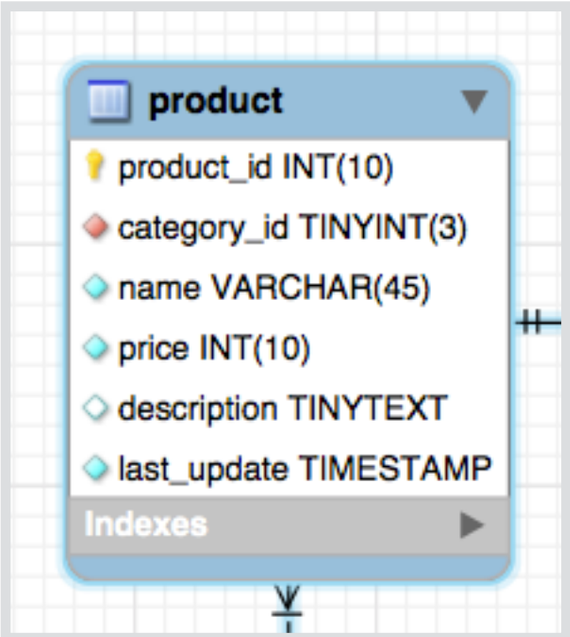
The DAO Pattern
&
Transactions

Data Access Object Pattern




product
product_id INT(10)
category_id TINYINT(3)
name VARCHAR(45)
price INT(10)
description TINYTEXT
last_update TIMESTAMP
Indexes
customer_order_line_item
customer_order_id INT(10)
product_id INT(10)
quantity SMALLINT(5)
Indexes
customer_order
customer_order_id INT(10)
customer_id INT(10)
amount INT(10)
date_created TIMESTAMP
confirmation_number INT(10)
Indexes
category
category_id TINYINT(3)
name VARCHAR(45)
Indexes
customer
customer_id INT(10)
name VARCHAR(45)
email VARCHAR(45)
phone VARCHAR(45)
address VARCHAR(45)
city_region VARCHAR(2)
cc_number VARCHAR(19)
Indexes
1:1
1:n
1:1
1:n
n:m
1:n
MySQL EER diagram for
Simple Affable Bean project

Model Class from a DB Table




```
public class Product {

    private long productId;
    private long categoryId;
    private String name;
    private int price;
    private String description;
    private Date lastUpdate;

    public Product(long productId,
                   long categoryId, ...) {
        this.productId = productId;
        this.categoryId = categoryId;
        ...

    public long getProductId() { return productId; }
    public long getCategoryId() { return categoryId; }
    public String getName() { return name; }
    public int getPrice() { return price; }
    public String getDescription() { return description; }
    public Date getLastUpdate() { return lastUpdate; }
}
```

DB fields become fields in model object

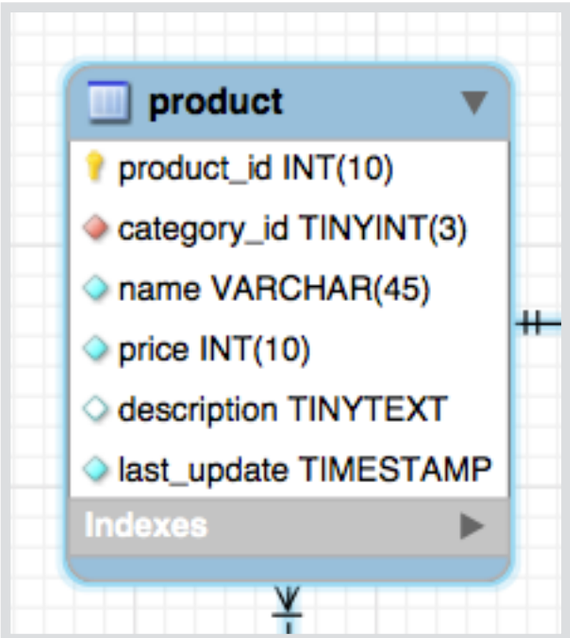
Constructor takes all fields

Getters for all fields

SQL to Java Type Mapping

SQL	Java
INT (primary or foreign key)	long
INT / SMALLINT	int
VARCHAR	String
TIMESTAMP / DATE	java.util.Date
BOOLEAN / TINYINT(1)	boolean

