CmpE 473
Internet Programming

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Java Persistence API

---Mostly from Java EE Tutorial

Persistence

- Synchronization with DB is difficult
- Everytime an update comes execute an update on the DB
- Alternatively,
  - Represent relational data in Java app
  - Reflect changes on data automatically
  - Change in app data *persists* in DB
  - Data lives longer than the app

Persistence Mechanisms

Supports:	Serialization	JDBC	ORM	ODB	EJB 2	JDO	JPA
Java Objects	**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Advanced OO Concepts	**Yes**	No	**Yes**	**Yes**	No	**Yes**	**Yes**
Transactional Integrity	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Concurrency	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Large Data Sets	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Existing Schema	No	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**
Relational and Non-Relational Stores	No	No	No	No	**Yes**	**Yes**	No
Queries	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Strict Standards / Portability	**Yes**	No	No	No	**Yes**	**Yes**	**Yes**
Simplicity	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**

From OpenJPA

Entity

- Used to be an EJB
- Persistence domain object
  - Corresponds to a table in DB
  - Each instance is a row in the table
- Specify explicitly which properties of entities will persist
- Those properties will be synchoronized with DB

Entity Requirements

- Must be annotated with the javax.persistence.Entity
- Must have a public or protected, no-argument constructor
- Must not be declared final.
- Must implement the Serializable interface if its instance is going to be passed by value to a different object
- May extend both entity and non-entity classes, and non-entity classes may extend entity classes.
- Can't have public persistent instance variables (clients must access the entity's state through accessor or business methods)

Enabling Persistence

- Persistent fields
  - Annotations applied to instance variables
  - Fields are persistent unless marked with `javax.persistence.Transient` or `transient`
- Persistent properties
  - Annotations applied to gettter methods
  - `getColor()` for instance variable color; `isSunny()` for boolean variables

Design

- Primary Key
  - Every entity must have a single or composite primary key
  - Primary keys must be annotated (simple primary key annotated with `javax.persistence.Id`)

- Relationships
  - One-to-one, one-to-many, many-to-one,many-to-many
  - Possibly directional
  - Inheritance

Entity Management

- Entity manager with a *persistence context*
  - Persistent identity-->Unique entity instance
- Container-managed
  - Container propagates the persistence context to all entity managers
  - Java EE manages the lifecycle
- Application-managed
  - Create a new, customized entity manager
  - Lifecycle managed by the application

Entity Operations

- Persist
  - Instances are being managed
  - When persist completes, commit
- Remove
  - Remove from the managed entity list
- Flush
  - Force synchronization to the data store
- Create query
  - Query the data store with dynamic queries that use application data
  - Annotation of static queries

Persistence Units

- A set of entity classes that are managed by an Entity Manager
- Persistence.xml

```
<persistence>
    <persistence-unit name="OrderManagement">
        <description>This unit manages orders and customers.
            It does not rely on any vendor-specific features
 and can
            therefore be deployed to any persistence provider.
        </description>
        <jta-data-source>jdbc/MyOrderDB</jta-data-source>
        <jar-file>MyOrderApp.jar</jar-file>
        <class>com.widgets.Order</class>
        <class>com.widgets.Customer</class>
    </persistence-unit>
</persistence>
```

JPA Query

- Queries for entities and their persistent state
- Enable portable queries independent of the data store
- Uses the abstract schema and path expressions
- Resembles SQL queries

Query Examples

- SELECT

QL_statement ::= select_clause from_clause
[where_clause][groupby_clause][having_clause][orderby_clause]

