Technologie odwzorowania obiektowo-relacyjnego: Hibernate

Plan prezentacji

- Wprowadzenie do technologii odwzorowania obiektowo-relacyjnego
- Architektura Hibernate
- Odwzorowanie klas na tabele w bazie danych
- Hibernate w akcji
- Odwzorowanie asocjacji
- Pakiet narzędzi Hibernate Tools

Technologie dostępu do baz danych z aplikacji J2EE

- **JDBC**
  - Podstawowy interfejs dostępu do baz danych
  - Ręczne kodowanie w JDBC uciążliwe i podatne na błędy
- **SQLJ**
  - Naturalne zagnieżdżanie poleceń SQL w kodzie Java
  - Odporna na błędy alternatywa dla bezpośredniego korzystania z JDBC
  - „Martwy” standard
- **Encyjne EJB**
  - Nienaturalne, złożone, dyskusyjna efektywność, „failed technology”
  - Trwają prace nad przebudowaną wersją 3.0 specyfikacji
- **Biblioteka znaczników JSTL SQL**
  - Wygodna w prostych aplikacjach opartych na JSP
  - Miesza logikę biznesową z logiką prezentacji, narusza model MVC
- **Technologie odwzorowania obiektowo-relacyjnego**
  - Efektywnie mapują świat obiektów na świat relacyjnej bazy danych
  - Najczęściej działają w warstwie webowej aplikacji
  - Najlepsze rozwiązanie w chwili obecnej

Technologie odwzorowania obiektowo-relacyjnego

- Implementacja aplikacji pracujących na relacyjnej bazie danych w obiektowych językach programowania może być czasochłonna i uciążliwa
- Odwzorowanie obiektowo-relacyjne (ang. O/RM (ORM) – object/relational mapping) to odwzorowanie reprezentacji danych z modelu obiektowego do modelu relacyjnego (wykorzystującego SQL)
  - Inne spojrzenie: Obiektowa reprezentacja relacyjnej bazy danych
- Popularne implementacje technologii odwzorowania obiektowo-relacyjnego dla aplikacji Java:
  - **Hibernate** (Open Source)
  - **Oracle Toplink** (rozwiązanie firmowe Oracle)
  - **JDO** (specyfikacja firmy Sun)

Charakterystyka technologii odwzorowania obiektowo-relacyjnego

- O/RM jest technologią skomplikowaną!
  - Jej implementacje są bardziej złożone niż np. szkielety aplikacji webowych takie jak Struts
- Zalety O/RM:
  - Produktywność (uniknięcie tworzenia uciążliwego kodu obsługującego bazę danych)
  - Łatwość pielęgnacji aplikacji (mniej kodu, elastyczność)
  - Wydajność (przy założeniu ograniczonego czasu i budżetu na zbudowanie i optymalizację aplikacji)
  - Niezależność od konkretnego SZBD
- Największe korzyści technologie O/RM przynoszą w aplikacjach pracujących na złożonym modelu obiektowym
  - Asocjacje, dziedziczenie, związki kompozycji, kolekcje

Elementy technologii odwzorowania obiektowo-relacyjnego

- API do wykonywania operacji CRUD na obiektach klas trwałych
- Język lub API do wykonywania zapytań odwołujących się do klas i właściwości
- Mechanizm specyfikowania metadanych opisujących odwzorowanie klas na relacje w bazach danych
- Techniki interakcji z obiektami trwałymi m.in.
  - Wykrywanie zmian w trwałych obiektach
  - Różne sposoby pobierania obiektów powiązanych asocjacjami
  - Techniki optymalizacji

Wprowadzenie do Hibernate

- Hibernate to technologia O/RM dla Javy
  - Z uwzględnieniem asocjacji, kompozycji, dziedziczenia, polimorfizmu, kolekcji
- Funkcjonalność Hibernate:
  - Odwzorowanie klas Javy na tabele w bazie danych
    - I typów danych Javy na typy danych SQL
  - Dostarczanie mechanizmów wykonywania zapytań do bazy danych
    - Pozwala uniknąć ręcznego zarządzania danymi na poziomie SQL i JDBC
- Wg dokumentacji celem Hibernate jest zwolnienie projektanta aplikacji z konieczności ręcznego kodowania 95% zadań związanych z zapewnieniem trwałości danych
- Hibernate jest najbardziej odpowiedni dla aplikacji Java pracujących w warstwie pośredniej
- Rozpowszechniany na licencji FSF Lesser Gnu Public License
- Najpopularniejsza obecnie implementacja odwzorowania obiektowo-relacyjnego dla Javy
  - Dużo materiałów i pomocy w Internecie
  - Udział twórców Hibernate w pracach nad specyfikacją EJB 3.0

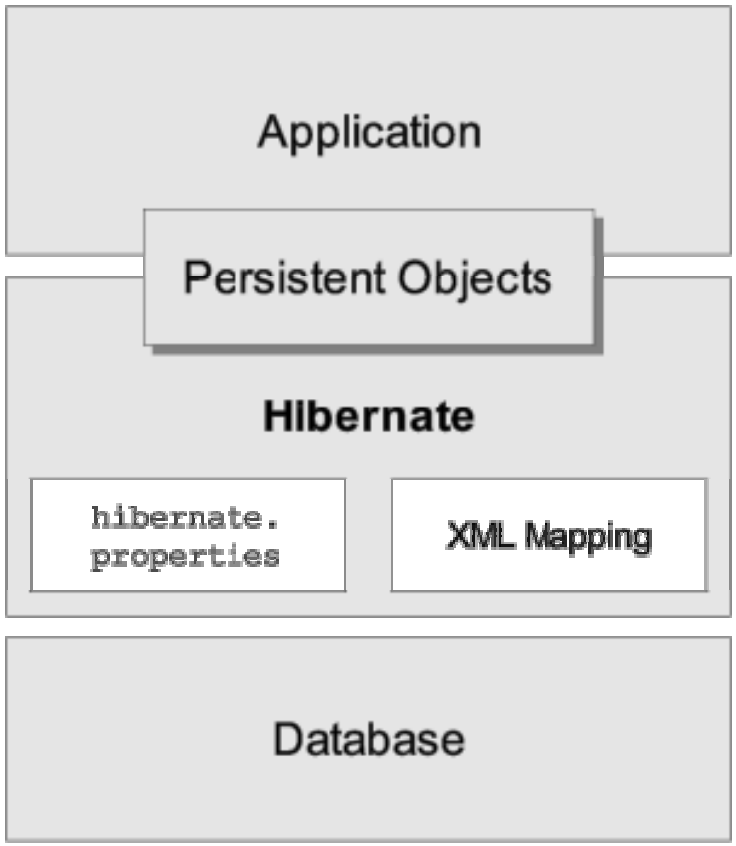
Dlaczego Hibernate?

- Wsparcie dla stylu programowania odpowiedniego dla Javy (Hibernate: Relational Persistence For Idiomatic Java)
  - Obsługa asocjacji, kompozycji, dziedziczenia, polimorfizmu, kolekcji
- Wysoka wydajność i skalowalność
  - Dual-layer cache, możliwość wykorzystania w klastrze
  - UPDATE tylko dla zmodyfikowanych obiektów i kolumn
- Wiele sposobów wydawania zapytań:
  - HQL – własne przenaszalne rozszerzenie SQL
  - Natywny SQL
  - Zapytania przez obiekty Javy: Criteria i Example
- Wykorzystuje siłę technologii relacyjnych baz danych, SQL, JDBC
- Professional Open Source
  - Zalety rozwiązań Open Source
  - Wsparcie uznanej firmy JBoss Inc.
  - Komponent JBoss Enterprise Middleware System
  - Elastyczna licencja LGPL
- Hibernate implementuje język zapytań i persistence API z EJB 3.0
  - Hibernate EntityManager i Annotations ponad Hibernate Core

Biblioteki Hibernate

- Hibernate można pobrać z http://hibernate.org/
  - Dystrybucja zawiera bibliotekę Hibernate i wykorzystywane przez nią biblioteki od innych dostawców (w katalogu /lib)
- Biblioteka JAR Hibernate: hibernate3.jar
- 3rd Party Libraries wykorzystywane przez Hibernate:
  - antlr
  - dom4j
  - CGLIB
  - Commons Collections, Commons Logging
  - EHCache
  - Log4j (opcjonalna, nie dołączać w JDeveloper 10.1.3)
  - asm, jta

