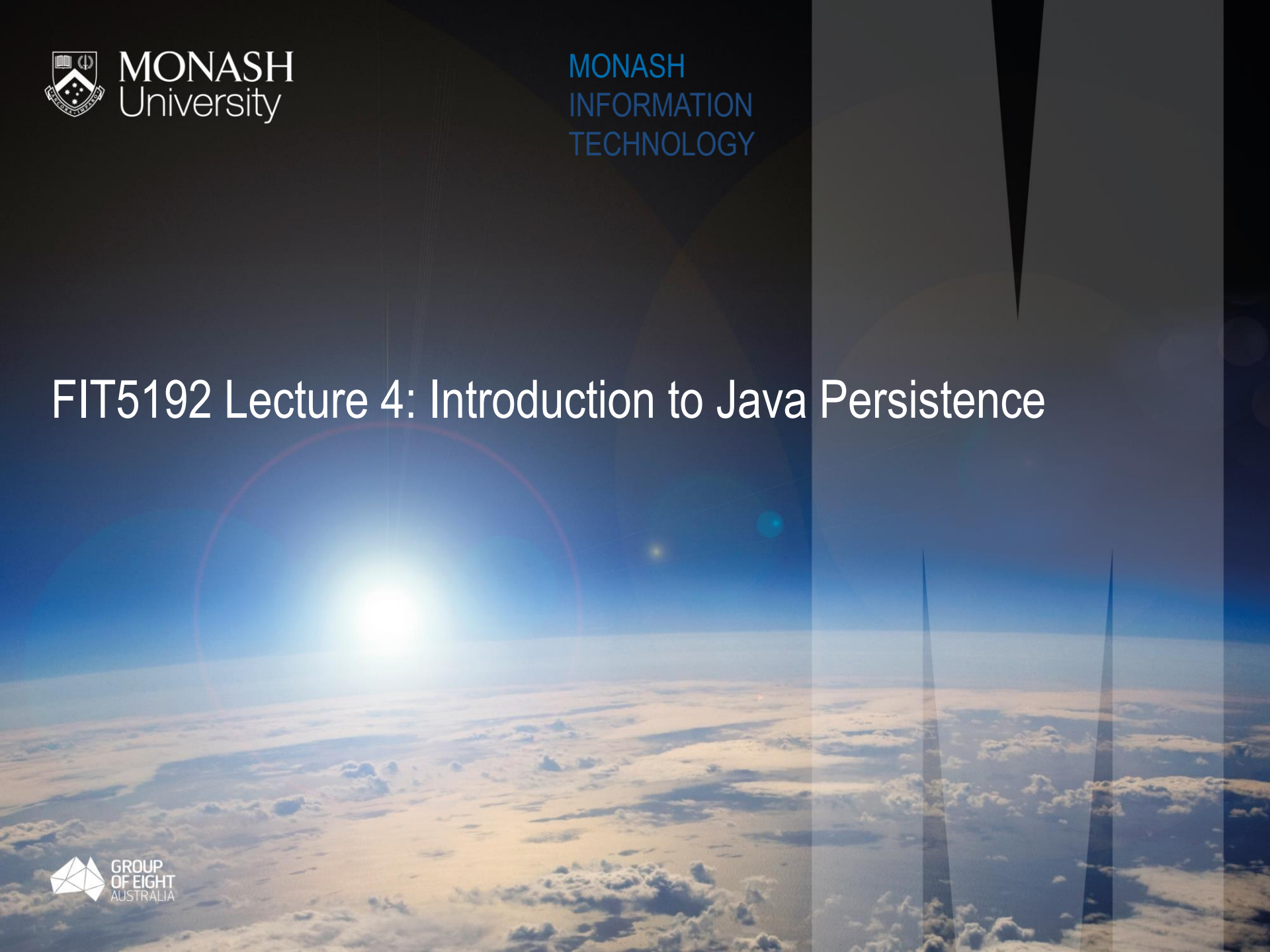

MONASH University
MONASH
INFORMATION
TECHNOLOGY
FIT5192 Lecture 4: Introduction to Java Persistence
GROUP
OF EIGHT
AUSTRALIA

MONASH University

- This Lecture
  - Quick revision of important database concepts
  - Data persistence in Java
  - Using JPA to communicate with a relational database (stand-alone application)

- Next Week in Lecture 7
  - More advanced topics on JPA
  - Mapping cardinality and inheritance
  - Using JPA in a container environment
  - Criteria API

GROUP OF EIGHT AUSTRALIA

Persistence

What is persistence?

