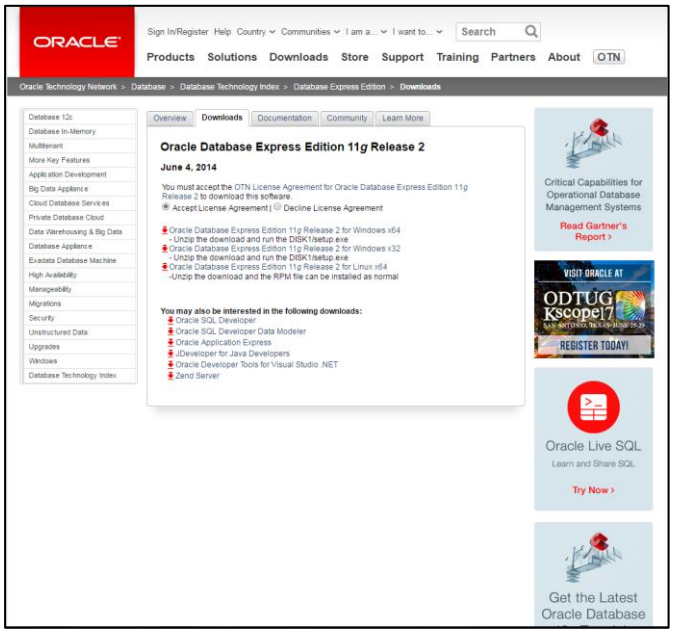
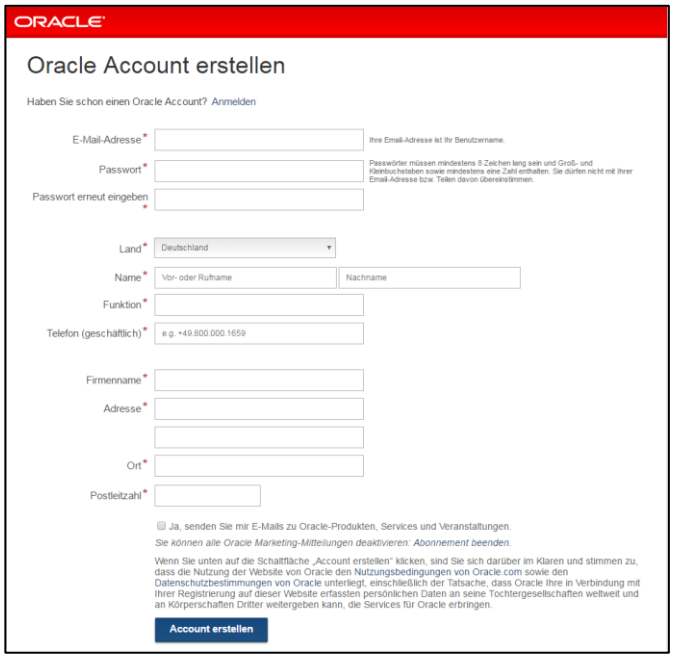
Java Hibernate Tutorial - How to establish a connection to Oracle DB?

In diesem Tutorial wird mithilfe eines minimalen Beispielprogramms gezeigt, wie man mit Java Hibernate auf eine Oracle Datenbank zugreifen kann.

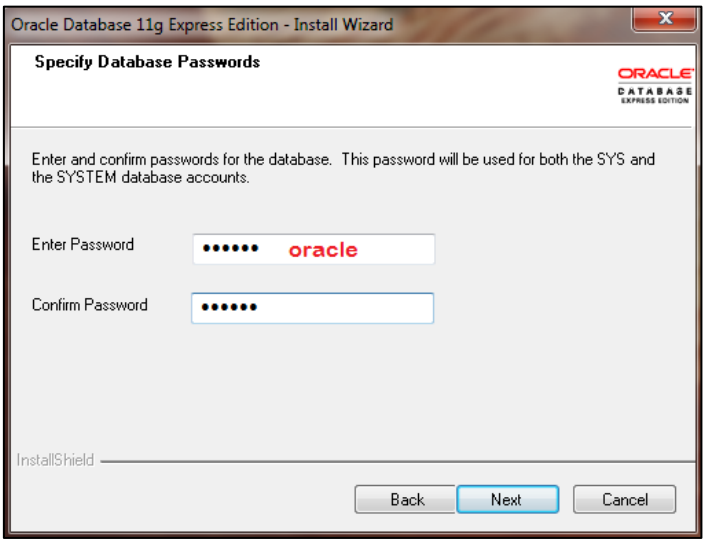
1) Hibernate-Distribution („hibernate-distribution-3.5.6-Final.zip“) downloaden & entpacken
   https://sourceforge.net/projects/hibernate/files/hibernate3/3.5.6-Final/

2) Download der „Oracle Database 11g Express Edition“ unter
   http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html
   (Für den Download muss ein Oracle-Account angelegt werden!)