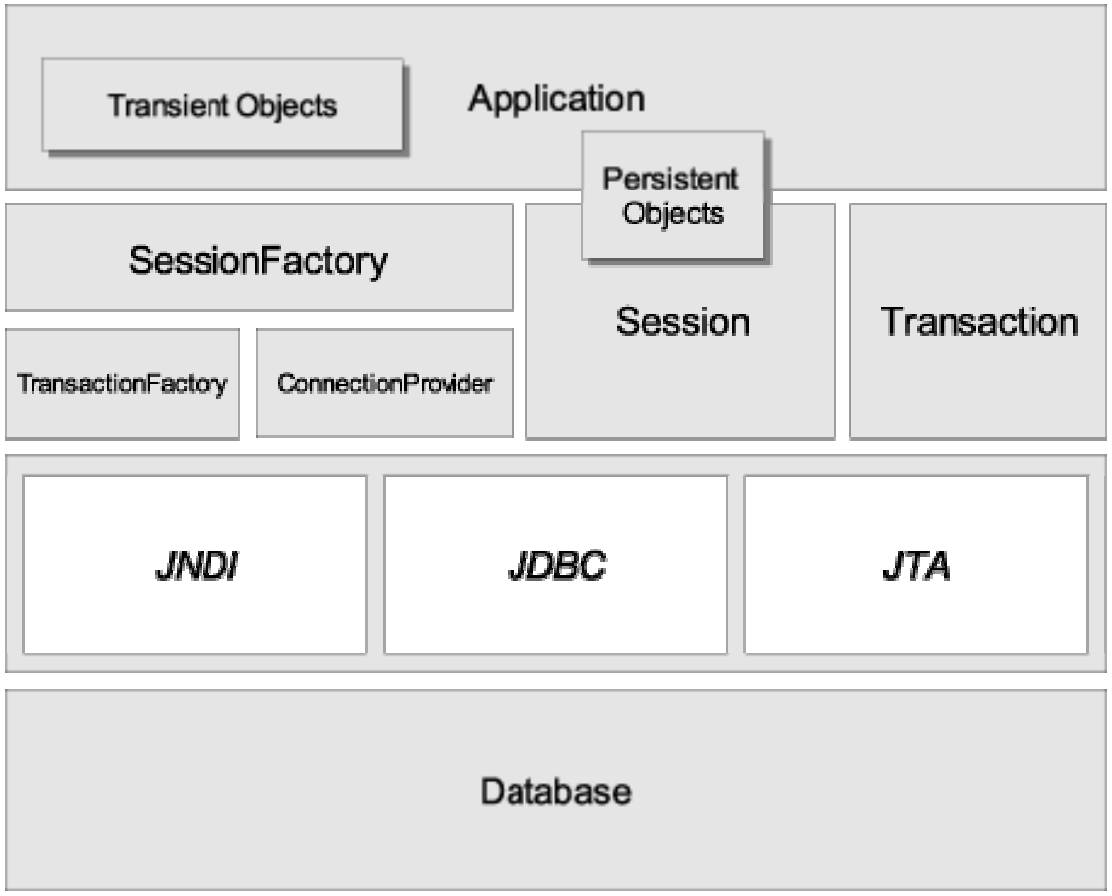
Ogólna architektura Hibernate



* Schemat z Hibernate Reference

Szczegółowa architektura Hibernate

- Wariant zakładający pełne wykorzystanie możliwości Hibernate w zakresie separacji aplikacji od JDBC/JTA API



* Schemat z Hibernate Reference

Konfiguracja Hibernate

- Konfiguracja programowa
  - Na poziomie aplikacji odwzorowania typów Java na relacyjną bazę danych reprezentuje obiekt klasy **`org.hibernate.cfg.Configuration`**
  - Odwzorowania klas Java na tabele są definiowane w plikach XML
  - Parametry konfiguracyjne Hibernate są ustawiane programowo poprzez obiekt **`Configuration`** lub definiowane w tekstowym pliku hibernate.properties

```
Configuration cfg = new Configuration()
.addResource("Emp.hbm.xml")
.addResource("Dept.hbm.xml")
.setProperty("hibernate.connection.datasource",
"java:comp/env/jdbc/test");
```

Pliki z odwzorowaniami pobierane ze ścieżki CLASSPATH

- Alternatywnym i wygodnym sposobem konfiguracji Hibernate jest wykorzystanie pliku konfiguracyjnego w formacie XML: hibernate.cfg.xml

Plik konfiguracyjny hibernate.cfg.xml

- Specyfikuje pełną konfigurację dla Hibernate:
  - Parametry konfiguracyjne (nadpisuje informacje z hibernate.properties, jeśli oba pliki dostępne)
  - Pliki XML z odwzorowaniami klas Java na tabele bazy danych
- Umieszczany w katalogu głównym CLASSPATH

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
  <!-- properties -->
  <property name="dialect">org.hibernate.dialect.OracleDialect</property>
  ...
  <!-- mapping files -->
  <mapping resource="myhib/Dept.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Dialekty SQL

- Wskazanie w pliku konfiguracyjnym dialektu SQL wykorzystywanego serwera bazy danych upraszcza konfigurację, gdyż wielu opcjonalnym parametrom Hibernate zostaną nadane odpowiednie wartości domyślne
- Niektóre dialekty obsługiwane przez Hibernate:
  - DB2: `org.hibernate.dialect.DB2Dialect`
  - PostgreSQL: `org.hibernate.dialect.PostgreSQLDialect`
  - MySQL: `org.hibernate.dialect.MySQLDialect`
  - MySQL with InnoDB: `org.hibernate.dialect.MySQLInnoDBDialect`
  - MySQL with MyISAM: `org.hibernate.dialect.MySQLMyISAMDialect`
  - Oracle (any version): `org.hibernate.dialect.OracleDialect`
  - Oracle 9i/10g: `org.hibernate.dialect.Oracle9Dialect`
  - Sybase: `org.hibernate.dialect.SybaseDialect`
  - Sybase Anywhere: `org.hibernate.dialect.SybaseAnywhereDialect`
  - Microsoft SQL Server: `org.hibernate.dialect.SQLServerDialect`
  - Informix: `org.hibernate.dialect.InformixDialect`

Połączenia z bazą danych poprzez DriverManager

- Zawartość hibernate.cfg.xml:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
  <!-- properties -->
  <property name="hibernate.connection.driver_class">
     oracle.jdbc.OracleDriver
  </property>
  <property  name="hibernate.connection.url">
     jdbc:oracle:thin:@localhost:1521:marek8i
  </property>
  <property name="hibernate.connection.username">scott</property>
  <property name="hibernate.connection.password">tiger</property>
  <property name="dialect">org.hibernate.dialect.OracleDialect</property>
  <!-- mapping files -->
  <mapping resource="myhib/Dept.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

Połączenia z bazą danych poprzez źródła danych (DataSource)

- Zawartość hibernate.cfg.xml:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
  <!-- properties -->
  <property name="hibernate.connection.datasource">jdbc/marek8iDS</property>
  <property name="hibernate.connection.username">scott</property>
  <property name="hibernate.connection.password">tiger</property>
  <property name="dialect">org.hibernate.dialect.OracleDialect</property>
  <!-- mapping files -->
  <mapping resource="myhib/Dept.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

- Rozwiązanie dla „środowisk zarządzanych” takich jak serwery aplikacji

Trwałe klasy (Persistent classes)

- Trwałe klasy to klasy w aplikacji, które implementują encje występujące w reprezentowanej rzeczywistości:
  - Np. Klient, Faktura
  - Nie wszystkie instancje trwałej klasy muszą być trwałe
- Hibernate najlepiej działa z klasami spełniającymi reguły Plain Old Java Object (POJO)
- Reguły POJO dla Hibernate:
  - Metody set/get (accessor/mutator) dla trwałych pól
    - Hibernate zapewnia trwałość właściwości w stylu JavaBeans
  - Bezargumentowy konstruktor (może być domyślny)
  - Identyfikator (opcjonalnie)
    - Hibernate może też wewnętrznie zarządzać identyfikatorami obiektu (niezalecane)
    - Pewna szczególna funkcjonalność dostępna tylko dla klas z identyfikatorem
    - Zalecany sztuczny identyfikator, typu nie-prostego (możliwość przypisania null)
    - Zalecane spójne nazewnictwo w klasach np. id
  - Klasa nie-final
    - Może być final, ale w pewnych przypadkach może to np. ograniczyć możliwości strojenia wydajności

