ORM With Hibernate

Objectives

www.infogain.com

- At the end of the course participants will be able to
  - Define Hibernate.
  - Explore what hibernate simplify
  - Define Need for Hibernate
  - Recognize the concept of ORM
  - Define Hibernate Architecture.
  - Describe State management of Objects
  - Define Mapping between Java Objects and Relational data (O/R Mapping)
  - Inheritance Mapping

infogain

Introduction to Hibernate

What is Hibernate?

- It is an ORM tool that simplifies the data creation, data manipulation and data access.
- Developed in 2001 by Gavin King.
- It is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables.
- It's purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

What Does Hibernate Simplify?

- Saving and retrieving your domain objects
- Perform Complex joins for retrieving related items
- Schema creation from object model

Hibernate……why?

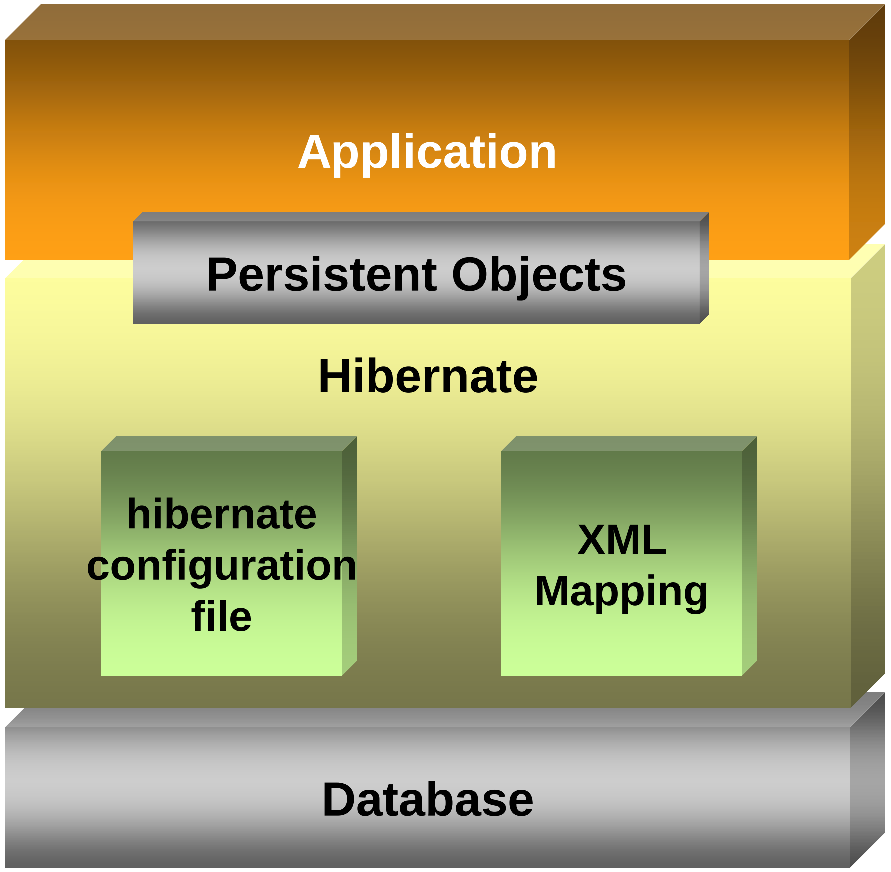
- What Relational Database systems do badly ?
  - Does not support polymorphism
  - Does not make use of real world objects
  - Extra code required to map the Java objects to database objects

Hibernate……why?

www.infogain.com

- <u>Problems with JDBC</u>
  - With JDBC, developer has to write Database dependent code.
  - JDBC supports only native Structured Query Language (SQL).
  - With JDBC, caching is maintained by hand-coding.
  - In JDBC there is no check that always every user has updated data. This check has to be added by the developer.