- Persistence in relation to application development relates to data being stored permanently so it can be reused in the future.
    - Examples include storing data in files or in a database.

Persistence in Java

- Three main approaches:
  - Serialization
  - Java Database Connectivity (JDBC)
  - Object-relational mapping (ORM)

Serialization

- It converts an object into a sequence of bits.
- It stores data in a file
- It is easy to use.
- It must store and retrieve the entire object graph at once. Hence not suitable for large dataset.
- It cannot undo changes that are made to object if an error occurs while information is being updated.

Java Database Connectivity (JDBC)

- It uses SQL to retrieve/manage data in a database
- It does not support storing objects due to the relational paradigm used. Hence, mapping between objects and relational database is necessary.
- Developer must know both languages in order to use this framework.
- The mapping process is very time-consuming and error prone.

Object-Relational Mapping (ORM)

- Converts data stored as objects into a relational database structure and vice versa
- Allows developers to focus on object model instead of the mapping between object-oriented and relational paradigms.
- Supports advanced object-oriented concepts such as inheritance.
- It is not limited to Java.
- Many products available (e.g. Hibernate, EclipseLink, ActiveJDBC, MyBaits and etc.)

Java Persistence API (JPA)

- Provided as part of EJB 3.0 specification.
- A specification that provides standardized interface for accessing, persisting and managing data.
- It does not have any implementation itself.
- JPA providers (e.g. Hibernate, EclipseLink) develop their own version of JPA implementation that meets the requirements of the JPA specification.
- JPA can be used in EJBs, web components and Java SE application.

Databases

What is Database?

- A collection of data organized into a particular structure.
- Aims to make data easily accessed, managed and updated.
- Types of database design:
  - Hierarchical
  - Network
  - Relational
  - Object-Oriented

