***Inheritance* vs *Composition***

**What is Inheritance?**

Inheritance is the process where one class acquires the properties (methods and fields) of another class. Inheritance is accomplished in Java using the **extends** keyword. With the use of inheritance, the information is manageable in a hierarchical order. Inheritance creates an **“IS-A”** relationship.

**What is Composition?**

Composition is the process of creating a *whole out of the parts.* Composition is accomplished in Java by having an *instance* of another class as a field of your class, instead of extending the class. In this way, the class holds a reference to another class. Composition creates a **“HAS-A”** relationship and uses the annotations ***@OneToOne***, ***@OneToMany*** and ***@ManyToOne***.

**Example of Inheritance and Composition**

Car **has a** Engine and Car **is a** Automobile

In programming this is represented as:

class Engine {} // The Engine class.

class Automobile {} // Automobile class which is parent to Car class.

class Car extends Automobile { // Car is an Automobile, so Car class extends Automobile class.

private Engine engine; // Car has an Engine so, Car class has an instance of Engine class as its member.

}

**Demo #1 Inheritance – SINGLE TABLE Strategy**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Java Classes**  **@Inheritance (strategy=InheritanceType.SINGLE\_TABLE)**  **@DiscriminatorColumn(name=“PRODUCT\_TYPE”)**   |  | | --- | | **Product** | | id  name |   **@DiscriminatorValue(value=“B”) -- Book**  **@DiscriminatorValue(value=“S”) -- Software**   |  |  |  | | --- | --- | --- | | **Book** |  | **Software** | | isbn |  | version | | **Relational Database**   |  | | --- | | **PRODUCT** | | *PRODUCT\_ID* ***(PK)***  *PRODUCT\_TYPE*  PRODUCT\_NAME  VERSION <NULL when PRODUCT\_TYPE=B>  ISBN <NULL when PRODUCT\_TYPE=S> | |

**Demo #2 Inheritance – JOINED Table Strategy**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Java Classes**  **@Inheritance (strategy=InheritanceType.JOINED)**  **@DiscriminatorColumn(name=“PRODUCT\_TYPE”)**   |  | | --- | | **Product** | | id  name |   **@PrimaryKeyJoinColumn**  **(ReferenceColumnName=”PRODUCT\_ID”)**  **@DiscriminatorValue(value=“B”) -- Book**  **@DiscriminatorValue(value=“S”) -- Software**   |  |  |  | | --- | --- | --- | | **Book** |  | **Software** | | isbn |  | version | | **Relational Database**   |  | | --- | | **PRODUCT** | | *PRODUCT\_ID (PK)*  PRODUCT\_TYPE  PRODUCT\_NAME |  |  | | --- | | **BOOK** | | *PRODUCT\_ID (PK)*  ISBN |  |  | | --- | | **SOFTWARE** | | *PRODUCT\_ID (PK)*  VERSION | |

**Demo #3 Inheritance – TABLE PER CLASS Strategy**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Java Classes**  **@Inheritance (strategy=InheritanceType.TABLE\_PER\_CLASS)**  **@GernatedValue(strategy=GenerationType.TABLE)**   |  | | --- | | **Product** | | id  name |  |  |  |  | | --- | --- | --- | | **Book** |  | **Software** | | isbn |  | version | | **Relational Database**   |  | | --- | | **PRODUCT** | | *<empty table>* |  |  | | --- | | **BOOK** | | *PRODUCT\_ID (PK)*  PRODUCT\_NAME  ISBN |  |  | | --- | | **SOFTWARE** | | *PRODUCT\_ID (PK)*  PRODUCT\_NAME  VERSION | |