Hibernate Architecture



High Level view of Hibernate Architecture

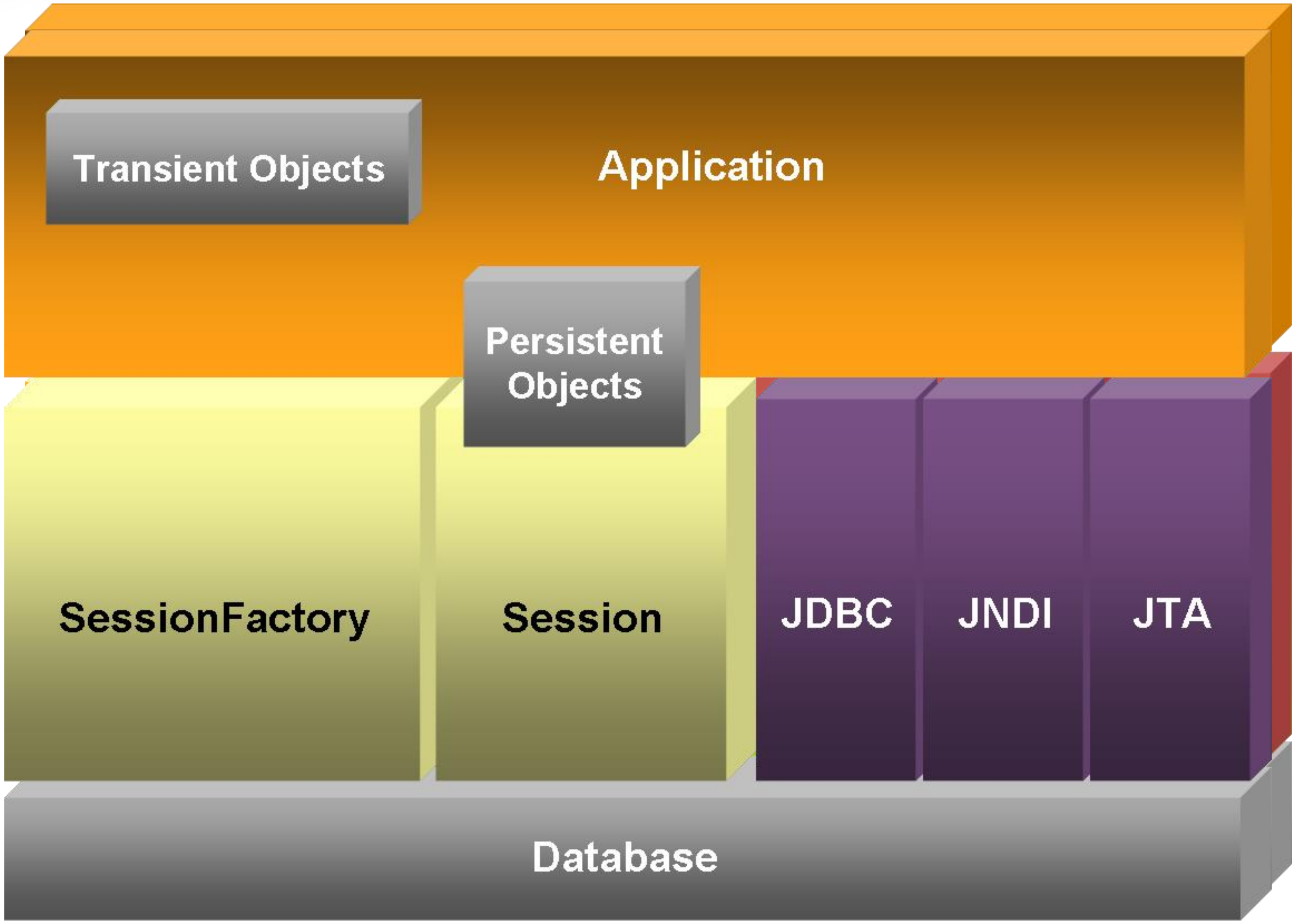
Features of Hibernate

- Retains natural object model
- Minimizes Code
- Does not require a container
- Remove 95% of common data persistence problems
- Object-oriented query language
- Object / Relational mappings
- Automatic primary key generation