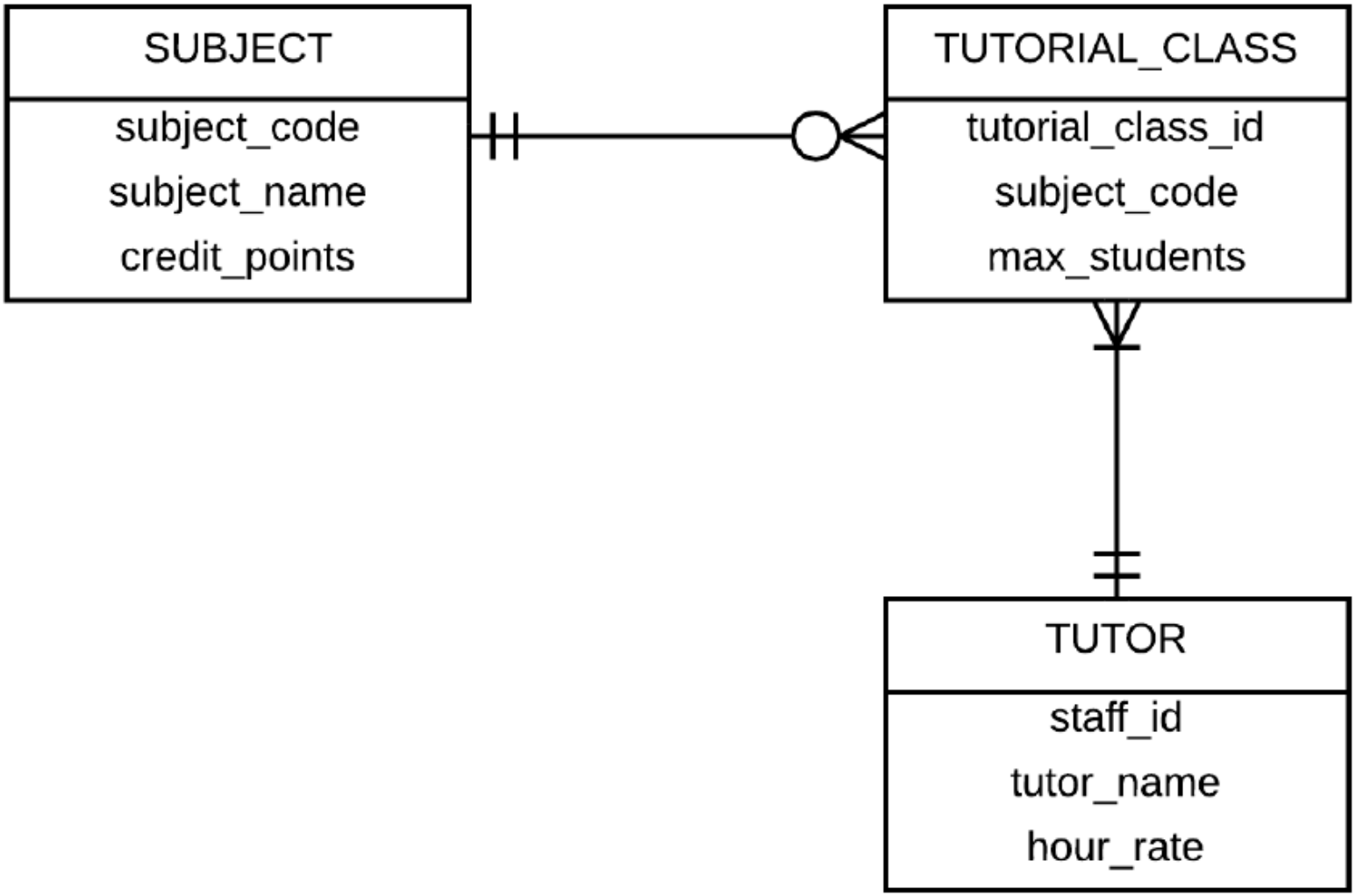
Relational Database Design (1)

- Relation
  - Commonly known as "table"
  - A collection of data related to a particular domain object
  - Organized in columns and rows
- Attribute / Field
  - Corresponds to a column in a table
  - A piece of information about the entity that will be tracked
- Record / Tuple
  - Corresponds to a row in a table
  - A set of data representing a single domain item
  - Each record represents a unique instance
  - Primary key is used to uniquely identify a record

Relational Database Design (2)

- Join
  - Records among tables can be linked together by primary keys & foreign keys
- Cardinality
  - Describes the number of records in one table can be related to that in another table
  - 3 degrees of cardinality:
    - one-to-one (1:1)
    - one-to-many (1:M)
    - many-to-many (M:M)
  - A relationship can be optional

Relational Database Example



Core Components of JPA

- Object-Relational Mapping (ORM)
- Entity Manager API
- Java Persistence Query Language (JPQL)
- Java Transaction API
- Callbacks & listeners

Entity

- A term to describe a domain object that is persisted in a database.
- It corresponds to a table in a relational database.
- An entity is presented as an entity class in a program.
- An instance of an entity corresponds to a row in that table.
- An entity instance is similar to a Plain Old Java Object (POJO).

Example of an Entity Class

```
@Entity
public class Student implements
Serializable {
   @Id
   private int id;
   private String name;
   private String educationLevel;
   public Student() {}
   //Getters & setters
}
```

Entity Class

An entity class must follow these requirements:

- It must be annotated with the `javax.persistence.Entity` annotation (i.e. @Entity)
- It must have a public / protected default constructor
- The class must not be `final`.
- It must implement `Serializable` if its objects is passed by value as a detached object.
- All attributes must be private, protected or package-private and they must be accessed only via accessors and business methods.

