1) Primary Key

a) HSQL AUTO Strategy => IDENTITY

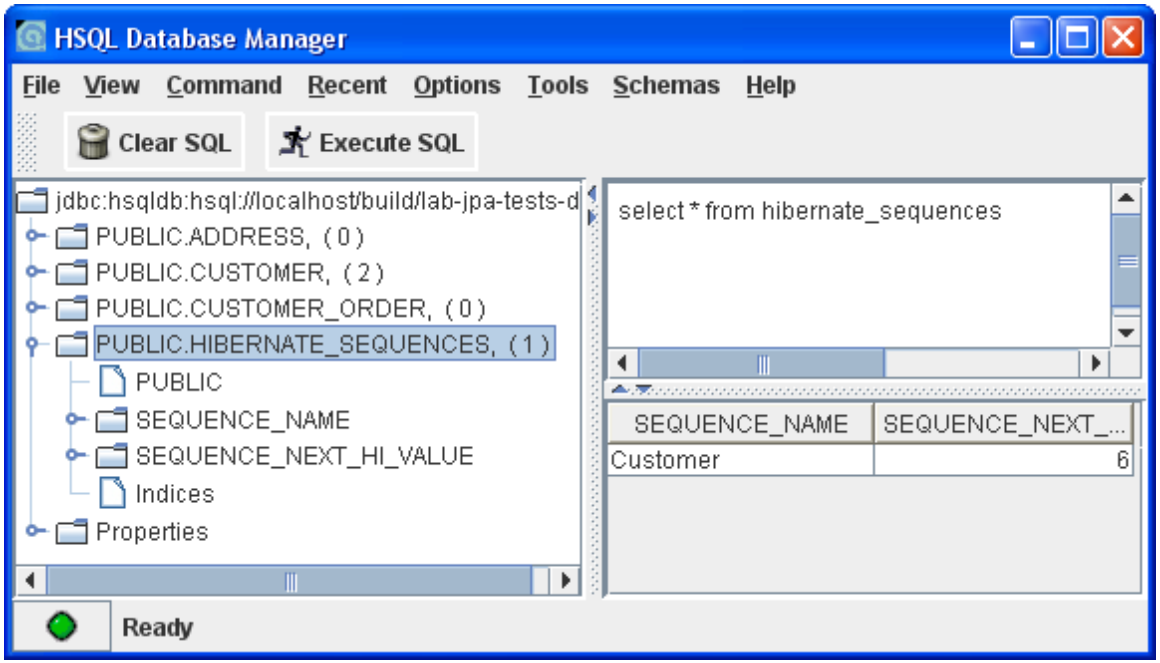
- lab-jpa\target\databases\lab-jpa-db.script

```
CREATE MEMORY TABLE CUSTOMER(
   ID INTEGER
      GENERATED BY DEFAULT AS IDENTITY(START WITH 1)
      NOT NULL
      PRIMARY KEY,
   AGE INTEGER
      NOT NULL,
   NAME VARCHAR(255),
   ADDRESS_ID INTEGER,
   CONSTRAINT FK27FBE3FEAAEE95A3
      FOREIGN KEY(ADDRESS_ID) REFERENCES ADDRESS(ID))
```

1) Primary Key

b) TableGeneratedValue(strategy=GenerationType.TABLE)

- PKs: 32768 * SEQUENCE_NEXT_HI_VALUE



1) Primary Key

c) Table Generator Annotation

```
@TableGenerator(
   name="generator",                    // name of the generator
   table="ID_GEN",                      // table name
   pkColumnName="GEN_KEY",              // name of key column
   valueColumnName="GEN_VALUE",         // name of value column
   pkColumnValue="CUSTOMER_ID",         // key entry
   allocationSize=10)                   // size of block
@Id
@GeneratedValue(
        strategy=GenerationType.TABLE, generator="generator")
```

1) Primary Key

- **Performance comparison**
  - 10'000 insert statements

  - AUTO 7534 msec
  - TABLE (allocationSize = 32768) 2244 msec
  - TABLE (allocationSize = 1) 9612 msec
  - TABLE (allocationSize = 2) 7429 msec
  - TABLE (allocationSize = 4) 5856 msec
  - ASSIGNED (user defined) 1959 msec