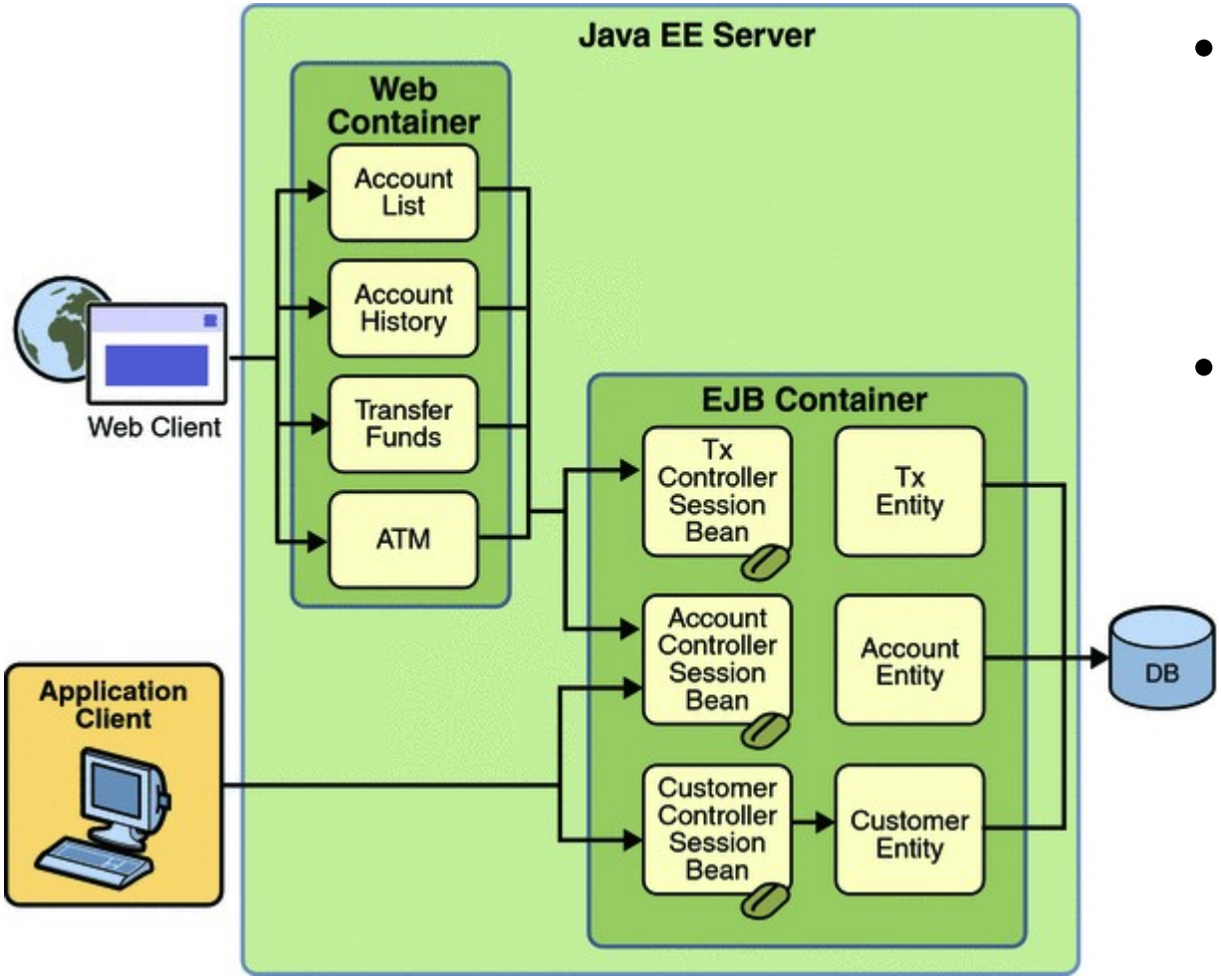
```
SELECT p
FROM Player p
```

- UPDATE

update_statement :: = update_clause [where_clause] delete_statement :: =
delete_clause [where_clause]

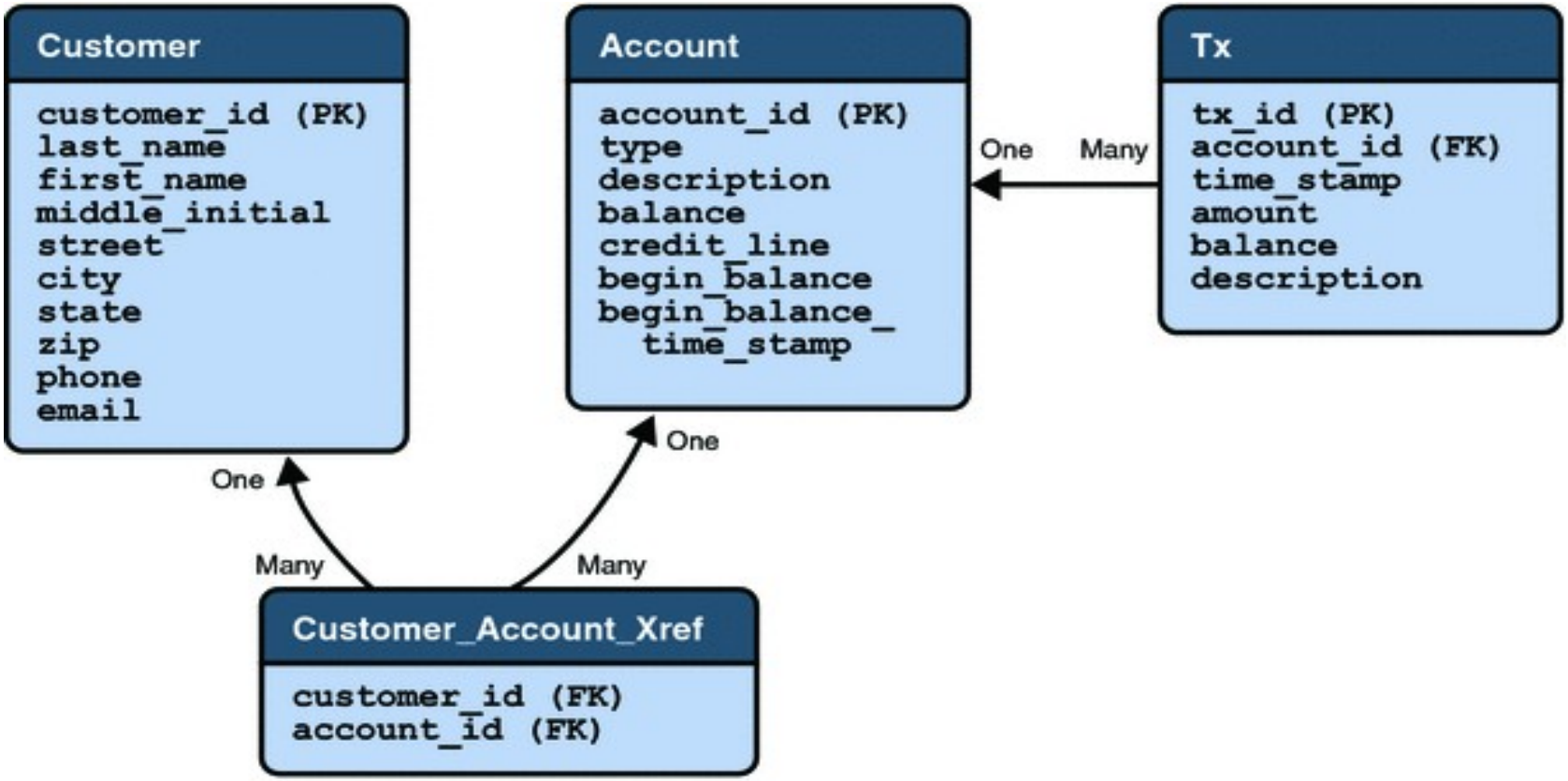
```
UPDATE Player p
SET p.status = 'inactive'
WHERE p.lastPlayed < :inactiveThresholdDate
```