Hibernate Architecture and Configuration

Architecture

www.infogain.com



The Core Interfaces and Classes

- The core interfaces and classes which are used in every Hibernate application.
  - Configuration class
  - Session Factory
  - Session
  - Transaction interface
  - Query and Criteria interfaces

Configuration class

- The Configuration object is used to configure and bootstrap Hibernate.
- The application uses a Configuration instance to specify the location of mapping documents and Hibernate-specific properties and then create the Session Factory.
- it’s the first object you’ll meet when you begin using Hibernate.

Hibernate Terminology (defined)

- Session Factory (org.hibernate.SessionFactory)
    - A factory for Session objects
    - Might hold an optional (second-level) cache of data that is reusable between transactions.
    - It has an application scope,used throughout the application life span.

Hibernate Terminology (defined)

- Session (org.hibernate.Session)

    - A single-threaded, short-lived object
    - Transactional scope , it is closed after completing the operation
    - Represents a conversation between the application and the persistent store.
    - Wraps a JDBC connection.
    - Factory for Transaction.
    - Holds a mandatory (first-level) cache of persistent objects.
    - Used when looking up objects by identifier.

Transaction Interface

- The Transaction interface is an optional API
- If any one step fails, then the whole unit of work fails.
- When we consider database, Transaction groups a set of statements/commands which gets committed together.
- If a single statement fails, whole work will be rolled back.