Mapping Metadata

- JPA maps objects to a database via metadata.
- It can be specified using:
  - Annotations
  - XML descriptors

Persistent Fields & Properties

- Information about the mapping between attributes of classes and tables is specified using annotations.
- The annotation can be applied to:
  – instance variables directly (persistent fields)
  – accessors (persistent properties)
- If persistent fields are used, JPA runtime access the instance variables directly
- If persistent properties are used, JPA runtimes access the instance variables via accessors and mutators, whose names must conform to JavaBean method conventions.
- Any attributes that are not marked as `@Transient` will be persisted to the database.

Configuration-by-Exception

- If not explicitly specified,
  - The entity name is mapped to a table name
  - Attribute names are mapped to column names
  - Primitive types are mapped to the relational types depending on the database used e.g. (`String` is mapped to `VARCHAR` in Derby and `VARCHAR2` in Oracle)

Basic Mapping Customization

- @Table
- @Id & @GeneratedValue
- @EmbeddedId & @IdClass
- @Column
- @Temporal
- @Transient
- etc.

Entity Manager

- The center piece of the API responsible for persisting, updating, retrieving and deleting entities to/from database.
- Serves as a bridge between an OO program and the relational world.
- It is only an interface and the implementation is provided by persistence provider (e.g. EclipseLink).

Obtaining an Entity Manager

The way to obtain an Entity Manager depends on the environment it is being used.

- Application-managed environment (e.g. J2SE application)
  - Via EntityManagerFactory
- Container-managed environment (e.g. Application Client, EJB, Web components)
  - Via Resource injection