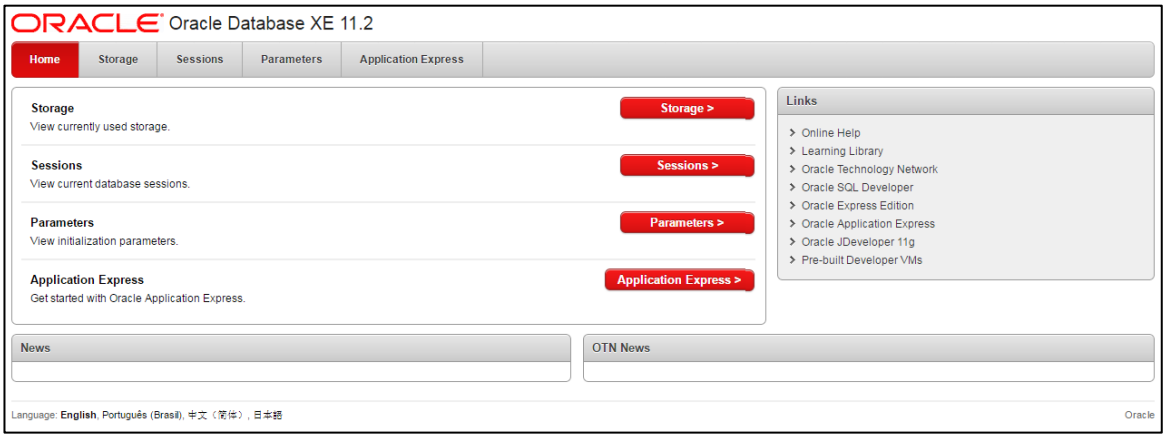


3) Installation der „Oracle Database 11g Express Edition“
   - ⇨ In der heruntergeladenen ZIP-Datei „OracleXE112_Win64.zip“ im Unterordner „DISK1“ die „setup.exe“ ausführen.
   - ⇨ Während der Installation muss ein Passwort für die Oracle DB gesetzt werden, welches später zum Login auf der Oracle-Startseite benötigt wird.

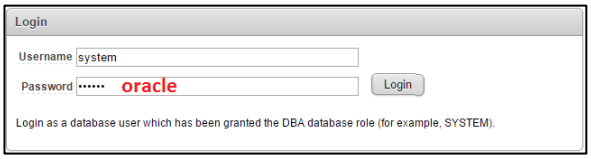


4) Nach der Installation:

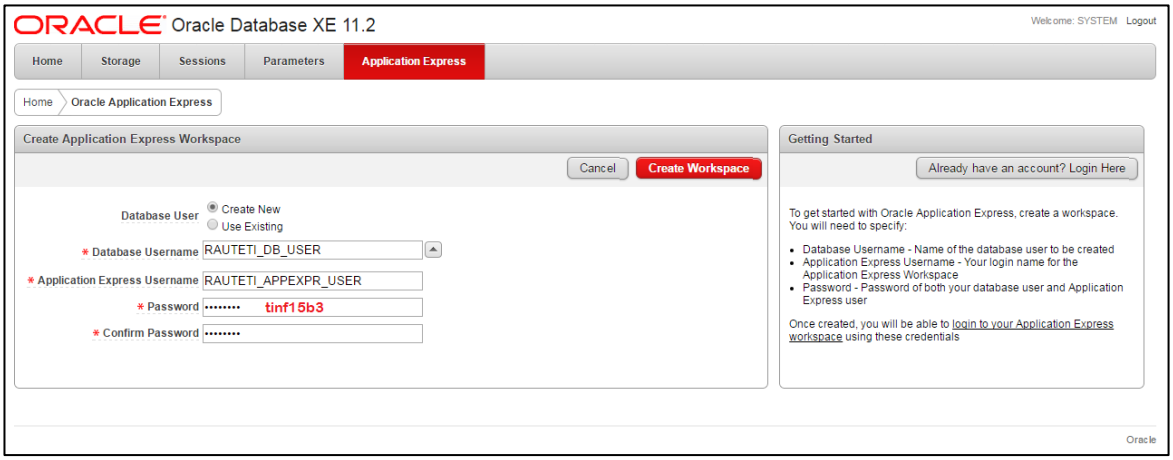
⇨ Aufruf der Verknüpfung „Get Started With Oracle Database 11g Express Edition“, um auf die Home-Seite der Oracle DB zu gelangen