Przykład POJO

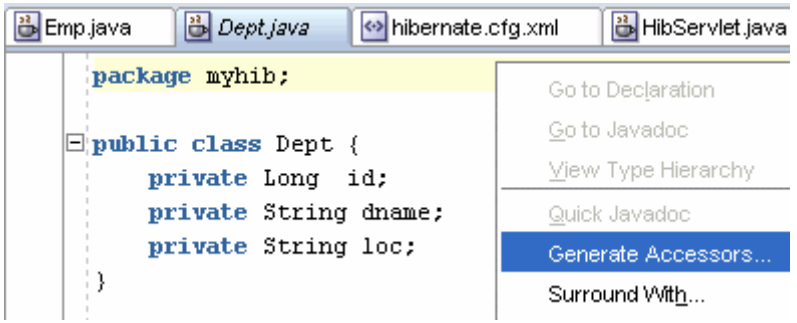
```
public class Dept {
  private Long   id;
  private String dname;
  private String loc;

    public void setId(Long id) {
        this.id = id;
    }
    public Long getId() {
        return id;
    }
    public void setDname(String dname) {
        this.dname = dname;
    }
    public String getDname() {
        return dname;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    public String getLoc() {
        return loc;
    }
}
```

Możliwa odpowiadająca tabela w Oracle:

```
DEPT
--------------------------------
DEPTNO NUMBER        PRIMARY KEY
DNAME  VARCHAR2(14)
LOC    VARCHAR2(13)
```

Środowiska Java IDE ułatwiają tworzenie POJO:



Odwzorowanie obiektowo-relacyjne

- Odwzorowanie definiowane w pliku XML
- Język do opisu odwzorowania jest zorientowany na Javę
  - Odwzorowanie budowane z punktu widzenia klasy Java a nie tabeli!
- Dokumenty opisujące odwzorowanie można tworzyć ręcznie lub korzystając z narzędzi
  - Generujących odwzorowanie z POJO (np. XDoclet)
  - Generujących POJO i odwzorowanie z tabel bazy danych na zasadzie reverse engineering (np. Hibernate Tools)
- Dokumenty opisujące odwzorowanie wskazywane w pliku konfiguracyjnym hibernate.cfg.xml (lub programowo w aplikacji)

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>

  ...
    <mapping resource="myhib/Dept.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Przykład odwzorowania

Dept.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="myhib">
<class name="Dept" table="dept">

<id name="id" type="long" column="deptno">
  <generator class="sequence">
    <param name="sequence">dept_seq</param>
  </generator>
</id>

<property name="dname"
          column="dname"
          type="string"
          not-null="true"
          update="true" insert="true"/>

<property name="loc"/>

</class>
</hibernate-mapping>
```

Odwzorowanie klasy na tabelę

Definicja identyfikatora
(dla Oracle oparty o sekwencję)

Nazwa kolumny w b.d. Domyślnie = name.

Typ danych Hibernate. Domyślny najczęściej OK.

Domyślnie false.

Domyślnie true. False dla pól wyliczeniowych

Uwagi o odwzorowaniu

- Odwzorowanie z założenia jest zwięzłe i proste
  - Wiele atrybutów dla **`class`** i **`property`**, ale sensowne wartości domyślne odpowiednie w większości przypadków
- Typy danych Hibernate
  - `integer, long, short, float, double, character, byte, boolean (BIT), yes_no (CHAR(1): Y/N), true_false (CHAR(1): T/F)`: odwzorowanie typów prostych Javy na odpowiednie typy SQL (vendor-specific)
  - `string`: odwzorowanie `java.lang.String` na `VARCHAR` (w Oracle `VARCHAR2`)
  - `date, time, timestamp`: odwzorowanie `java.util.Date` na `DATE, TIME, TIMESTAMP`
  - `big_decimal, big_integer`: odwzorowanie z `java.math.BigDecimal` i `java.math.BigInteger` na `NUMERIC` (w Oracle `NUMBER`)
  - `calendar, calendar_date, locale, timezone, currency, class, binary, text, serializable, clob, blob`

Odwzorowanie identyfikatora na klucz główny

- Odwzorowane klasy **muszą** deklarować klucz główny tabeli
  - Do odwzorowania na kolumnę klucza głównego służy **`<id>`**
  - Istnieje możliwość stosowania złożonych kluczy głównych z Hibernate, odwzorowanych przez **`<composite-id>`**, ale jest to mocno niezalecane!
- Specyfikacja odwzorowania klucza głównego zawiera element opisujący sposób generacji jego wartości (**`<generator>`**):
  - `assigned`: aplikacja musi wypełnić wartość (strategia domyślna)
  - `increment`: generacja programowa (działa gdy jeden proces w danej chwili może wstawiać wiersz do tabeli)
  - `identity`: wsparcie dla automatycznie generowanych wartości kolumny w DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL (np. AUTO_INCREMENT w MySQL)
  - `sequence`: oparty o sekwencję w bazie danych w DB2, PostgreSQL, Oracle, SAP DB, McKoi lub generator w Interbase
  - `select`: pobiera wartość ustawioną przez wyzwalacz (trigger) w b.d.
  - `hilo, uuid, ...`

Hibernate w akcji

```
SessionFactory sf = new Configuration().configure().buildSessionFactory();

Session s = sf.openSession();            // rozpoczęcie sesji
Transaction tx = s.beginTransaction();   // rozpoczęcie transakcji
Dept d = new Dept();
d.setDname("MARKETING");
d.setLoc("DETROIT");
s.save(d);               // zapis obiektu do bazy danych
tx.commit();             // zatwierdzenie transakcji (wycofanie: tx.rollback())
s.close();               // zakończenie sesji
```

- **SessionFactory** – obiekt za pomocą którego tworzone są sesje
  - Z założenia tworzony raz, na początku pracy aplikacji
  - Współdzielony przez wiele wątków
  - Utworzenie jest kosztowne
- **Session** – „unit of work”
  - Do jednokrotnego wykorzystania, dla jednego procesu biznesowego (non-treadsafe)
  - Tworzenie mało kosztowne
  - Typowy model: „session-per-request”
- **Transaction** – transakcja w bazie danych
  - Interfejs Transaction oddziela aplikację od implementacji transakcji (JDBC/JTA)
  - Często jedna transakcja w sesji, ale możliwe wiele kolejnych transakcji w jednej sesji
  - Hibernate nie pozwala na tryb auto-commit!

Uwagi o sesjach Hibernate

- Sesja (Session) w Hibernate jest czymś pomiędzy połączeniem a transakcją
- Można interpretować sesję jako kolekcję (cache) obiektów załadowanych do pamięci, związanych z jednostką pracy użytkownika („unit of work”)
  - Hibernate automatycznie wykrywa zmiany w obiektach związanych z sesją
- Sesja (Session) określana jest często również mianem zarządcy trwałością obiektów (persistence manager), gdyż stanowi interfejs do składowania i pobierania obiektów z bazy danych
- Uwaga: pojęcie sesji w Hibernate nie ma nic wspólnego z sesją w aplikacji webowej (HttpSession)!

Współdzielenie SessionFactory (1/2)

- Możliwe poprzez klasę pomocniczą implementującą wzorzec projektowy singleton (jedna instancja klasy składowana w statycznej zmiennej)
  - Stosowalne w aplikacjach klienckich i na serwerze aplikacji (np. w serwlecie)

```
public class HibernateUtil {
  private static final SessionFactory sessionFactory;
  static {
    try {
      sessionFactory = newConfiguration().configure().buildSessionFactory();
    } catch (Throwable ex) {
        System.err.println("Initial SessionFactory creation failed." + ex);
        throw new ExceptionInInitializerError(ex);
      }
  }
  public static SessionFactory getSessionFactory() {
    return sessionFactory;
  }
}
```

Sposób wykorzystania:

```
SessionFactory sf = HibernateUtil.getSessionFactory();
```

Współdzielenie SessionFactory (2/2)

- W aplikacjach na serwerze aplikacji (np. w serwlecie) można uzyskiwać dostęp do współdzielonej SessionFactory przez JNDI
- Jeśli w pliku konfiguracyjnym podana została nazwa JNDI dla SessionFactory, to przy jej tworzeniu zostanie ona związana z tą nazwą w JNDI (jeśli serwer udostępnia serwis JNDI)

Konfiguracja (plik konfiguracyjny Hibernate):

hibernate.cfg.xml

```
...
<hibernate-configuration>
  <session-factory name="java:hibernate/SessionFactory">
  ...
  </session-factory>
</hibernate-configuration>
```

Tworzenie (raz gdzieś w kodzie inicjalizującym):

```
new Configuration().configure().buildSessionFactory();
```

Sposób wykorzystania (w aplikacji na serwerze J2EE, np. w serwlecie):