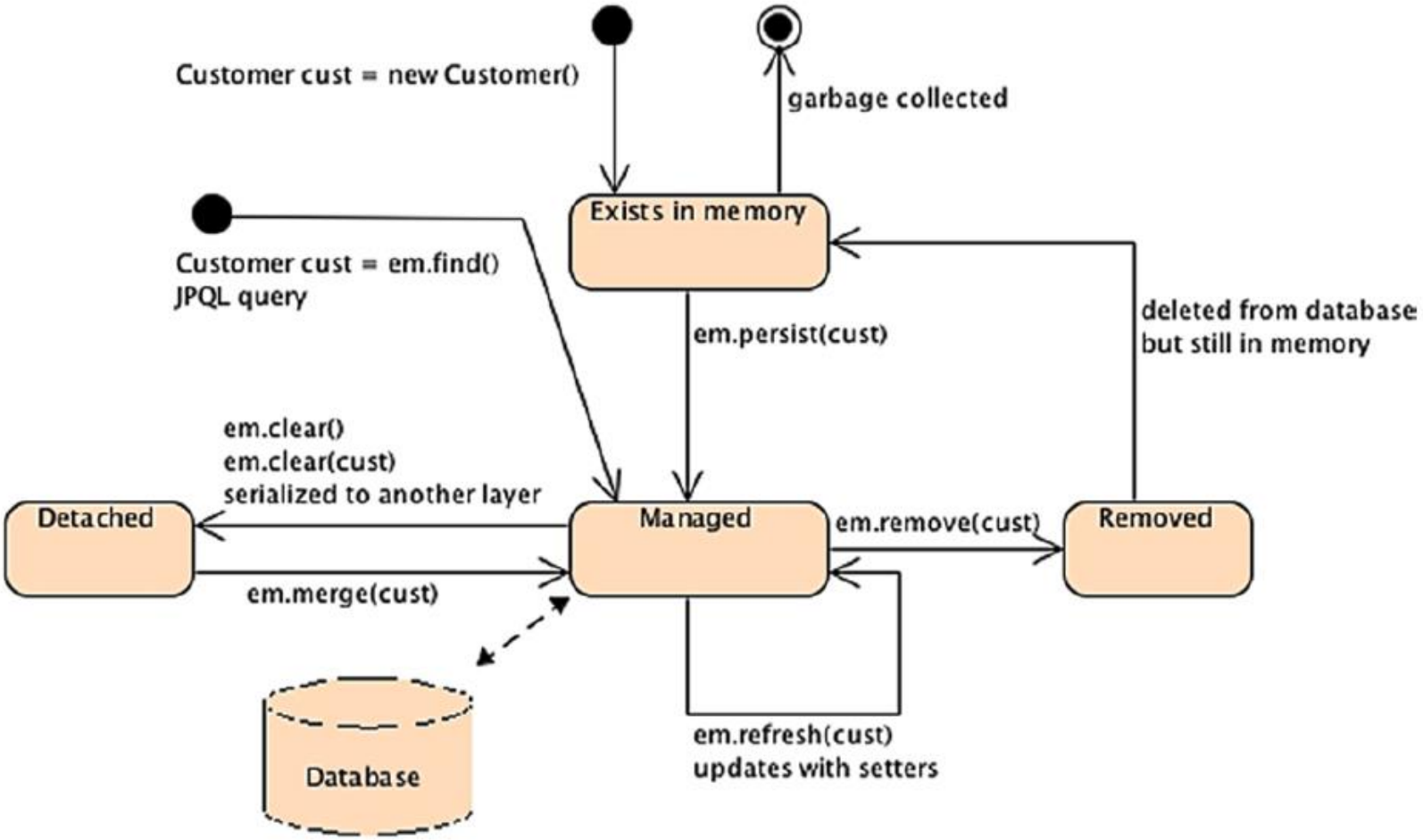
Application-Managed Entity Manager

- Developer is responsible for creating and closing the entity manager.
- Transaction management is handled by developer.

Entity Manager

- Main methods for managing entities include:
- `persist(Object)`
- `merge(Object)`
- `refresh(Object)`
- `remove(Object)`
- `clear()`