**Demo #4 Inheritance – MAPPED SUPERCLASS**

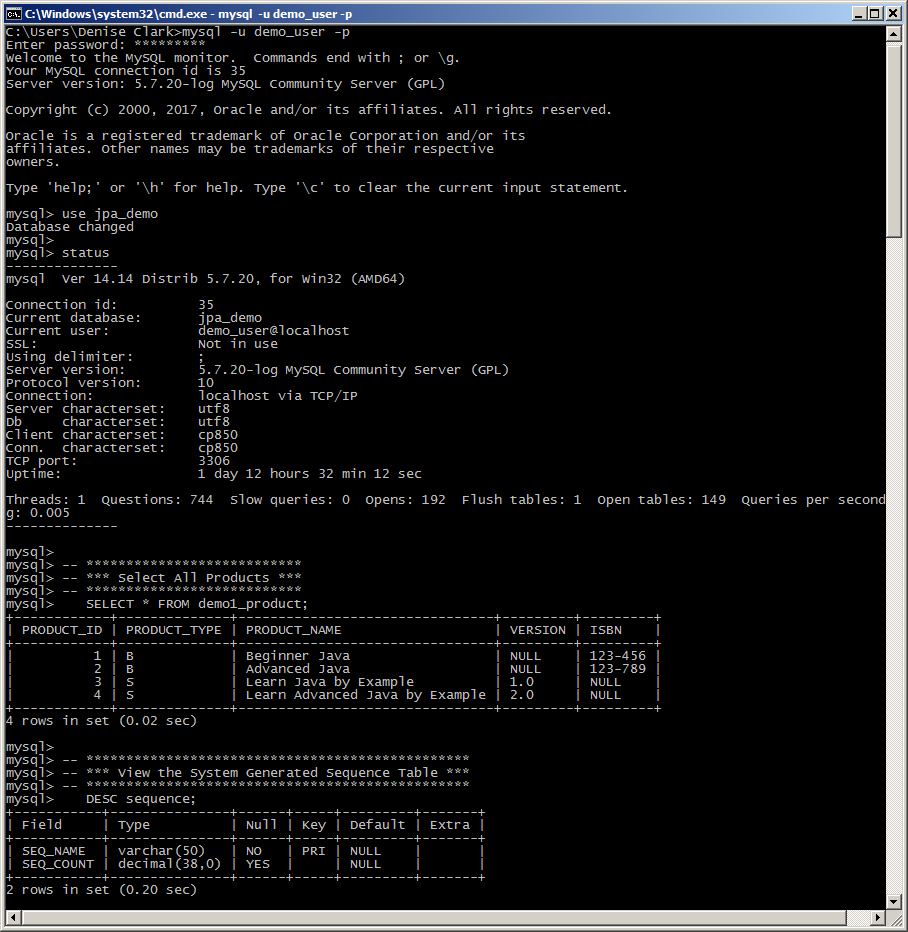
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Java Classes**  **@MappedSuperclass**   |  | | --- | | **Employee**  *<abstract Employee>* | | name  age |  |  |  |  | | --- | --- | --- | | **FTEmployee** |  | **PTEmployee** | | id  salary |  | id  hourlyRate | | **Relational Database**   |  | | --- | | **EMPLOYEE\_FT** | | *EMPOYEE\_ID (PK)*  NAME  AGE  SALARY |  |  | | --- | | **EMPLOYEE\_PT** | | *EMPOYEE\_ID (PK)*  NAME  AGE  SALARY | |