```
InitialContext ic = new InitialContext();  //javax.naming.*
SessionFactory sf = (SessionFactory) ic.lookup("java:hibernate/SessionFactory");
```

Stany obiektów w Hibernate

- Twórcy aplikacji korzystający z Hibernate powinni myśleć o stanie obiektów przetwarzanych przez aplikację a nie o wykonywanych poleceniach SQL
  - Hibernate generuje i wysyła do bazy danych polecenia SQL
  - Twórca aplikacji interesuje się tym ewentualnie przy strojeniu
- Stany obiektu w Hibernate:
  - **Transient** (ulotny)
    - utworzony operatorem **`new`**, ale niezwiązany z sesją
  - **Persistent** (trwały)
    - posiada identyfikator i reprezentację w bazie danych
    - związany z sesją
    - Hibernate wykrywa zmiany dokonane na trwałych obiektach i synchronizuje ich stan z bazą danych
  - **Detached** (odłączony)
    - obiekt, który był trwały, ale jego sesja się zakończyła
    - można go modyfikować, a następnie związać z nową sesją (funkcjonalność przydatna do realizacji długich „transakcji aplikacyjnych”)

Zapis i odczyt obiektu do/z bazy danych

- Uczynienie nowego obiektu trwałym

```
Dept d = new Dept();
d.setDname("MARKETING");
d.setLoc("MIAMI");
Long genId = (Long)s.save(d);
```

- Odczyt obiektu o znanym identyfikatorze

```
Dept d = (Dept) s.load(Dept.class, new Long(20));
System.out.println(d.getDname());
```

← Wyjątek gdy błędny klucz

```
Dept d = (Dept) s.get(Dept.class, new Long(20));
System.out.println(d.getDname());
```

← Zwraca null gdy błędny klucz

- Usuwanie obiektu

```
s.delete(d);   // obiekt stanie się ulotny, referencja może zostać
```

- Modyfikacja trwałego obiektu

```
// W czasie otwartej sesji (Session s):
d.setLoc("PORTLAND");
s.flush(); // jaśli zmiany mają natychmiast powędrować do bazy
```

Zaawansowane operacje dla obiektów odłączonych

- **`update`**() (lub **`saveOrUpdate`**())
  - Do zsynchronizowania z bazą danych stanu zmienionego odłączonego obiektu (w nowej sesji)
- **`saveOrUpdate`**()
  - W uproszczeniu: **`save`**() gdy nowy, w przeciwnym wypadku **`update`**()
- **`merge`**()
  - Skopiowanie stanu obiektu do trwałej instancji o tym samym identyfikatorze
    - Istniejącej w danej sesji
    - Załadowanej z bazy danych
    - Utworzonej

Uwagi o współbieżności

- **Poziom izolacji**
  - Hibernate domyślnie dostosowuje się do poziomu izolacji ustawionego dla bazy danych
  - Poziom izolacji dla Hibernate można podać jawnie za pomocą opcji konfiguracyjnej hibernate.connection.isolation (1,2,4,8)
- **Blokowanie**
  - Hibernate domyślnie stosuje blokowanie **optymistyczne** z **wersjonowaniem**
    - Dla pełnego bezpieczeństwa wymagane utworzenie kolumny version/timestamp w każdej tabeli
    - Automatyczne blokowanie optymistyczne bez dodatkowych kolumn z informacjami o wersji działa prawidłowo (unikając problemu Lost Update) tylko w ramach danej sesji – problemy z obiektami odłączonymi
  - Można zlecić jawnie blokowanie **pesymistyczne** dla konkretnych operacji odczytu obiektu z bazy danych (`Session.get`())
- Dwupoziomowy **cache**
  - Poziom sesji (Session)
  - Poziom SessionFactory

Zapytania do bazy danych

- Zapytania w języku HQL (Hibernate Query Language)
  - Język od strony składni wyglądający jak SQL
    - SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, połączenia, podzapytania (jeśli wykorzystywany SZBD je wspiera)
  - W pełni zorientowany obiektowo
    - dziedziczenie, polimorfizm, asocjacje
- Zapytania w natywnym SQL
  - Możliwość wykorzystania specyficznych konstrukcji np. CONNECT
- Zapytania poprzez obiekty **`Criteria`**
  - Budowa zapytań poprzez obiektowe API
- Zapytania poprzez obiekty **`Example`**
  - Kryteria zapytania budowane w oparciu o przykładową instancję (QBE – Query By Example)
- Filtry
  - Aplikowane do kolekcji lub tablic

Wykonywanie zapytań (HQL): list()

- Wykonywanie zapytań poprzez **`list`**()
  - Zwraca cały wynik zapytania do kolekcji w pamięci (instancje pozostają w stanie trwałym)

```
List depts = (List)s.createQuery(
"from Dept as dept where dept.loc = 'DALLAS'")
.list();

for (int i=0; i<depts.size(); i++)
  System.out.println(((Dept)depts.get(i)).getDname());
```

```
RESEARCH
MARKETING
MARKETING
MARKETING
```

```
List depts = (List)s.createQuery(
"select dept.id, dept.dname from Dept as dept where
dept.loc = 'DALLAS'")
.list();

for (int i=0; i<depts.size(); i++)
{
  Object [] tab = (Object []) depts.get(i);
  System.out.println(tab[0]+" "+tab[1]);
}
```

```
20 RESEARCH
53 MARKETING
71 MARKETING
72 MARKETING
```

Wykonywanie zapytań (HQL): iterate()

- Wykonywanie zapytań poprzez **`iterate`**()
  - Zwraca wynik w kilku zapytaniach SELECT:
    - Pobranie identyfikatorów
    - Oddzielne zapytania pobierające poszczególne instancje
  - Może być efektywniejsze od **`list`**(), gdy instancje już są w pamięci podręcznej, ale zazwyczaj wolniejsze

```
Iterator depts = s.createQuery(
"from Dept as dept where dept.loc = 'DALLAS'")
.iterate();

while (depts.hasNext())
{
  Dept d = (Dept) depts.next();
  System.out.println(d.getDname());
}
```

```
RESEARCH
MARKETING
MARKETING
MARKETING
```

Przykłady zapytań w HQL

```
from
mypackage.Dept
```

```
from
Dept
```

```
select dept.dname
from Dept dept
where dept.loc like 'D%'
```

```
from
java.lang.Object
```

```
select dept.dname||' '||dept.loc
from Dept dept
```

```
from Emp
where comm is not null
order by sal desc, ename
```

```
select sum(sal)
from Emp
group by dept.id
```

```
from Emp e, Dept d
where e.dept.id = d.id
```

```
from Dept as dept join dept.emps as emp
```

```
from Dept as dept left outer join dept.emps as emp
```

```
select distinct emp.dept.dname
from Emp as emp
order by 1
```

```
select sum(sal)
from Emp
group by dept.id
having sum(sal) >
9000
```

```
select new
java.lang.String(ename)
from Emp
```

HQL – Pobieranie obiektów z bazy danych

- Domyślnie w przypadku operacji połączenia, HQL nie pobiera natychmiast związanych obiektów i kolekcji
  - Domyślnie są one pobierane gdy nastąpi do nich pierwsze odwołanie (tryb „lazy”)
  - HQL ignoruje w tym względzie ewentualne ustawienia podane przy odwzorowaniu
  - Stanowi to problem, gdy odwołanie do dowiązanego obiektu lub kolekcji nastąpi po zamknięciu sesji, w której wykonano zapytanie
  - Rozwiązaniem jest zastosowanie klauzul (działają dla **`list`**()):
    - INNER JOIN FETCH - dla pobrania pojedynczych obiektów
    - LEFT JOIN FETCH – dla pobrania kolekcji

```
from Dept as dept left join fetch dept.emps as emp
```

```
from Emp as emp inner join fetch emp.dept
```

Przetwarzanie wyników zapytań – Dodatkowe mechanizmy

- Zapytania sparametryzowane (styl ? lub :nazwa)

```
List depts = (List)s.createQuery(
  "from Dept as dept where dept.loc = ?")
  .setString(0, "DALLAS")                    // numeracja od zera!
  .list();
```

- Paginacja

```
Query q = s.createQuery("from Dept as dept where dept.loc = 'DALLAS'");
q.setFirstResult(0);  q.setMaxResults(2);
List depts = q.list();
```

- Przewijalne wyniki zapytań
  - Poprzez rzutowanie do **`ScrollableResults`**
  - Wymaga otwartego połączenia z bazą i otwartego kursora
  - Dostępne gdy sterownik JDBC wspiera przewijalne zbiory wynikowe