Hibernate Mapping And Configuration

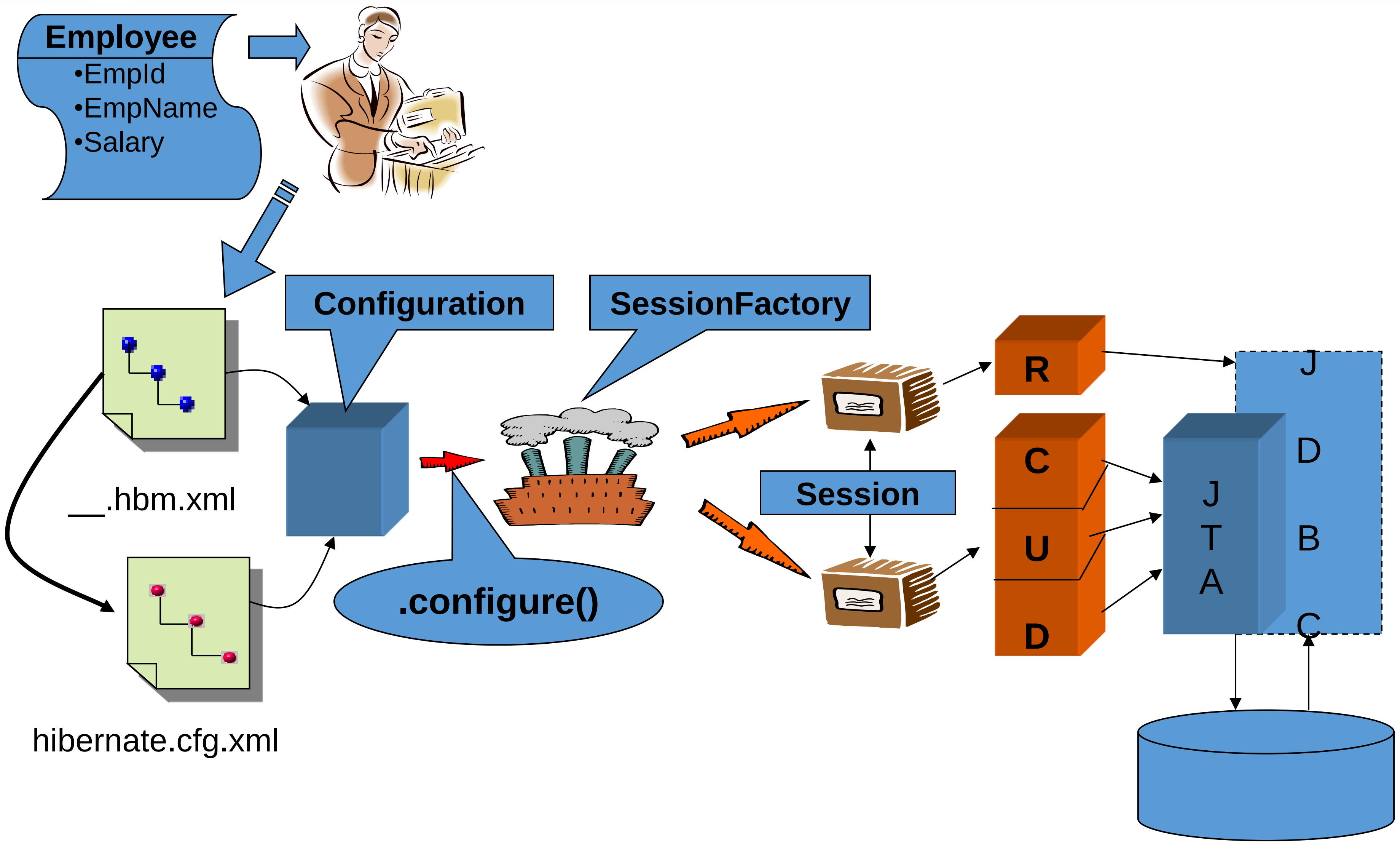
www.infogain.com

- **Mapping**
  - The mapping file contains mapping from a pojo class name to a table name and pojo class variable names to table column names.
  - Mapping can be given to an ORM tool either in the form of an XML or in the form of annotations.
  - While writing an hibernate application, we can construct one or more mapping files, mean a hibernate application can contain any number of mapping files.

- **Mapping can be done using 2 ways**
  - XML
  - Annotations.(since JDK1.5)

Hibernate Architecture

www.infogain.com



Steps to Implementation.

- Identify the DB columns.
- Create a domain class consists of the DB columns as attributes of class.
- Create getters and setters of those attributes.
- Declare annotations on fields or accessors with respect to the database table or object relationships.
- Create file named “hibernate.cfg.xml” ,consists of mapping with respect to the database.
- Create a class consists of main method which will bind the application and create the connection with the database.