2) EntityManager Cache

- **Same Entity Manager**

```
Customer c1 = em.find(Customer.class, 1);
Customer c2 = em.find(Customer.class, 1);
```

  - Identical references due to entity cache

- **Different Entity Managers**

```
Customer c1 = emf.createEntityManager().find(Customer.class,1);
Customer c2 = emf.createEntityManager().find(Customer.class,1);
```

  - Different references/instances

3) Lazy Loading

```
@Entity
public class Customer {
   @Id
   private int id;

   @OneToOne
   private Address address;
   ...
```

```
Customer c = em.find(Customer.class, 1);
System.out.println(c.getAddress().getClass().getName());

ch.fhnw.edu.model.Address
```

3) Lazy Loading

```
@Entity
public class Customer {
   @Id
   private int id;

   @OneToOne(fetch=FetchType.LAZY)
   private Address address;
   ...
```

```
Customer c = em.find(Customer.class, 1);
System.out.println(c.getAddress().getClass().getName());

ch.fhnw.edu.model.Address_$$_javassist_2
```

=> Address is a proxy which knows how to load the data

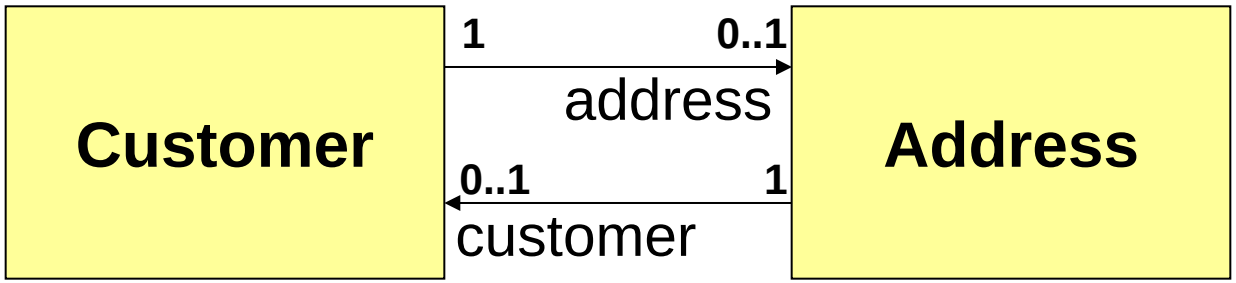
3) Lazy Loading

- **Byte-Code manipulation engine can be specified**
  - hibernate.bytecode.provider = javaassist [default]
    - ch.fhnw.edu.model.Address_$$_javassist_2
  - hibernate.bytecode.provider = cglib [deprecated]
    - ch.fhnw.edu.model.Address$$EnhancerByCGLIB$$8cbef091