Duke's Bank Example (1)



- Web client for the users to manage accounts
- Application client for the administrators

Duke's Bank Example (2)



Duke's Bank Example (3)

- Three entities that match to JPA entities
- Account
    - Keep track of account information
    - Corresponds to bank_account
    - Each row is an instance
- Customer
    - Keep track of customer information
    - Corresponds to bank_customer
- Tx
    - Keep track of transaction information
    - Corresponds to bank_tx

Duke's Bank Example (4)

```
@Entity
@Table(name = "BANK_ACCOUNT")
@NamedQueries({
    @NamedQuery(name = "Account.FindById",query = "SELECT a FROM Account WHERE a.id
      = :id")
    , @NamedQuery(name = "Account.FindByType", query = "SELECT a FROM Account a
      WHERE a.type = :type")
    , @NamedQuery(name = "Account.FindByDescription", query = "SELECT a FROM Account a
      WHERE a.description = :description")
    , @NamedQuery(name = "Account.FindByBalance", query = "SELECT a FROM Account a
      WHERE a.balance = :balance")
    , @NamedQuery(name = "Account.FindByCreditLine", query = "SELECT a FROM Account a
      WHERE a.creditLine = :creditLine")
    , @NamedQuery(name = "Account.FindByBeginBalance", query = "SELECT a FROM Account
      a WHERE a.beginBalance = :beginBalance")
    , @NamedQuery(name = "Account.FindByBeginBalanceTimeStamp", query = "SELECT a
      FROM Account a WHERE a.beginBalanceTimeStamp = :beginBalanceTimeStamp")
})
```

Duke's Bank Example (5)

```
public class Account implements java.io.Serializable {
    @Column(name = "BALANCE")
    private BigDecimal balance;
    @Column(name = "BEGIN_BALANCE")
    private BigDecimal beginBalance;
    @Column(name = "CREDIT_LINE")
    private BigDecimal creditLine;
    @JoinTable(name = "BANK_CUSTOMER_ACCOUNT_XREF",
     joinColumns = @JoinColumn(name = "ACCOUNT_ID",
     referencedColumnName = "ACCOUNT_ID")
    , inverseJoinColumns = @JoinColumn(name = "CUSTOMER_ID",
     referencedColumnName = "CUSTOMER_ID")
    )
      @ManyToMany
      private Collection<Customer> customers;
```

Duke's Bank Example (6)

```
@Column(name "BEGIN_BALANCE_TIME_STAMP")
@Temporal(TemporalType.TIMESTAMP)
private Date beginBalanceTimeStamp;
@TableGenerator(name = "accountIdGen", table = "BANK_SEQUENCE_GENERATOR",
  pkColumnName = "GEN_KEY", valueColumnName = "GEN_VALUE", pkColumnValue =
  "ACCOUNT_ID", initialValue = 5050, allocationSize = 1)
@Id
@GeneratedValue(strategy = GenerationType.TABLE, generator = "accountIdGen")
@Column(name = "ACCOUNT_ID", nullable = false)
private Long id;
@Column(name = "DESCRIPTION")
private String description;
@Column(name = "TYPE")
private String type;
```

Duke's Bank Example (6)

- Three session beans for client's view of business logic:
- AccountControllerBean
  - Create and remove entities (by calling the create and remove methods of Account entity)
  - Account-Customer relationship (by calling addCustomerToAccount)
- CustomerControllerBean
  - To manage Customer entitiy
- TxControllerBean.
  - To handle bank transactions

Beans

- Three session beans for client's view of business logic:
- AccountControllerBean
    - Create and remove entities (by calling the create and remove methods of Account entity)
    - Account-Customer relationship (by calling addCustomerToAccount)
- CustomerControllerBean
    - To manage Customer entitiy
- TxControllerBean.
    - To handle bank transactions