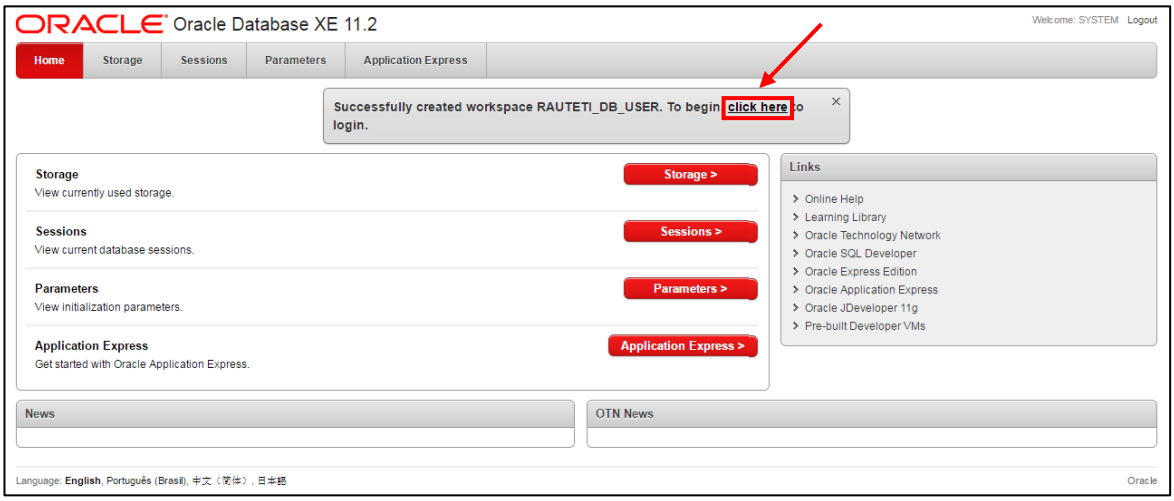
⇨ GOTO „Application Express“ in der Navigationsleiste:
Eingabe der Login-Daten notwendig, die während der Oracle-Installation gesetzt wurden.
Username: system
Password: oracle (Passwort, das bei der Installation gesetzt wurde)

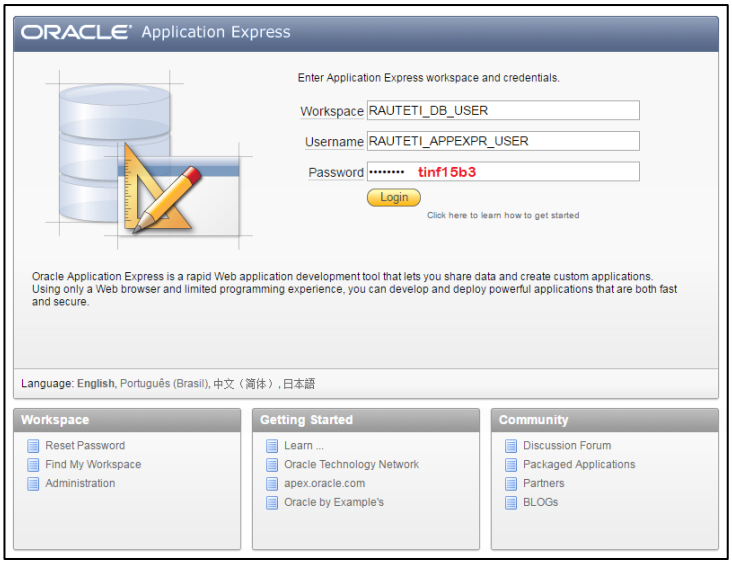


⇨ Anlegen eines neuen „Application Express Workspace“:
Database Username: RAUTETI_DB_USER
Application Express Username: RAUTETI_APPEXPR_USER
Password: tinf15b3



=> anschließend mit den Login-Daten in den „Application Express Workspace“ einloggen.