Entity Instance Life Cycle



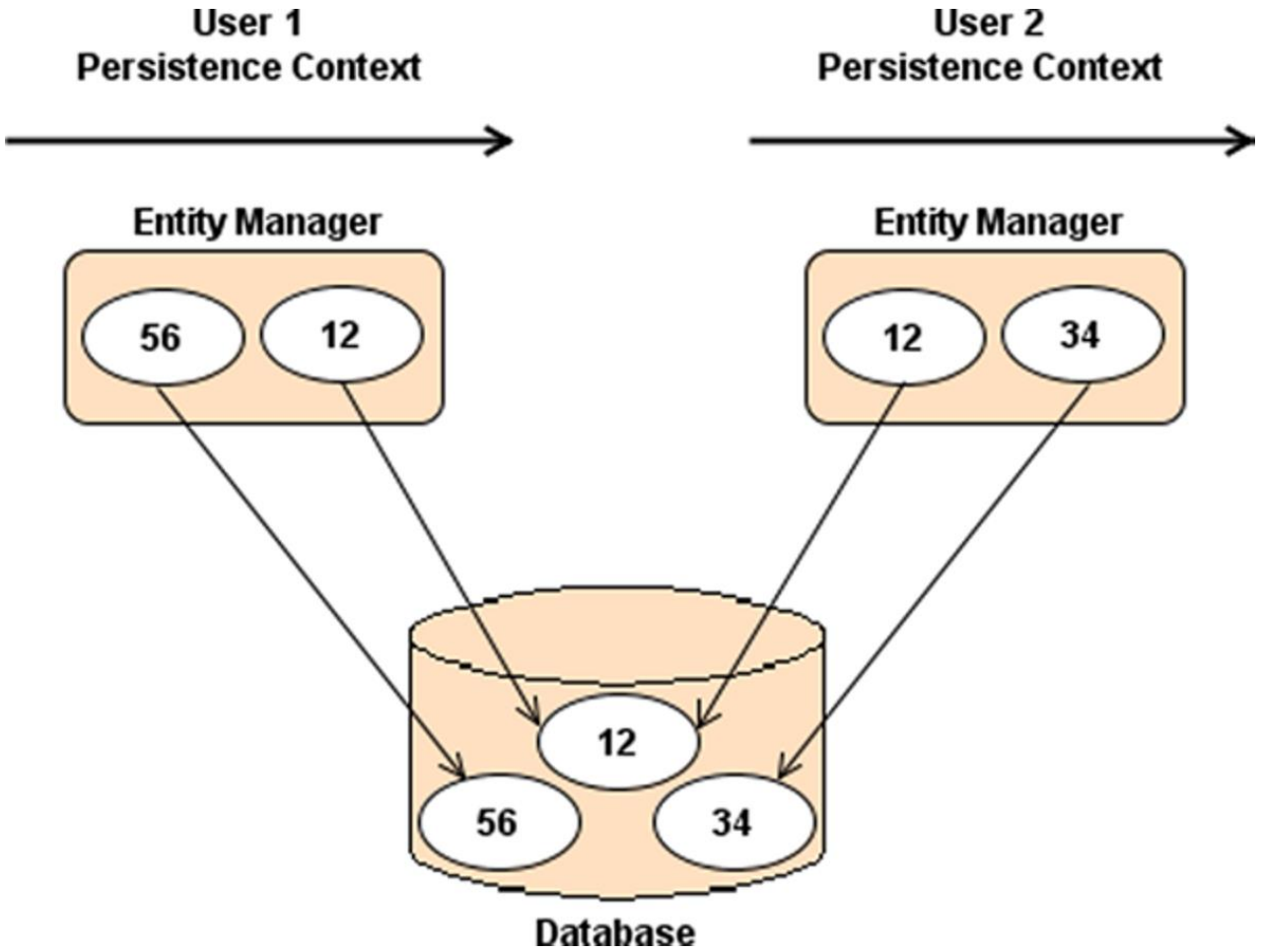
Persistence Unit

- Defined in `persistence.xml`
- Indicates the type of database to use
- Specifies the connection parameters

Persistence Context

- A set of managed entity instances at a given time.
- Created when one of the entity manager's method is invoked
- Entity Manager updates or consults the persistence context when one of its methods is called.
- Each instance must have a unique identity.
- Defines the scope under which particular entity instances are created, persisted, and removed.
- Can be seen as a first-level cache.

Entities living in different users' persistence context



Source: Figure 6-1, Beginning Java EE 7 p.181

Types of Persistence Context

- Transaction-scoped
  - Lifetime depends on a transaction.
  - Start when one of the entity manager's invoked.
  - Ends when a transaction ends (commits or rollback).
  - All managed entities are detached.
- Extended
  - Entities remained managed across transactions.
  - Entities are only detached when `remove()` is called.

Transaction Management

- Ensures data integrity by:
  - restricting update of data by multiple programs/thread simultaneously
  - allowing data to be restored to the original state when a failure happens in the middle of an operation
- When using application-managed entity manager, transaction management is manually handled by developers
  - Explicitly create a transaction context by creating an `EntityTransaction`
  - Explicitly commit and rollback of a transaction when finishes

Releasing Resources

- Release `EntityManager` and `EntityManagerFactory` by calling the `close()` when the program finishes.

Java Persistence Query Language (JPQL)

- An object-oriented language for querying data
- Root in SQL but designed with object-oriented concepts in mind
- Use dot (.) notation
- Manages objects and attributes, not tables and columns

Examples of JPQL Query

```
SELECT s FROM Student s;

SELECT s.parent.firstName FROM Student s;

SELECT s FROM Student s WHERE s.age > 10;

SELECT COUNT(s) FROM Student s;

SELECT s FROM Student s WHERE s.age > ?1;

SELECT S FROM Student s WHERE s.age > :sage;
```

Java Persistence Query Language (JPQL) Functions

- Named Query – `createNamedQuery(String queryName)`
- Dynamic Query – `createQuery(String query)`
- SQL Query- `createNativeyQuery(String sqlQuery)`

Summary

- This Lecture
  – Quick revision of important database concepts
  – Data persistence in Java
  – Using JPA to communicate with a relational database (stand-alone application)
- Lecture 5 and 6: Java Server Faces
- Lecture 7
  – More advanced topics on JPA

MONASH University

See you in the Studio !

Readings

- ***Recommended reading*** - Chapter 4: Java Persistence API and Chapter 5: Object Relational Mapping in Beginning Java EE 7, Antonio Goncalves