**Demo #5 Composition – *OneToOne* Relationship (1:1)**

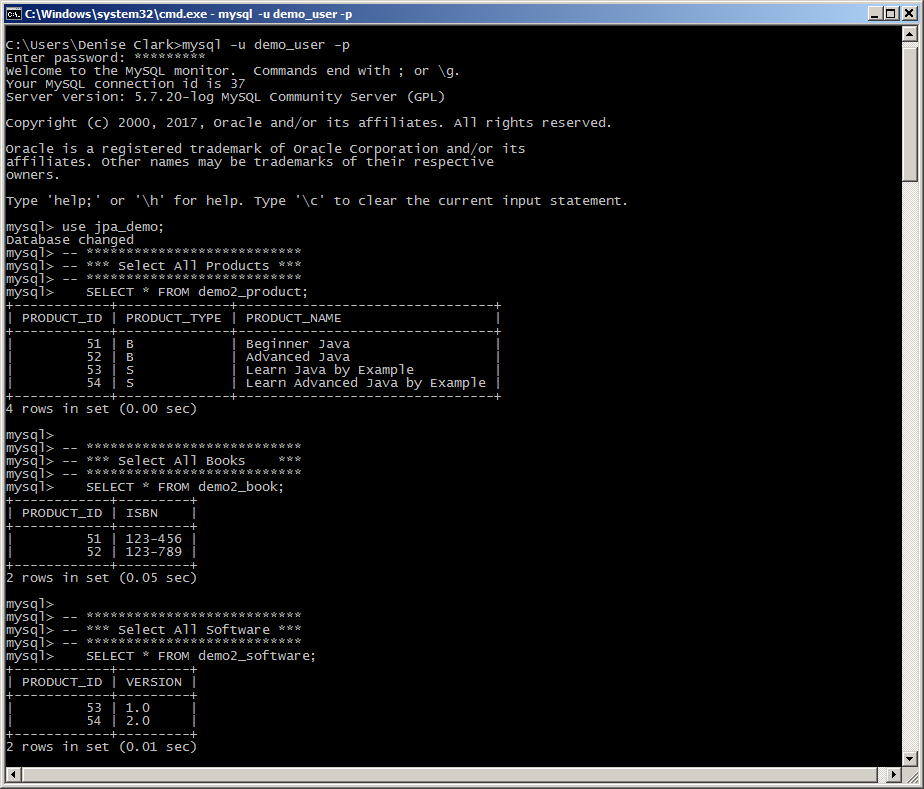
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Java Classes**   |  |  |  | | --- | --- | --- | | **Employee** |  | **Address** | | id  name  *@OneToOne*  *(mappedBy=”employee)*  Address ***(object)*** |  | id  address  city  state  zip  *@OneToOne*  Employee ***(object)*** |   *\*\*\* Model assumes that 1 Employee may have 1 Address and that 1 Address may belong to only 1 Employee* | **Relational Database**   |  | | --- | | **EMPLOYEE** | | *EMPLOYEE\_ID (PK)*  NAME |  |  | | --- | | **SOFTWARE** | | *ADDRESS\_ID (PK)*  ADDRESS  CITY  STATE\_CODE  ZIP\_CODE  ***EMPLOYEE\_EMPLOYEE\_ID*** *(FK)* | |

**Demo #6 Composition – *OneToMany* and *ManyToOne* Relationship (1:M)**

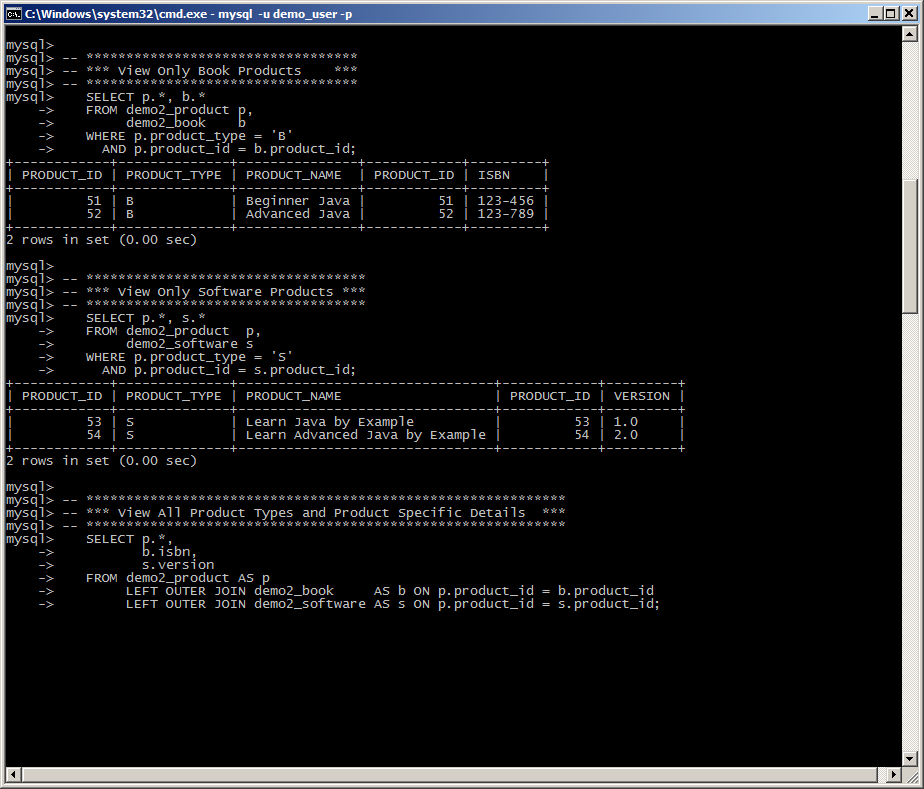
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Java Classes**   |  |  |  | | --- | --- | --- | | **University** |  | **Student** | | id  name  *@OneToMany*  *(mappedBy=”university)*  **List<Student>** students |  | id  firstName  lastName  *@ManyToOne*  University ***(object)*** |   *The University Class MUST have a “helper” method when a 1:M relationship is implemented:*  **public void addStudent(Student student) {**  **this.students.add(student);**  **}**  *\*\*\* Model assumes that 1 University may have 1 or more Students and that a Student may attend 1 University* | **Relational Database**   |  | | --- | | **UNIVERSITY** | | *UNIVERSITY\_ID (PK)*  NAME |  |  | | --- | | **STUDENT** | | *STUDENT\_ID (PK)*  FIRST\_NAME  LAST\_NAME  ***UNIVERSITY\_UNIVERSITY\_ID*** *(FK)* | |