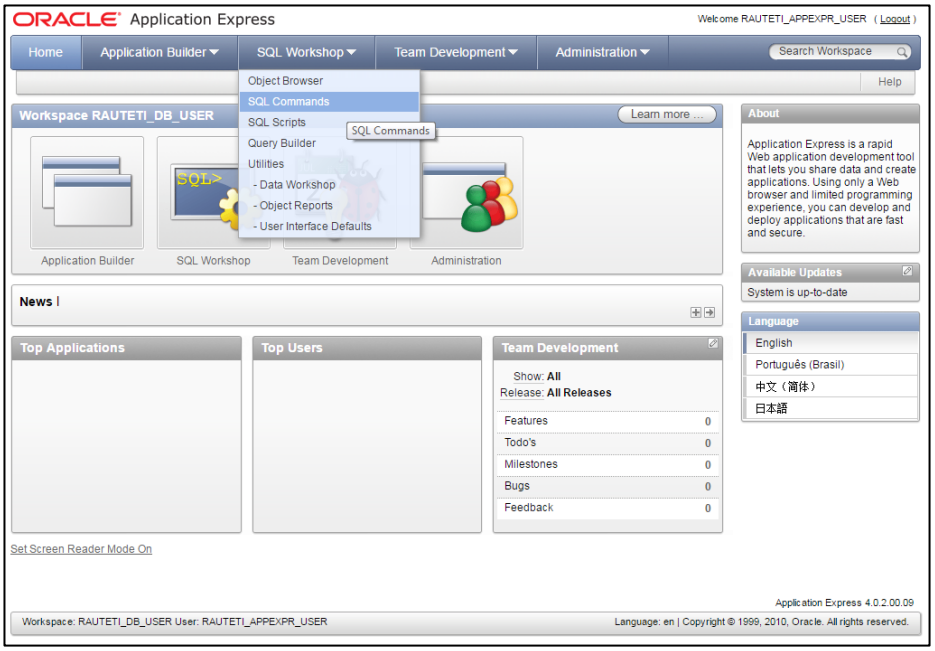


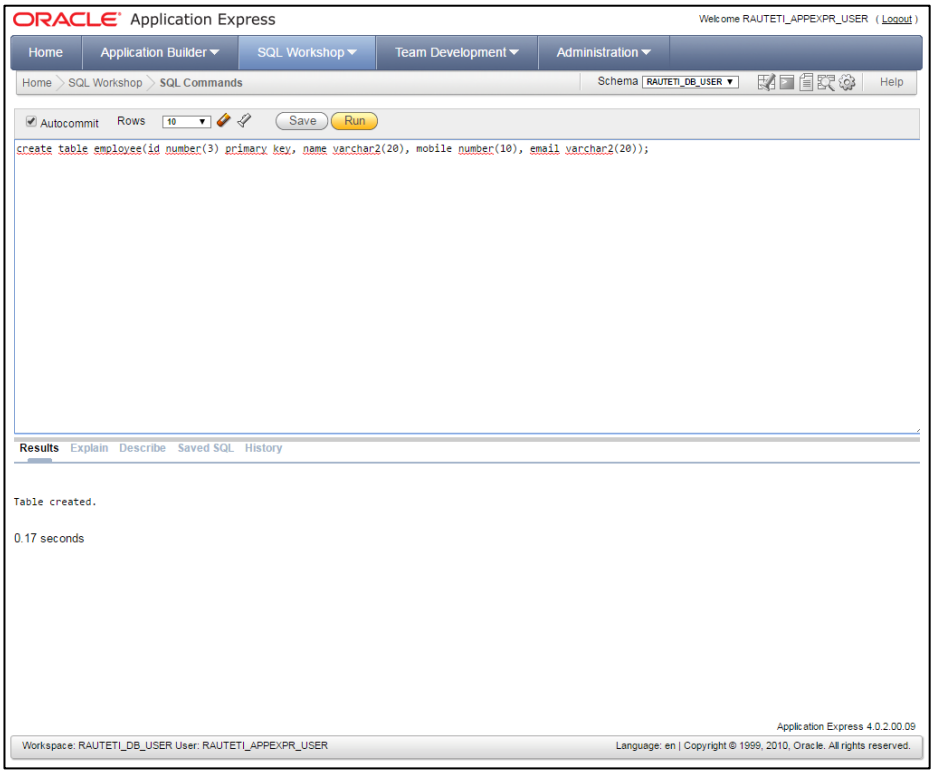


⇨ Navigationsleiste: SQL Workshop --> SQL Commands

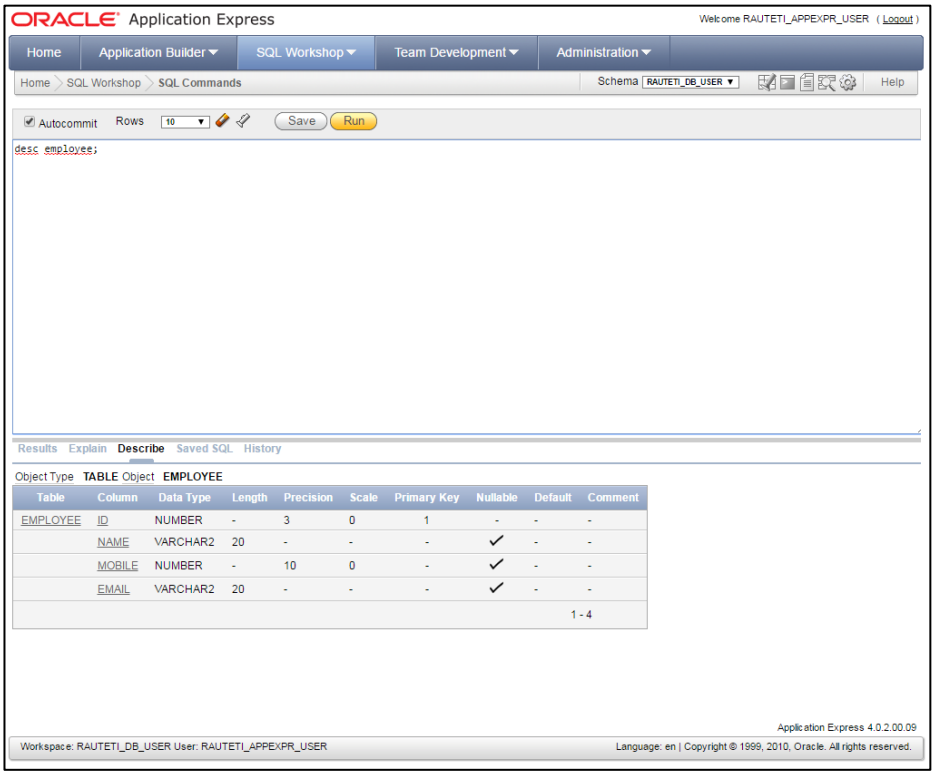
```
create table employee(id number(3) primary key, name varchar2(20),
mobile number(10), email varchar2(20));
```

=> RUN





⇨ Möglichkeit zur Überprüfung der angelegten Tabelle (Beschreibung der Tabelle wird angezeigt):
`desc employee;`



5) Erstellen eines neuen Java Projektes (in diesem Tutorial wurde als IDE Eclipse verwendet)
   ⇨ Für den Zugriff auf die Oracle Datenbank aus der Java Anwendung heraus müssen innerhalb des „src“-Ordners des Projektes vier Dateien erstellt und dementsprechend implementiert werden. Nachfolgend wird der Codeinhalt der zu implementierenden Dateien zuerst ausführlich beschrieben und anschließend nochmals in einer Zusammenfassung aufgeführt.

"Hibernate.cfg.xml" – File:

Erstellen eines XML-Files (Namenskonvention "... .cfg.xml") mit folgenden Einstellungen:

- DOCTYPE-Zeile aus dem File
  „hibernate-distribution-3.5.6-Final\project\core\src\main\resources\org\hibernate\hibernate-configuration-3.0.dtd“
  in das erstellte "... .cfg.xml"-File kopieren

- Folgende Zeilen als Grundgerüst in das "... .cfg.xml"-File einfügen, um darin die Einstellungen (properties) für den Verbindungsaufbau zur Oracle Database zu setzen:

```