```
Query q = s.createQuery("select ... from ...");
ScrollableResults kursor = q.scroll();
```

Rodzaje asocjacji w Hibernate

- Jednokierunkowe
  - N:1
  - 1:1
  - 1:N
- Jednokierunkowe z tabelą pośrednią
  - N:1
  - 1:1
  - 1:N
  - N:M
- Dwukierunkowe
  - N:1
  - 1:1
  - 1:N
- Dwukierunkowe z tabelą pośrednią
  - N:1
  - 1:1
  - 1:N
  - N:M

Asocjacja 1:N – Przykład (1/5)

PRZYKŁAD:
Implementacja
w postaci
jednokierunkowej
1:N

MGR_FK

EMP

EMPNO	NUMBER(4)	PRIMARY KEY
ENAME	VARCHAR2(10)	NOT NULL UNIQUE
JOB	VARCHAR2(9)	NOT NULL
MGR	NUMBER(4)	NOT NULL
HIREDATE	DATE	NOT NULL
SAL	NUMBER(7,2)	NOT NULL
COMM	NUMBER(7,2)	
DEPTNO	NUMBER(2)	NOT NULL

DEPTNO_FK

PRZYKŁAD:
Implementacja w postaci
dwukierunkowej 1:N

DEPT

DEPTNO	NUMBER(2)	PRIMARY KEY
DNAME	VARCHAR2(14)	NOT NULL UNIQUE
LOC	VARCHAR2(13)	NOT NULL

Asocjacja 1:N – Przykład (2/5)

- POJO i odwzorowanie dla Dept

Dept.java

```
public class Dept {
  private Long  id;
  private String dname;
  private String loc;
  private Set emps;
...
}
```

Przechodnia trwałość: **all** oznacza, że wszystkie operacje (zapis, usuwanie, ...) propagują się na obiekty podrzędne

Dept.hbm.xml

```
<hibernate-mapping package="myhib">
<class name="Dept" table="dept">
...
  <set name="emps" inverse="true" cascade="all">
    <key column="deptno"/>
    <one-to-many class="Emp"/>
  </set>
</class>
</hibernate-mapping>
```

Drugi „koniec” asocjacji spowoduje wygenerowanie polecenia DML

Asocjacja 1:N – Przykład (3/5)

- POJO dla Emp

Emp.java

```
public class Emp {
  private  Long id;
  private  String ename;
  private  String job;
  private  Emp mgr;
  private  Date hiredate;
  private  double sal;
  private  Double comm;
  private  Dept dept;
...
 public boolean equals(Object other) {
      if (this == other) return true;
      if ( !(other instanceof Emp) ) return false;
      final Emp emp = (Emp) other;
      if ( !emp.getEname().equals( getEname() ) ) return false;
      return true;
    }
    public int hashCode() {
      return getEname().hashCode();
    }
}
```

Wymagane gdy instancje trwałych klas mają być składowane w zbiorach Set. Implementacje nie mogą odwoływać się do klucza głównego (muszą się opierać o naturalny klucz kandydujący).

Asocjacja 1:N – Przykład (4/5)

- Odwzorowanie dla Emp

Emp.hbm.xml

```
<hibernate-mapping package="myhib">
<class name="Emp" table="emp">

<id name="id" type="long" column="empno">
  <generator class="sequence">
    <param name="sequence">emp_seq</param>
  </generator>
</id>

<property name="ename"/>
<property name="job"/>

<many-to-one name="mgr"
column="mgr"
unique="true"
not-null="true"/>

<property name="hiredate" type="date"/>
<property name="sal"/>
<property name="comm"/>

<many-to-one name="dept"
column="deptno"
not-null="true"/>

</class>
</hibernate-mapping>
```

Asocjacja 1:N – Przykład (5/5)

```
SessionFactory sf = HibernateUtil.getSessionFactory();
Session s = sf.openSession();
Transaction tx = s.beginTransaction();

Dept d = new Dept();
d.setDname("SALES");
d.setLoc("POZNAN");
d.setEmps(new HashSet());

Emp e = new Emp();
e.setEname("KOWALSKI");
e.setSal(2000);

e.setDept(d);
d.getEmps().add(e);

s.save(d);

tx.commit();
s.close();
```

Dla dwukierunkowej asocjacji należy obsłużyć oba jej końce!

Dla ułatwienia można w klasie Dept zdefiniować metodę pomocniczą addEmployee():

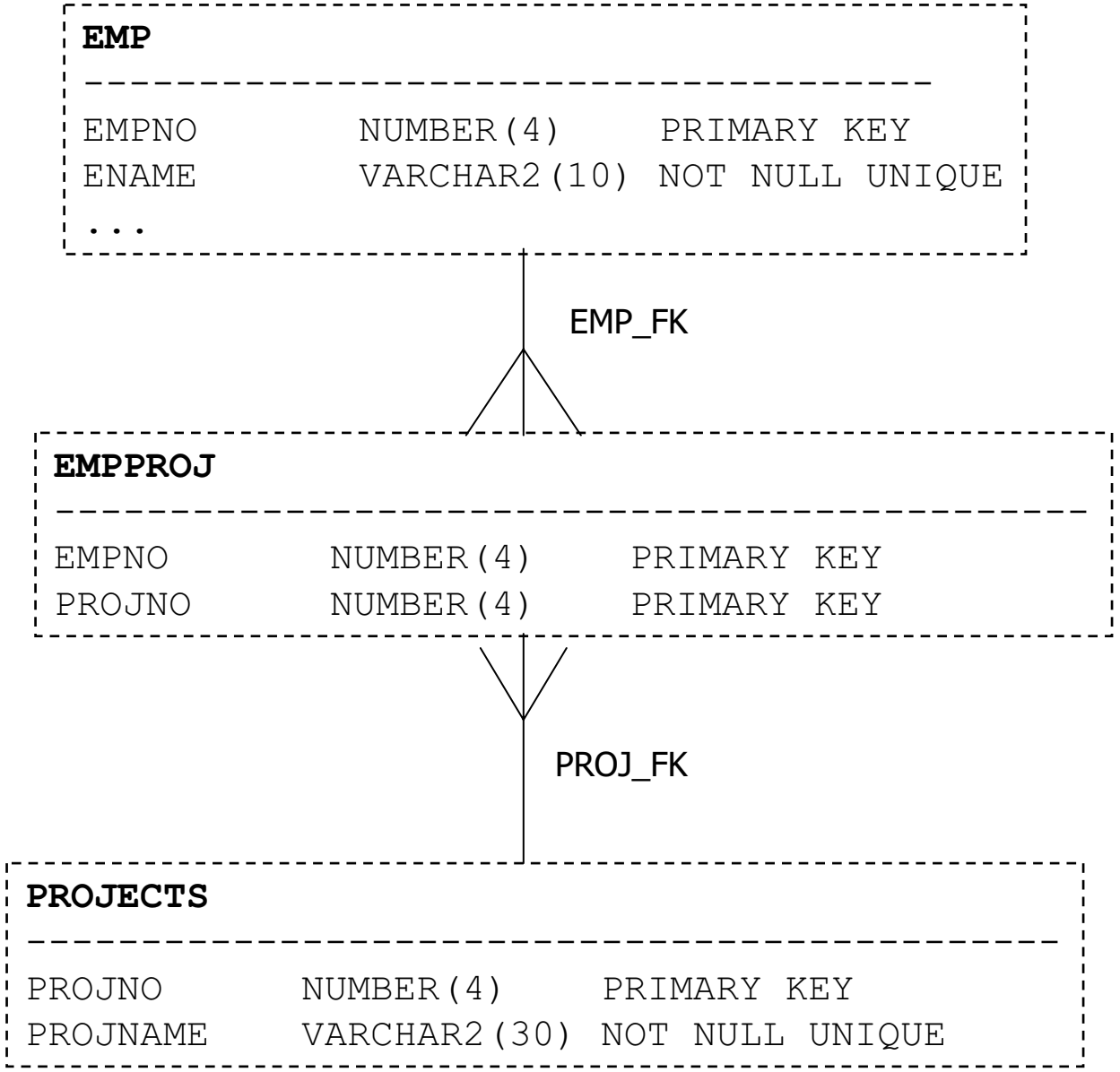
```
public class Dept {
...
   public void addEmployee(Emp e) {
     if (e.getDept() != null)
      e.getDept().getEmps().remove(e);
    e.setDept(this);
    emps.add(e);
  }
}
```

Dept.java

Dzięki **cascade="all"** zachowany będzie również powiązany asocjacją z **d** obiekt **e**

Asocjacja M:N – Przykład (1/4)

PRZYKŁAD:
Implementacja
w postaci
dwukierunkowej
M:N



Asocjacja M:N – Przykład (2/4)

- POJO i odwzorowanie dla Emp

Emp.java