**Data Structures for Demo #1** 

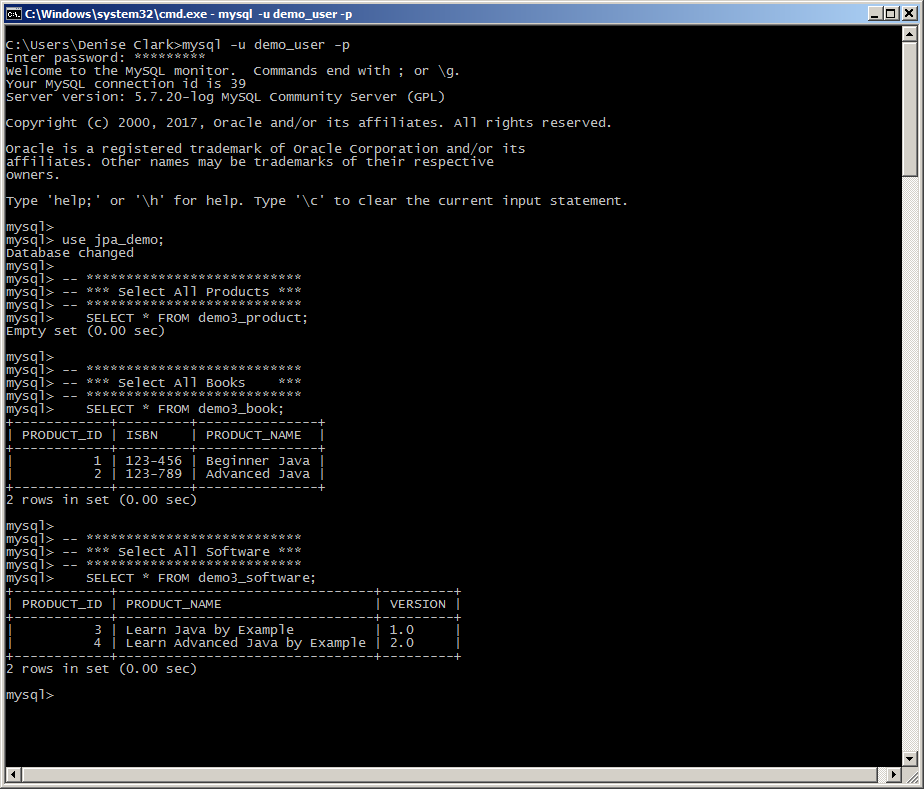
**Data Structures for Demo #2**



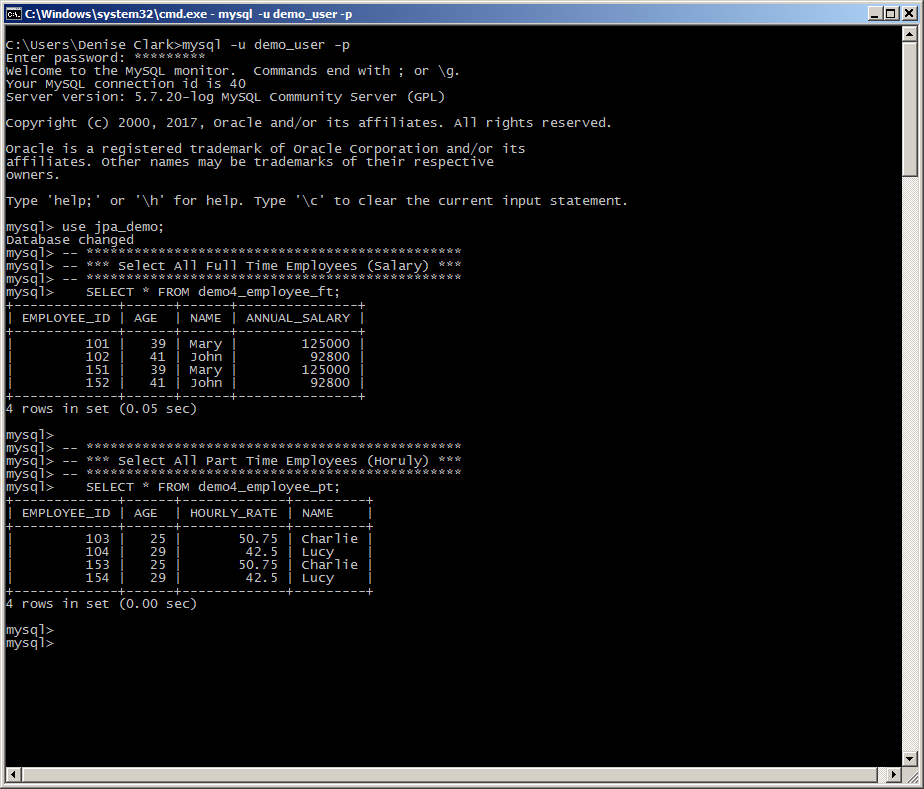
**Data Structures for Demo #2 (cont.)