Constructing a DAO Interface




```
public interface ProductDao {

    public List<Product> findAll();
    public Product findByProductId(long productId);
    public List<Product> findByCategoryId(long categoryId);

}
```

A findAll method that returns a list of model objects

A findBy method that takes a primary key and returns a model object

A findBy method for each foreign key that takes a key and returns a list of model objects

Implementing a DAO interface using JDBC

```
public class ProductDaoJdbc {

    private static final String FIND_ALL_SQL =
        "SELECT product_id, category_id, name, price, last_update " +
        "FROM product";

    vate static final String FIND_BY_PRODUCT_ID_SQL =
        "SELECT product_id, category_id, name, price, last_update " +
        "FROM product WHERE product_id = ?";

    private static final String FIND_BY_CATEGORY_ID_SQL =
        "SELECT product_id, category_id, name, price, last_up
        "FROM product WHERE category_id = ?";

    public List<Product> findAll() { ... }
    public Product findByProductId(long productId) { ... }
    public List<Product> findByCategoryId(long categoryId) { ... }

    private Product readProduct(ResultSet resultSet) throws SQLException {
        Product result;
        long productId = resultSet.getLong("product_id");
        String name = resultSet.getString("name");
        int price = resultSet.getInt("price");
        Date lastUpdate = resultSet.getTimestamp("last_update");
        result = new Product(productId, name, price, lastUpdate);
        return result;
    }
}
```

Create constants for each SQL query

Question marks (?) represent parameters

readProduct returns the product in the current row of the result set

Note: the result set is effectively a table

The try-with-resources Statement

- The **try-with-resources** statement is a try statement that declares one or more resources
- A resource is an object that **must be closed** after the program is finished with it
- The try-with-resources statement **ensures that each resource is closed** at the end of the statement
- Any object that **implements java.lang.AutoCloseable** can be used as a resource

– The Java Tutorials

The findAll Method

```
@Override
public List<Product> findAll() {
    List<Product> result = new ArrayList<>();
    try (Connection connection = getConnection();
         PreparedStatement statement = connection.prepareStatement(FIND_ALL_SQL);
         ResultSet resultSet = statement.executeQuery()) {
        while (resultSet.next()) {
            Product p = readProduct(resultSet);
            result.add(p);
        }
    } catch (SQLException e) {
        throw new SimpleAffableQueryDbException(
                "Encountered problem finding all products", e);
    }
    return result;
}
```

This try-by-resources statement has three resources: connection, statement, and resultSet

The result set is essentially an iterator over the rows of the table returned by the SQL query

result is what the method is returning: a list of products

The findByProductId Method

```
@Override
public Product findByProductId(long productId) {
    Product result = null;
    try (Connection connection = getConnection();
         PreparedStatement statement = connection.prepareStatement(FIND_BY_PRODUCT_ID_SQL)) {
        statement.setLong(1, productId);
        try (ResultSet resultSet = statement.executeQuery()) {
            if (resultSet.next()) {
                result = readProduct(resultSet);
            }
        }
    } catch (SQLException e) {
        throw new SimpleAffableQueryDbException(
                "Encountered problem finding product by product i
    }
    return result;
}
```

This try-by-resources statement has two resources

The FIND_BY_PRODUCT_ID_SQL query string has one question mark (takes a parameter), so a setter method must be called to set it

Another try-with-resources statement is used for the resultSet

The if statement says if the result of the query contains something, it must be the desired product

The same catch block catches both try statements

The findByCategoryId Method

```
@Override
public Product findByCategoryId(long categoryId) {
    List<Product> result = new ArrayList<>();
    try (Connection connection = getConnection();
         PreparedStatement statement = connection.prepareStatement(FIND_BY_CATEGORY_ID_SQL)) {
        statement.setLong(1, categoryId);
        try (ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                Product p = readProduct(resultSet);
                result.add(p);
            }
        }
    } catch (SQLException e) {
        throw new SimpleAffableQueryDbException(
                "Encountered problem finding products by category id", e);
    }
    return result;
}
```

Finding products by the foreign key category id is similar to finding products by the product id, except that here a list of products is returned