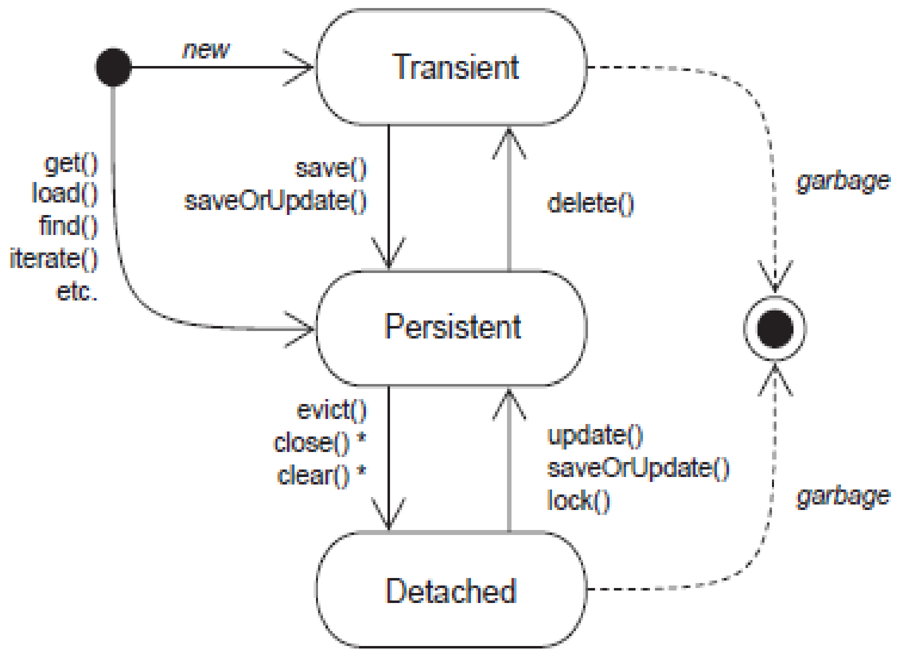
Working with objects

www.infogain.com

- Entity
  - An object which contains data is called an entity.
- Transient Entity
  - The instance is not associated with any persistence context. It has no persistent identity i.e. primary key.
- Persistent Entity
  - The instance is currently associated with a persistence context. It has a persistent identity and, perhaps, a corresponding row in the database.
- Detached Entity
  - The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process. It has a persistent identity and, perhaps, a corresponding row in the the database

The Persistence Lifecycle

www.infogain.com



* affects all instances in a Session

The Persistence Manager

- Any transparent persistence tool includes a *persistence manager API,* which usually provides services for
  - Basic CRUD operations
  - Query execution
  - Control of transactions
  - Management of the transaction-level cache

Session interface plays the role of persistent manager in hibernate.

Session Interface methods

www.infogain.com

- Commonly used method of session interface
  - Save
  - persist
  - Load
  - Get
  - Merge
  - update
  - beginTransaction
  - close

Making an Object Persistent

- A call to save() makes the transient instance to persistent.
- It’s now associated with the current Session. However, no SQL INSERT has yet been executed.
- The Hibernate Session never executes any SQL statement until absolutely necessary.

Retrieving a Persistent Object

- The Session is also used to query the database and retrieve existing persistent objects.

- special methods are provided on the Session API for the simplest kind of query: retrieval by identifier. One of these methods is get()

Eager Loading Vs Lazy Loading

www.infogain.com

Eager Loading

- State of the requested entity are loaded from the database when the entity is requested from the session.
- Get method provides the implementation of this approach.

Lazy Loading

- When an entity is requested ,simply a proxy is created and returned , no interaction to the database is made.
- When some state or relation is asked from the proxy only then thay are loaded from the database.
- Load method provides the implementation of this approach.

Updating the Persistent State Of a Detached Instance

- When the session is closed, object becomes a detached instance. It may be re associated with a new Session by calling update().

- Any persistent object returned by get() or any other kind of query is already associated with the current Session and transaction context.

- It can be modified, and its state will be synchronized with the database. This mechanism is called automatic dirty checking,
- which means Hibernate will track and save the changes you make to an object inside a session

Making a persistent object transient

www.infogain.com

- The SQL DELETE will be executed only when the Session is synchronized with the database at the end of the transaction.
- After the Session is closed, the user object is considered an ordinary transient instance.
- The transient instance will be destroyed by the garbage collector if it's no longer referenced by any other object.
- Both the in-memory object instance and the persistent database row will have been removed.

- Note: For deletion of object , we should use load or get method?

Hibernate Generator Class

www.infogain.com

- In hibernate mapping file, we used <generator /> in the id element.
- assigned is the default generator.
- This assigned means hibernate will not generate the primary key while saving any object. user has to take the responsibility.
- While saving an object into the database, the generator informs to the hibernate that, how the primary key value for the new record is going to generate
- hibernate using different primary key generator algorithms, for each algorithm internally a class is created by hibernate for its implementation
- hibernate provided different primary key generator classes and all these classes are implemented from
- **org.hibernate.id.IdentifierGeneratar Interface**

List of Generators

- assigned
- increment
- sequence
- identity
- native

Assigned

www.infogain.com

- This generator supports in all the databases
- This is the default generator class used by the hibernate, if we do not specify <generator –> element under id element then hibernate by default assumes it as “assigned”
- If generator class is assigned, then the programmer is responsible for assigning the primary key value to object which is going to save into the database
    - <id name="prodId" column="pid">
    - <generator />
    - </id>