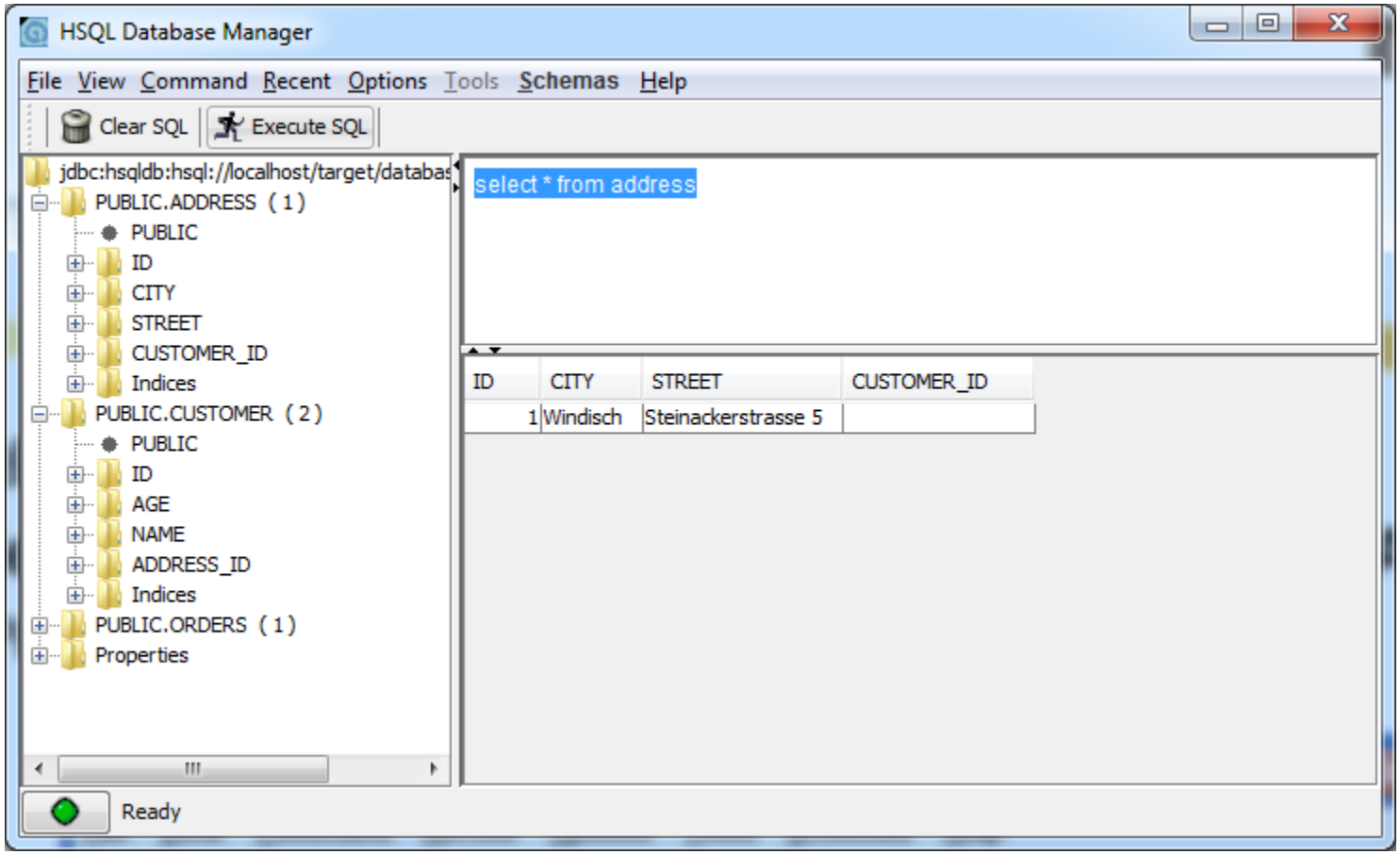
- **Specification**
  - persistence.xml
    - <property name=*"hibernate.bytecode.provider" value="cglib"*/>
  - hibernate.properties [overrides definitions in persistence.xml]
    - hibernate.bytecode.provider=javassist

4) OneToOne

```
@Entity                                  @Entity
public class Customer {                  public class Address {
   @Id                                      @Id
   private int id;                          private int id;

   @OneToOne                                @OneToOne
   private Address address;                 private Customer customer;
   ...                                      ...
```



4) OneToOne



5) Bidirectional OneToOne

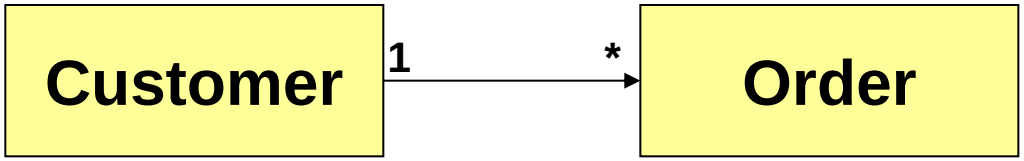
```
Customer c = new Customer("Gosling", 44);
Address a = new Address("Infinite Loop 1", "Cupertino");
c.setAddress(a);

em.persist(a); // necessary ???
em.persist(c);
```