```
public class Emp {
  private Long  id;
  private String ename;
...
  private Set projs;
...
}
```

Pobieranie powiązanych obiektów, gdy będą potrzebne (**lazy** domyślne dla kolekcji)

Emp.hbm.xml

```
<hibernate-mapping package="myhib">
<class name="Emp" table="emp">
...
<set name="projs"
  table="EMPPROJ"
  lazy="true"
  cascade="save-update">
  <key column="EMPNO"/>
  <many-to-many class="Project" column="PROJNO"/>
</set>
</class>
</hibernate-mapping>
```

W tym wypadku **all** nieodpowiednie, bo nie chcemy kaskadowego usuwania

Asocjacja M:N – Przykład (3/4)

- POJO i odwzorowanie dla Project

Project.java

```
public class Project {
  private Long  id;
  private String projname;
  private Set emps;
}
```

Project.hbm.xml

```
<hibernate-mapping package="myhib">
<class name="Project" table="projects">
<id name="id" type="long" column="projno">
  <generator class="sequence">
    <param name="sequence">proj_seq</param>
  </generator>
</id>

<property name="projname"/>

<set name="emps"
  table="EMPPROJ"
  lazy="true"
  inverse="true"
  cascade="save-update">
  <key column="PROJNO"/>
  <many-to-many class="Emp" column="EMPNO"/>
</set>
</class>
</hibernate-mapping>
```

Drugi „koniec” asocjacji spowoduje wygenerowanie polecenia DML

Asocjacja M:N – Przykład (4/4)

```
SessionFactory sf = HibernateUtil.getSessionFactory();
Session s = sf.openSession();
Transaction tx = s.beginTransaction();

Emp e = (Emp) s.get(Emp.class, new Long(7902));
  System.out.println(e.getEname());
Project p = (Project) s.get(Project.class, new Long(10));
  System.out.println(p.getProjname());

p.getEmps().add(e);
e.getProjs().add(p);

s.save(p);

for (Emp ep : (Set<Emp>) p.getEmps())
  System.out.println(ep.getEname());

for (Project pe : (Set<Project>) e.getProjs())
  System.out.println(pe.getProjname());