<hibernate-configuration>
  <session-factory>
       <property name=""></property>
       ...
  </session-factory>
</hibernate-configuration>
```

- Das Setzen der Parameter-Einstellungen in den Property-Tags wird nun mithilfe folgendes Files durchgeführt:
  „hibernate-distribution-3.5.6-Final\project\etc\hibernate.properties“
  Je nach verwendeter Datenbank-Schnittstelle müssen in den Property-Tags `<property name="?">?</property>` unterschiedliche Parameter bzw. Werte gesetzt werden.
  Für eine Verbindung zu einer Oracle Database sind folgende Zeilen notwendig:

```
<property name="hibernate.connection.driver_class">
  oracle.jdbc.driver.OracleDriver
</property>

<property name="hibernate.connection.url">
  jdbc:oracle:thin:@localhost:1521:XE
</property>

<property name="hibernate.dialect">
  org.hibernate.dialect.Oracle10gDialect
</property>

<property name="hibernate.connection.username">?</property>

<property name="hibernate.connection.password">?</property>
```

- Autocommit abschalten:
  --> Sorgt dafür, dass Änderungen am Datenbestand während einer Sitzung normalerweise nicht direkt im Datenbestand vorgenommen, sondern gesammelt und erst am Ende der sogenannten Transaktion durch ein commit() weggeschrieben oder ein rollback() rückgängig gemacht.

```
<property name="hibernate.connection.autocommit">false</property>
```