Increment

- It is supported by all the databases i.e database independent
- This generator is used for generating the id value for the new record by using the formula
- Max of id value in Database + 1
- If we manually assigned the value for primary key for an object, then hibernate doesn’t considers that value and uses max value of id in database + 1 concept only
- If there is no record initially in the database, then for the first time this will saves primary key value as 1, as…
- max of id value in database + 1
  0 + 1
  result -> 1

Sequence

www.infogain.com

- Not supported in MySql
- This generator class is database dependent it means, we cannot use this generator class for all the database.
- While inserting a new record in a database, hibernate gets next value from the sequence under assigns that value for the new record

Identity

- It will not work for oracle
- <id name="productid" column="pid">
- <generator class="......."/>
- </id>

Native

www.infogain.com

- when we use this generator class, it first checks whether the database supports identity or not, if not checks for sequence and if not, then hilo will be used finally the order will be..
- identity
- sequence
- Hilo

- For example, if we are connecting with oracle, if we use generator class as native then it is equal to the generator class sequence.
- Foreign
- This generator is used in one-to-one relationship.

Mapping in Hibernate

INHERITANCE MAPPING

Class Inheritance

- Object oriented systems can model both “is a” and “has a” relationship.
- Inheritance Mapping allows to maintain inheritance hierarchy classes with the table of the database.
- Types of inheritance Mapping
  - Table Per Class Hierarchy
  - Table Per Subclass
  - Table Per Concrete Class joined strategy

Table Per Class Hierarchy

- In Table per Class Hierarchy scheme, objects of a family are mapped to a single table .
- This table contains columns to store data members of all the classes .
- A discriminator is used to uniquely identify the which record in the table represents object of which classes of the family.

Single Table Per Class Hierarchy (contd..)

www.infogain.com

- Advantage of Single Table per Class Hierarchy Scheme.
  - This hierarchy offers the best performance even for in the deep hierarchy .
- Disadvantage
  - Table is not normalized
  - Not null constraint can not be applied to the columns which are mapped to the data members of sub classes.

Inheritance Mapping (contd..)

- **Implementing One Table Per Class Hierarchy**
  - @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
  - @DiscriminatorColumn
  - @DiscriminatorValue
  - These are the annotations used for mapping table per class hierarchy strategy.
  - Here discriminator column identify the class type.

Table Per Subclass

- In Table per sub class scheme, a separate table is created for each class of the family .
- Each table contains columns to store declared as well as inherited data members of the class.

Table Per Subclass(Contd..)

- **Advantages of Table per Subclass scheme.**
  - More Normalized.
  - Not null constraints can be enforced.

- **Disadvantages of Table per Subclass scheme.**
  - Major disadvantage is that only state of objects is saved ,their relation is lost.
  - For this reason it is not supported by most of the frameworks.
  - Duplicate columns are created in the subclass tables.

- Note: This strategy is not popular and also have been made optional in Java Persistence API.

Table Per Subclass(Contd..)

- **Implementing Table Per sub Class**
    - @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
    - It should be specified in the parent class only.
    - In table structure, parent class table columns will be added in the subclass table.

Table Per Concrete Class joined strategy

www.infogain.com

- Each class persist the data in its own separate table. Each table contains columns only to store declared data members of the class.
- All the tables of the subclasses are joined to the super class table by using the primary key of the super class as foreign key in sub class tables.
- Inherited property remains in parent class table itself.

- Note:- That a foreign key relationship exists between the subclass tables and super class table.

Table Per Concrete Class Joined Strategy(Contd..)

www.infogain.com

- **Advantage**
  - Tables are normalized.
  - Not null constraints can be applied onto the columns mapped to the data members of sub classes.
- **Disadvantage**
  - As the hierarchy grows, it may result in poor performance.
  - Multiple queries and joins are required to save and load objects of subclasses.

Inheritance Mapping (contd..)

- **Implementing Table Per Concrete Class joined strategy**
  - Use Inheritance(strategy=InheritanceType.JOINED) in the parent class.
  - Use @PrimaryKeyJoinColumn annotation in the subclasses.

infogain

Thank You