tx.commit();
s.close();
```

Dla dwukierunkowej asocjacji należy obsłużyć oba jej końce!

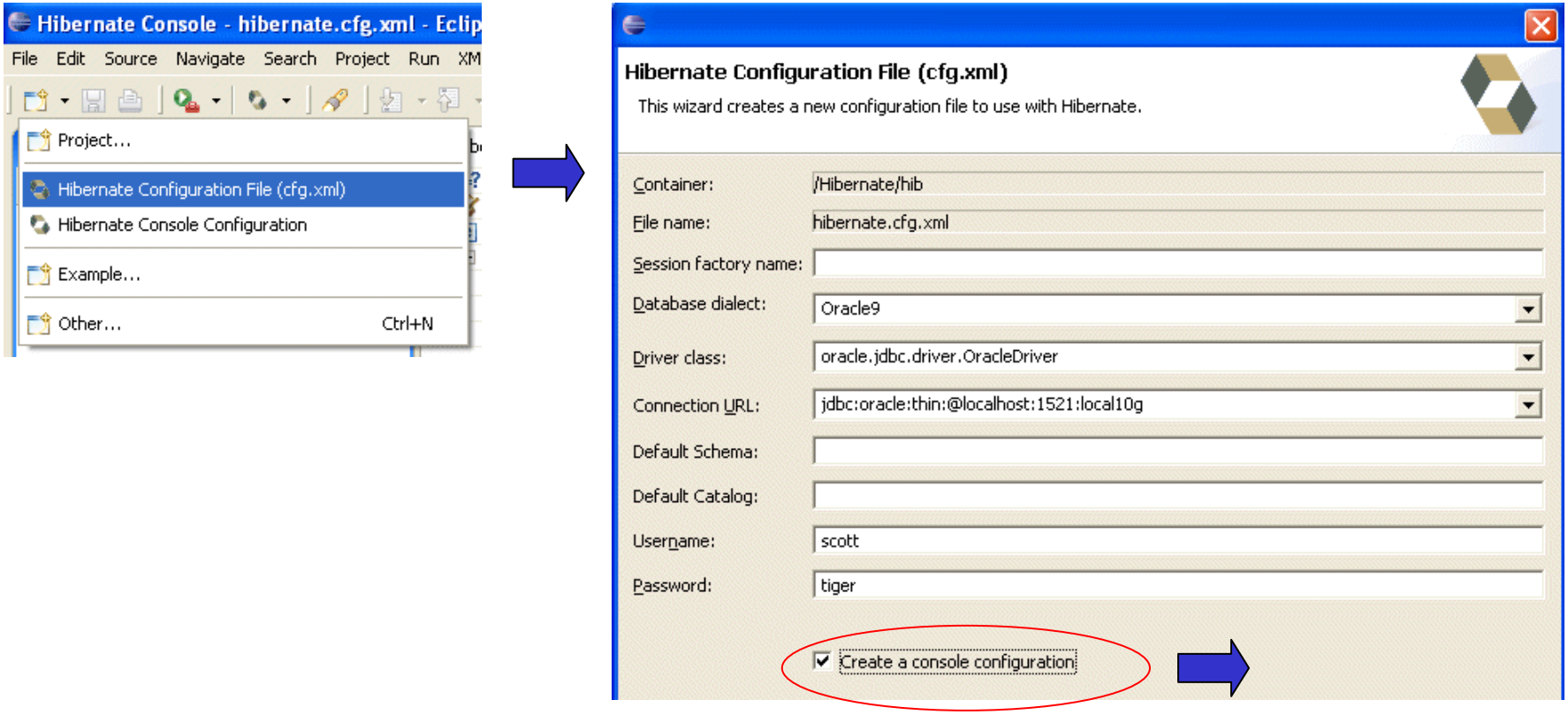
Dzięki **cascade="save-update"** zachowany będzie również powiązany asocjacją z **p** obiekt **e**

Hibernate Tools

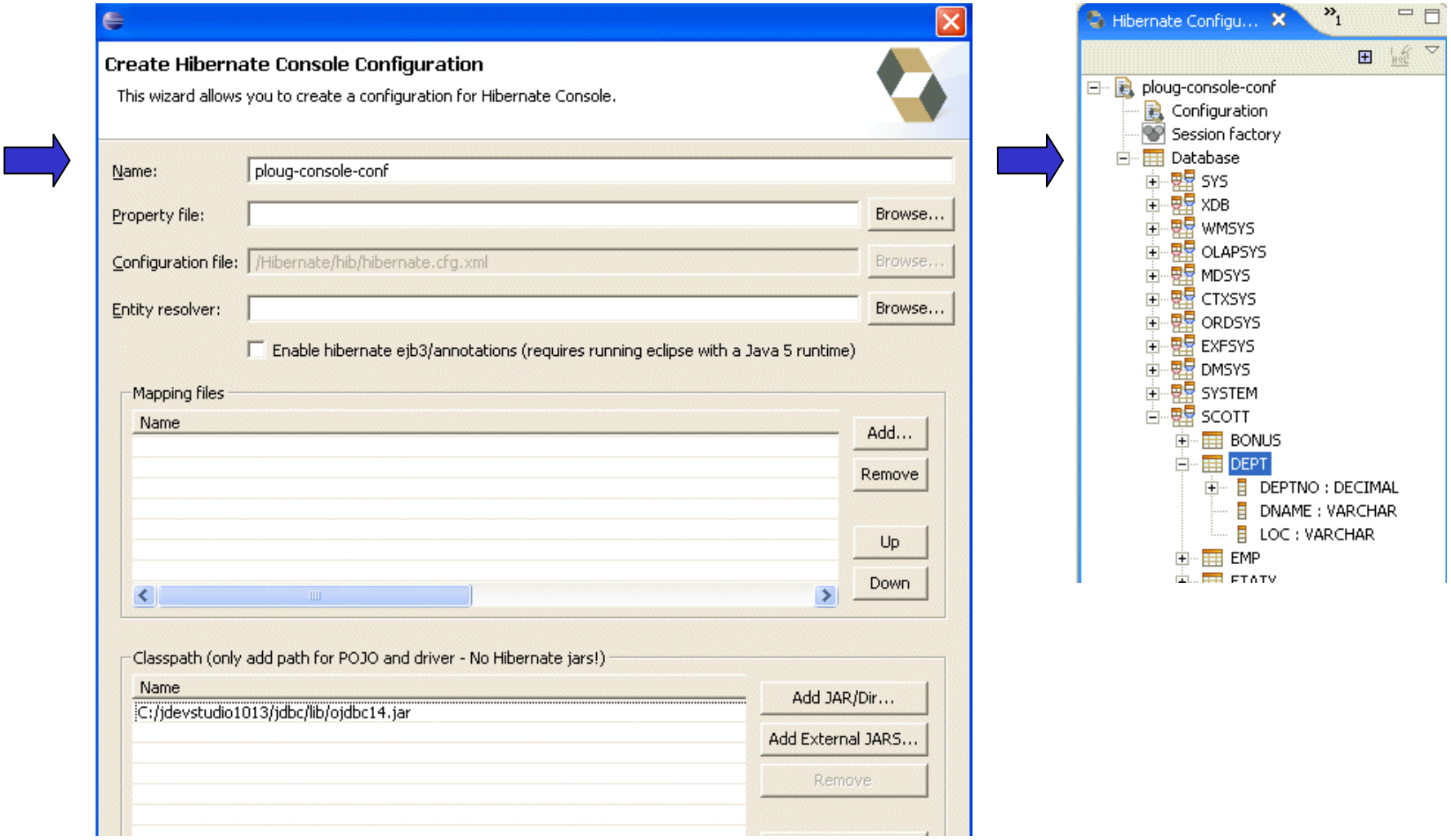
- Zbiór narzędzi dla Hibernate
  - Mają na celu ułatwienie korzystania z Hibernate nie ukrywając jego funkcjonalności
- Dostępne w postaci:
  - Zadań Anta
  - Wtyczki dla Eclipse (automatycznie dostępna w JBoss Eclipse IDE)
- Funkcjonalność:
  - Reverse engineering
  - Generacja kodu
  - Wizualizacja
  - Interakcja z Hibernate
- Wtyczka Hibernate Tools dla Eclipse oferuje:
  - Edytor odwzorowań (uzupełnianie i kolorowanie kodu)
  - Konsola Hibernate (przegląd konfiguracji, klas, itp. oraz interaktywne zapytania HQL
  - Kreatorzy i generatory kodu (w tym kompletny reverse engineering istniejącego schematu bazy danych)

Hibernate Tools: Konfiguracja połączenia z bazą danych (1/2)

- Połączenie z bazą danych konfiguruje się poprzez utworzenie odpowiedniego pliku hibernate.xml.cfg, a następnie tzw. konfiguracji konsoli (console configuration)