- Konsolenausgabe für die Anzeige von SQL-Querys einschalten:

```
<property name="show_sql">true</property>
<property name="format_sql">true</property>
<property name="use_sql_comments">true</property>
```

- Einstellungen für die JDBC-Connection:

```
<property name="hibernate.transaction.factory_class">
  org.hibernate.transaction.JDBCTransactionFactory
</property>
```

- Angabe des Dateinamens des Mapping-Files:
  --> Angabe wie die Mapping-Datei "... .hbm.xml" heißt, in welcher das Mapping von einer Klasse auf die DB-Tabelle stattfindet

```
<mapping resource="EmployeeMapping.hbm.xml"/>
```

Hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>

<!-- connection to the database -->
<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>
<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
<property name="hibernate.connection.username">?</property>
<property name="hibernate.connection.password">?</property>

<!-- autocommit false -->
<property name="hibernate.connection.autocommit">false</property>

<!-- to display sql query generated when running the program -->
<property name="show_sql">true</property>
<property name="format_sql">true</property>
<property name="use_sql_comments">true</property>

<!-- for jdbc transaction -->
<property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</p
roperty>

<!-- mapping file -->
<mapping resource="EmployeeMapping.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

"Employee.java" – File:

Erstellen einer normalen POJO-Klasse (Plain Old Java Objects) mit einem Aufbau wie die Tabelle der Oracle-Datenbank:

- Die Klasse „Employee“ muss dieselbe Attributs-Struktur aufweisen wie die Spalten der Oracle Datenbank-Tabelle „EMPLOYEE“, sodass im nächsten Schritt ein sogenanntes Mapping durchgeführt werden kann.

- Die Klasse muss das Interface „Serializable“ implementieren, damit man mit mehreren Threads/Sessions, wie in DB-Transaktionen üblich, auf das Objekt zugreifen kann.

- Für jedes Attribut der Klasse „Employee“ muss eine get- & set-Methode implementiert werden.

Employee.java

```
import java.io.Serializable;

public class Employee implements Serializable {

	// Attributes
	private int id;
	private String name;
	private long mobile;
	private String email;

	// Constructor
	public Employee() {

	}

	// Methods
	public int getId() { return id; }
	public void setId(int id) { this.id = id; }
	public String getName() { return name; }
	public void setName(String name) { this.name = name; }
	public long getMobile() { return mobile; }
	public void setMobile(long mobile) { this.mobile = mobile; }
	public String getEmail() { return email; }
	public void setEmail(String email) { this.email = email; }
}
```

"EmployeeMapping.hbm.xml" – File:

Erstellen eines weiteren XML-Files (Namenskonvention "... .hbm.xml") mit folgenden Einstellungen:

- DOCTYPE-Zeile aus dem File
  „hibernate-distribution-3.5.6-Final\project\core\src\main\resources\org\hibernate\
  hibernate-mapping-3.0.dtd“
  in das erstellte "... .cfg.xml"-File kopieren

- Folgende Zeilen als Grundgerüst in das "... .hbm.xml"-File einfügen. Innerhalb dieser Tags muss nun das Mapping von einer Java Klasse auf die Oracle Datenbank-Tabelle definiert werden.

```
<hibernate-mapping>
...
</hibernate-mapping>
```

- Für jede Klasse, die auf eine Datenbank-Tabelle gemappt werden soll, muss nun ein neuer `<class>`–Tag eingefügt werden:

```
<class name="Employee" table="EMPLOYEE">
  <id name="id" column="ID" type="integer">
       <generator class="assigned"></generator>
  </id>
  <property name="name" column="NAME" type="string"></property>
  <property name="mobile" column="MOBILE" type="long"></property>
  <property name="email" column="EMAIL" type="string"></property>
</class>
```

  Unter dem `type` – Parameter müssen fest definierte „Hibernate Mapping Types“ angegeben werden, sodass die Java-Datentypen in RDBMS-Datentypen übersetzt werden können.
  Eine Zusammenstellung dieser „Hibernate Mapping Types“ findet man unter folgendem Link:
  https://www.tutorialspoint.com/hibernate/hibernate_mapping_types.htm

  Zur klaren Unterscheidung von Java-Attributen und Datenbank-Tabellenspalten wurden die Datenbank-Tabellenspalten-Namen zusätzlich großgeschrieben, da sie ohnehin key-insensitive sind.

- Der Dateiname dieses Hibernate-Mapping-Files muss nun unter dem XML-Tag
  `<mapping resource="EmployeeMapping.hbm.xml"/>`
  im „Hibernate.cfg.xml“-File eingetragen werden.