**



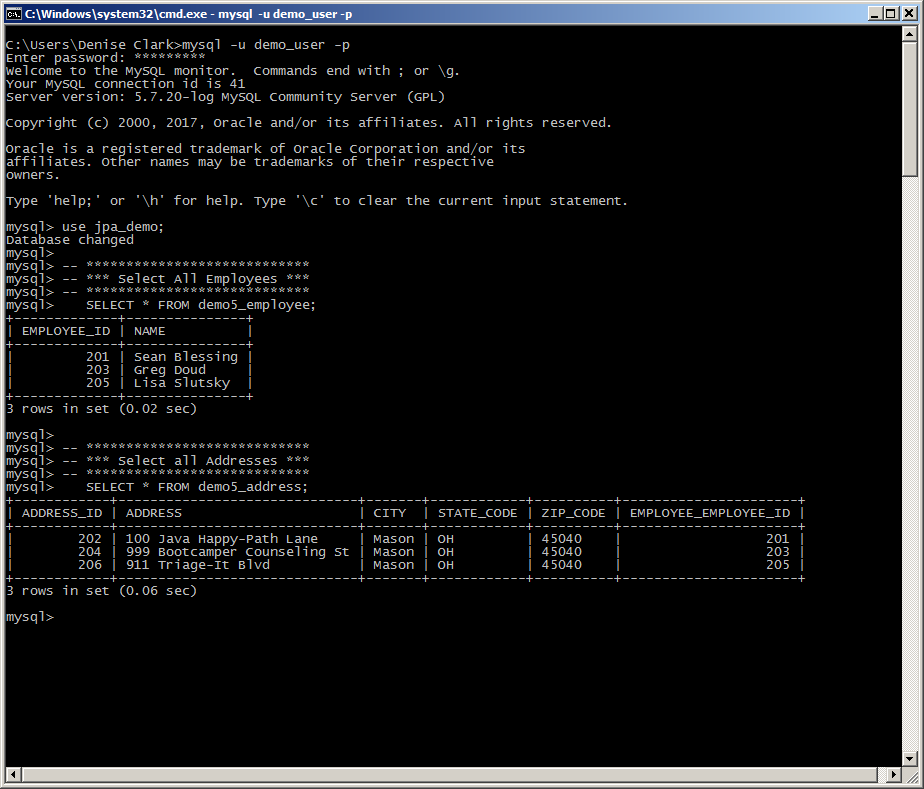
**Data Structures for Demo #3**



**Data Structures for Demo #4**



**Data Structures for Demo #5**



**Data Structures for Demo #6**