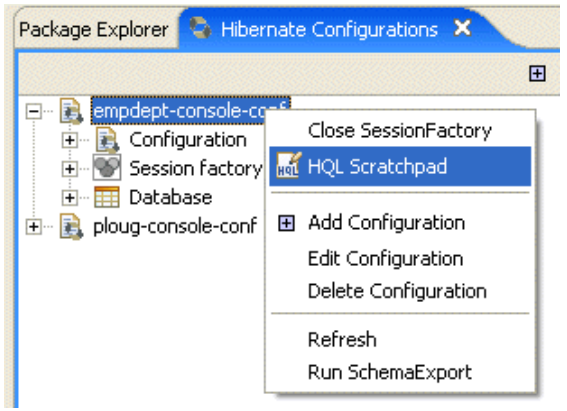


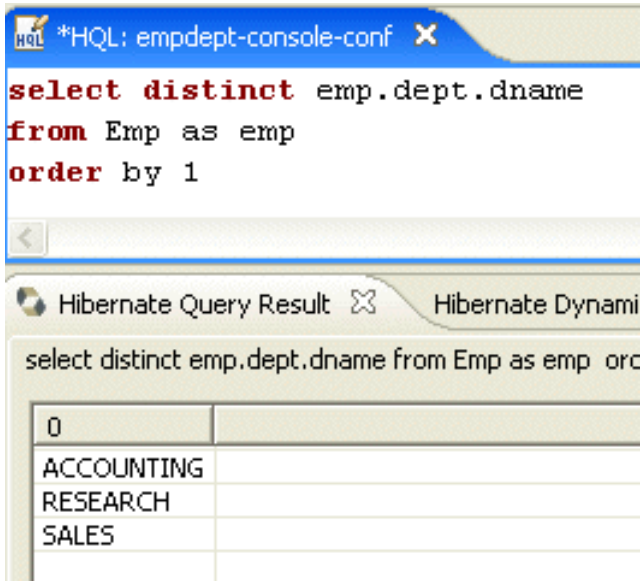
Hibernate Tools: Konfiguracja połączenia z bazą danych (2/2)

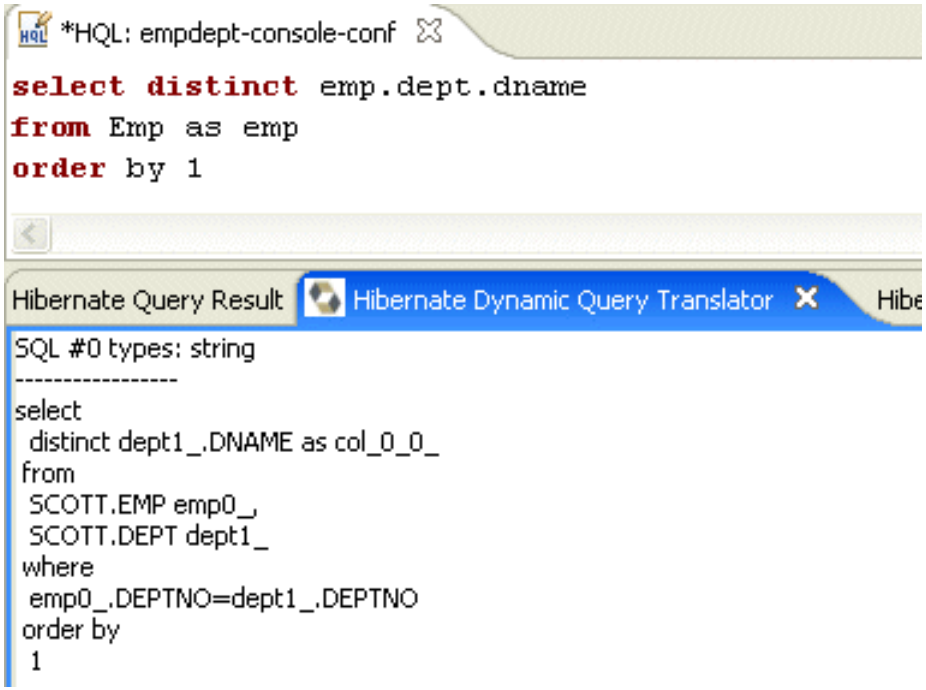


Hibernate Tools: HQL Scratchpad (1/2)

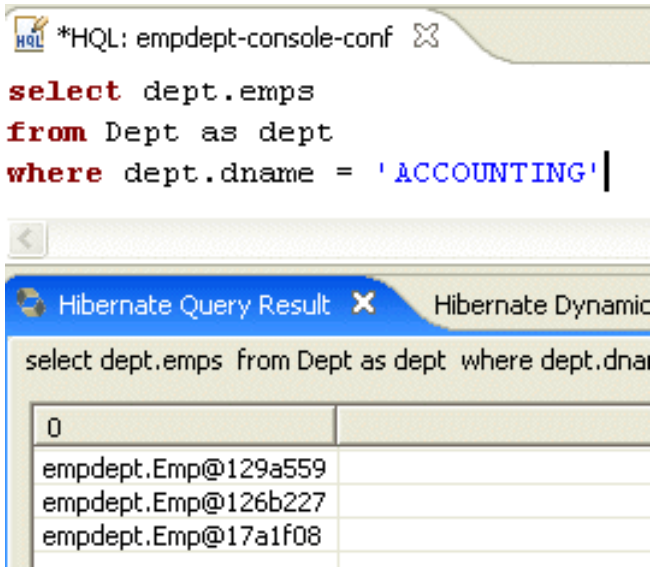
- Umożliwia wykonywanie zapytań ad-hoc w HQL oraz podgląd wyniku ich translacji do SQL

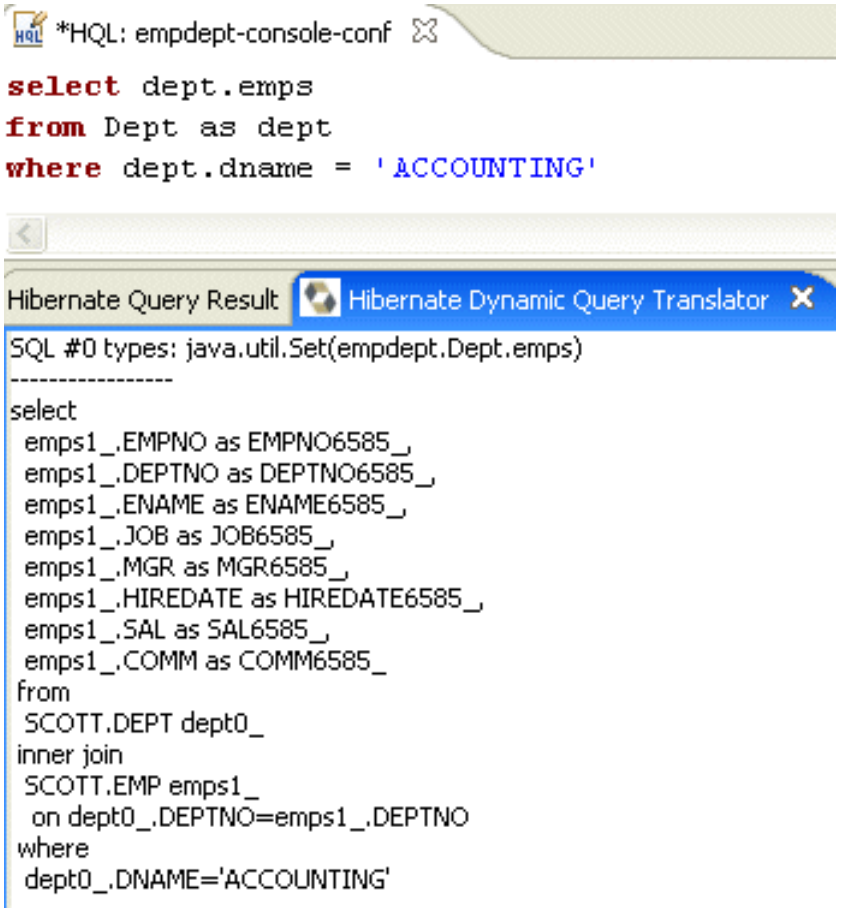






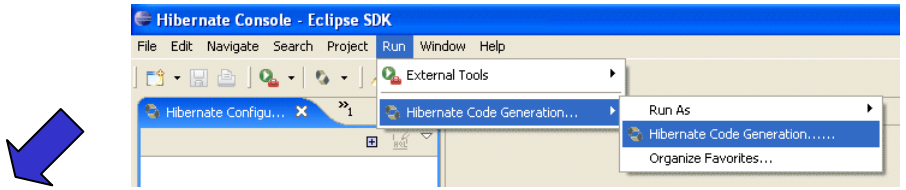
Hibernate Tools: HQL Scratchpad (2/2)

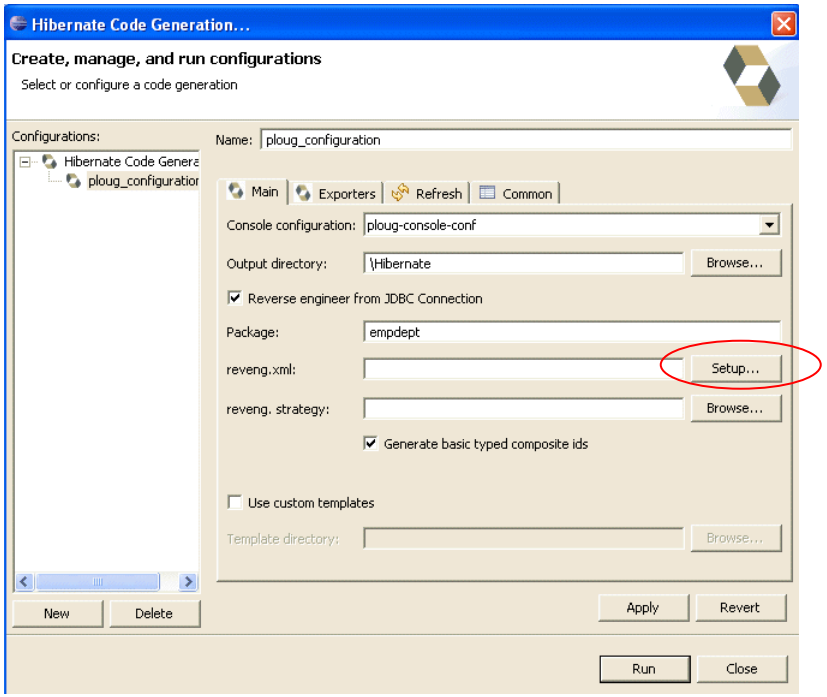




Hibernate Tools: Reverse Engineering (1/2)

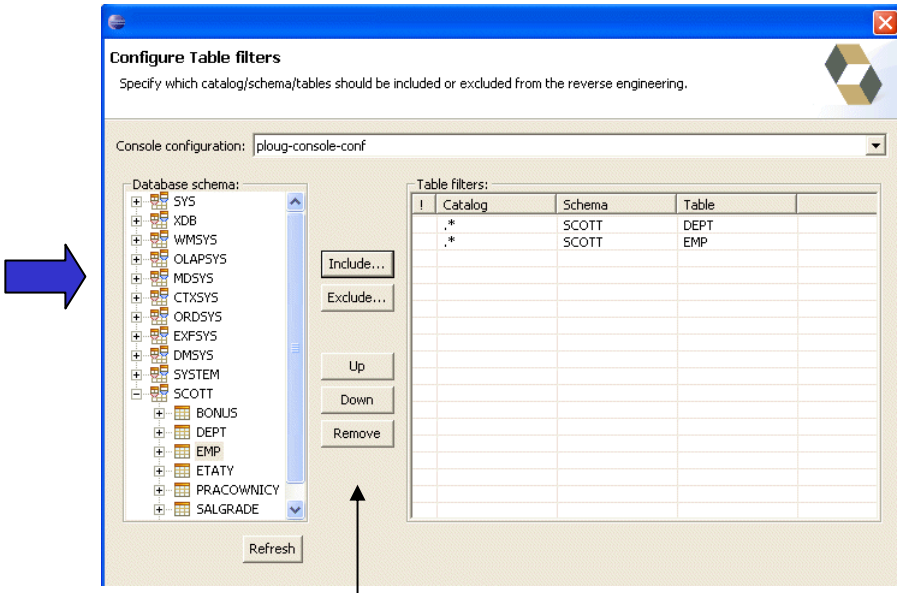
- Generacja klas POJO, plików .hbm.xml i pliku konfiguracyjnego hibernate.cfg.xml na podstawie istniejącego schematu bazy danych



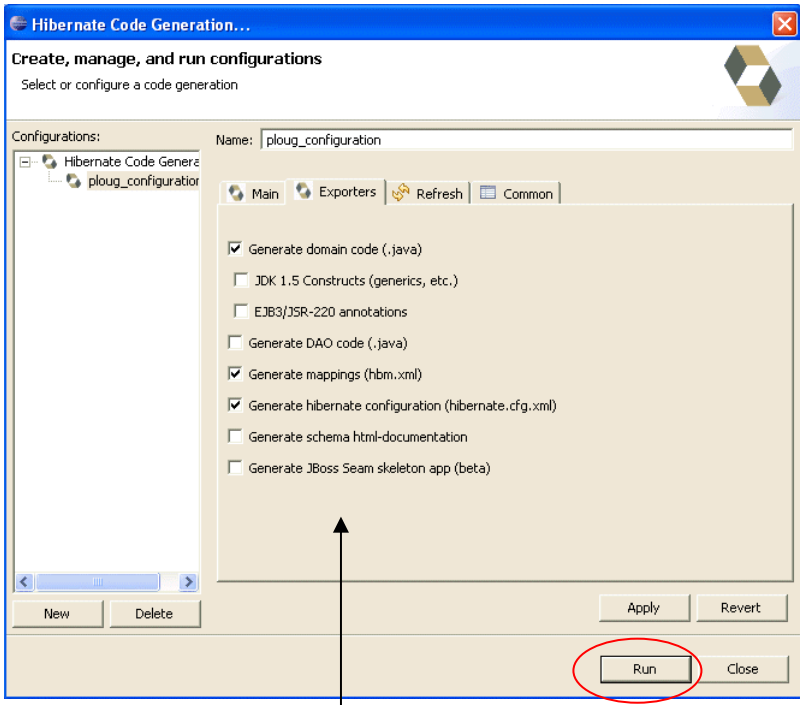


Przejście do definiowania pliku konfiguracyjnego zawierającego m.in. wskazanie schematu i wybór tabel

Hibernate Tools: Reverse Engineering (2/2)



Wskazanie schematu
i wybór tabel



Wskazanie co ma zostać
wygenerowane