EmployeeMapping.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="Employee" table="EMPLOYEE">
       <id name="id" column="ID" type="integer">
              <generator class="assigned"></generator>
       </id>
       <property name="name" column="NAME" type="string"></property>
       <property name="mobile" column="MOBILE" type="long"></property>
       <property name="email" column="EMAIL" type="string"></property>
</class>

</hibernate-mapping>
```

"DatabaseAccess.java " – File:

Erstellen einer Klasse mit main-Methode, in der letztendlich die komplette Logik für den Datenbank-Zugriff erfolgt:

- Implementierung folgender Codezeilen zur Herstellung einer Verbindung & Durchführung von Transaktionen mit der Oracle-Datenbank:

```
Configuration cfg = new Configuration();
cfg.configure("Hibernate.cfg.xml");
SessionFactory sf = cfg.buildSessionFactory();
Session s = sf.openSession();
Transaction tx = s.beginTransaction();
```

- Erzeugung eines Employee-Objektes:

```
Employee emp = new Employee();
emp.setId(1);
emp.setName("Timo");
emp.setMobile(1234567890);
emp.setEmail("abc@xyz.de");
```

- Mapping des Employee-Objektes auf die Oracle Datenbank:

```
s.save(emp);
s.flush();
tx.commit();
s.close();
```

DatabaseAccess.java

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class DatabaseAccess {

	public static void main(String[] args) {
		Configuration cfg = new Configuration();
		cfg.configure("Hibernate.cfg.xml");
		SessionFactory sf = cfg.buildSessionFactory();
		Session s = sf.openSession();
		Transaction tx = s.beginTransaction();

		Employee emp = new Employee();
		emp.setId(1);
		emp.setName("Timo");
		emp.setMobile(1234567890);
		emp.setEmail("abc@xyz.de");

		s.save(emp);
		s.flush();
		tx.commit();
		s.close();
	}
}
```

6) Einbinden der JAR-Bibliotheken der Hibernate-Distribution
--> Aus dem „hibernate-distribution-3.5.6-Final.zip“-Package müssen nun bestimmte *.jar – Files ins vorhandene Java Projekt importiert werden:

- antlr-2.7.6
- commons-collections-3.1
- commons-logging-1.0.4
- dom4j-1.6.1
- ehcache-1.2.3
- hibernate3
- hibernate-testing
- javassist-3.0.0.GA
- jta-1.1
- p6spy-1.3 // nicht unbedingt notwendig für dieses Beispielprogramm
- slf4j-api-1.5.8

Folgende JAR-Datei muss zusätzlich online heruntergeladen werden, da sie sich nicht im Hibernate-Package befindet:

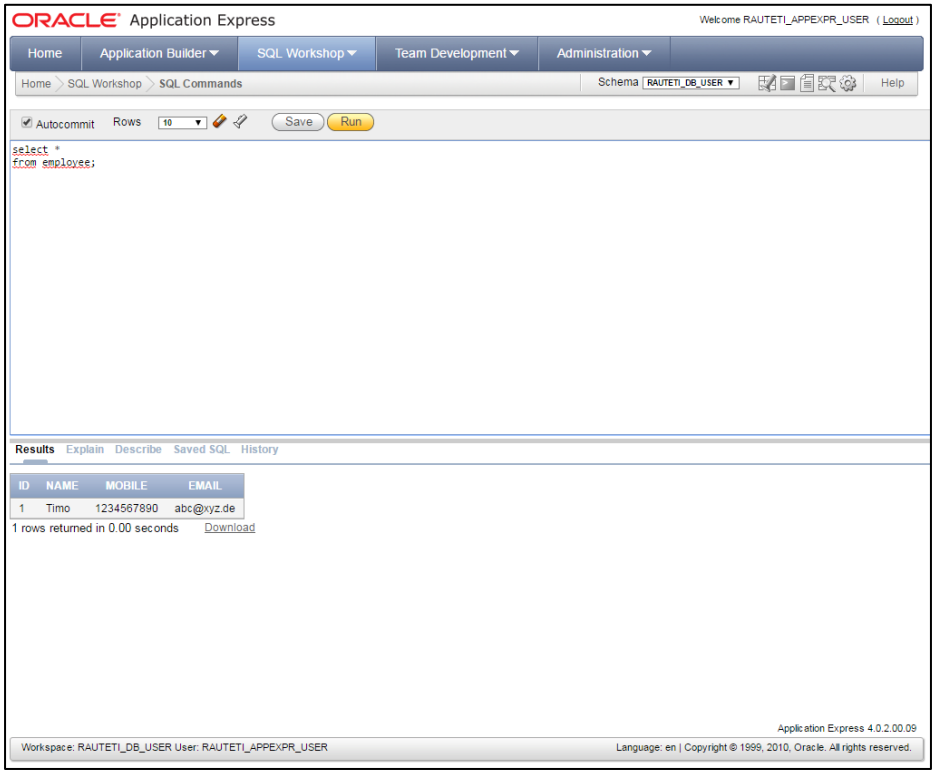
- slf4j-jdk14-1.5.8

Zusätzlich muss noch die JAR-Datei aus dem Oracle-Server-Paket hinzugefügt werden, welche sich unter dem Pfad „<Oracle-Installationsverzeichnis>\app\oracle\product\11.2.0\server\jdbc\lib“ befindet:

- ojdbc14_g

In der verwendeten Eclipse IDE importiert man diese unter:
Projekt-Properties --> Java Build Path --> Add Exteral JARs

7) Nach dem Ausführen der Java-Anwendung und anschließender Kontrolle der Oracle Datenbank wurde in der Datenbank-Tabelle ein Tupel der Employee-Tabelle erstellt.



Grundlegende Datenbank-Transaktionen:

In den bisherigen Schritten wurde fast ausschließlich auf das Herstellen einer Verbindung zur Datenbank eingegangen und zum Test wurde ein erster Datensatz (Tupel) in die Datenbank eingetragen.
Hibernate besitzt jedoch viele weitere Funktionen. Deshalb hier eine kurze Auflistung der Implementierung von den grundlegendsten Datenbank-Transaktionen:

Datenbank-Transaktion	**Code-Beispiel**

insert	`Configuration cfg = new Configuration();`
	`cfg.configure("Hibernate.cfg.xml");`
	`SessionFactory sf = cfg.buildSessionFactory();`
	`Session s = sf.openSession();`
	`Transaction tx = s.beginTransaction();`
	
	`Employee emp = new Employee();`
	`emp.setId(1);`
	`emp.setName("Timo");`
	`emp.setMobile(1234567890);`
	`emp.setEmail("abc@xyz.de");`
	
	`s.save(emp);`
	`s.flush();`
	`tx.commit();`
	`s.close();`

select	`Configuration cfg = new Configuration();`
	`cfg.configure("Hibernate.cfg.xml");`
	`SessionFactory sf = cfg.buildSessionFactory();`
	`Session s = sf.openSession();`
	`Transaction tx = s.beginTransaction();`
	
	`// Tabellenzeile muss existieren`
	`Employee emp = (Employee) s.load(Employee.class, new Integer(1));`
	`oder`
	`// wenn Tabellenzeile nicht vorhanden ist, wird ein NULL-Pointer zurückgegeben`
	`Employee emp = (Employee) s.get(Employee.class, new Integer(1));`
	
	`System.out.println(emp.getId());`
	`System.out.println(emp.getName());`
	`System.out.println(emp.getMobile());`
	`System.out.println(emp.getEmail());`
	`s.close();`

update	`Configuration cfg = new Configuration();`
	`cfg.configure("Hibernate.cfg.xml");`
	`SessionFactory sf = cfg.buildSessionFactory();`
	`Session s = sf.openSession();`
	`Transaction tx = s.beginTransaction();`
	
	`Employee emp = (Employee) s.get(Employee.class, new Integer(1));`
	
	`emp.setId(1);`
	`emp.setName("Daniel");`
	`emp.setMobile(2468);`
	`emp.setEmail("daniel@gmx.de");`
	
	`s.update(emp);` `// Tabellenzeile muss existieren`
	`oder`
	`s.saveOrUpdate(emp);` `// wenn Tabellenzeile nicht vorhanden ist`
	`// wird ein neuer Datensatz angelegt`
	
	`s.flush();`
	`tx.commit();`
	`s.close();`

delete	`Configuration cfg = new Configuration();`
	`cfg.configure("Hibernate.cfg.xml");`
	`SessionFactory sf = cfg.buildSessionFactory();`
	`Session s = sf.openSession();`
	`Transaction tx = s.beginTransaction();`
	
	`Employee emp = (Employee) s.get(Employee.class, new Integer(1));`
	
	`s.delete(emp);`
	
	`s.flush();`
	`tx.commit();`
	`s.close();`

create table	Hinzufügen eines Property-Tags im Hibernate-Mapping-File:
	
	`<!-- Tabelle löschen & neu anlegen -->`
	`<property name="hbm2ddl.auto">create</property>`
	
	`<!-- Tabelle nur neu anlegen falls nicht vorhanden -->`
	`<property name="hbm2ddl.auto">update</property>`