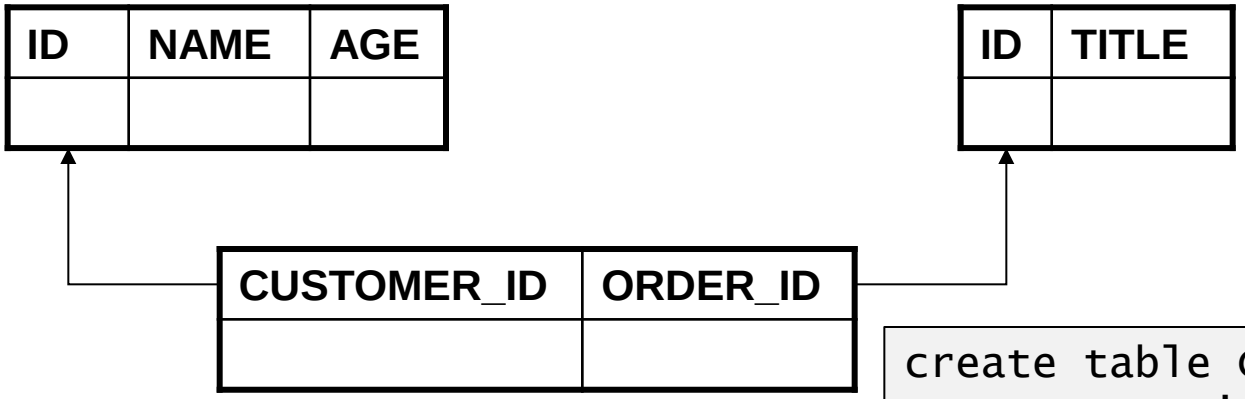
- **em.persist(a)**
  - not necessary if *cascade=CascadeType.PERSIST*
  - Otherwise, if ***a*** is not persisted, an exception is thrown

```
object references an unsaved transient instance - save the
transient instance before flushing:
ch.fhnw.edu.model.Customer.address -> ch.fhnw.edu.model.Address
```

6) Unidirectional OneToMany

```
@OneToMany
private List<Order> orders = new ArrayList<Order>();
```

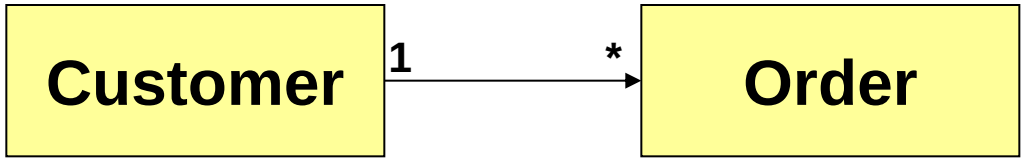


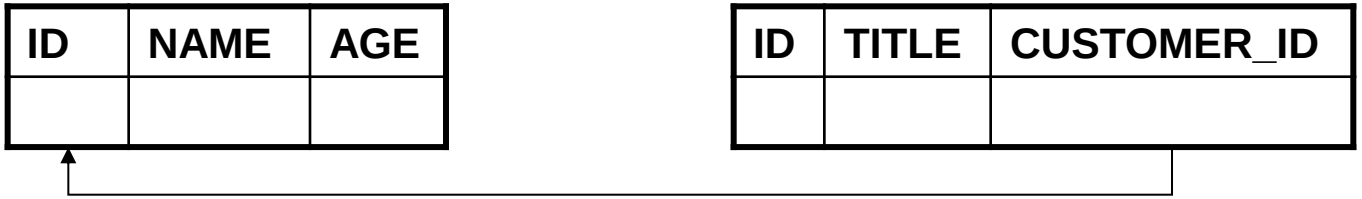



```
create table CUSTOMER_ORDERS (
   Customer_id integer not null,
   orders_id integer not null,
   unique (orders_id))
```

6) Unidirectional OneToMany

```
@OneToMany
@JoinColumn(name="CUSTOMER_ID")
private List<Order> orders = new ArrayList<Order>();
```





6) Unidirectional OneToMany

- **Inconsistent Model: what happens?**

```
c1.addOrder(o1);
c1.addOrder(o2);

c2.addOrder(o1);
c2.addOrder(o3);
```

  - Intermediate Table:
    - Exception in thread "main" java.sql.SQLException: Integrity constraint violation FKE0BB9646C73D8EAC table: CUSTOMER_ORDERS
  - Foreign Key:
    - Last Insert wins

7) Flush Mode

```
Customer c = em.find(Customer.class, 1);
c.getAddress().setCity("Basel");

Query q = em.createQuery("select a.city from Address a");
List<?> cities = q.getResultList();
for(Object city : cities)
        System.out.println(city);
```

- **em.setFlushMode(FlushModeType.*COMMIT*);**
  - Windisch
  - *Pending changes are synchronized with the database upon commit*
- **em.setFlushMode(FlushModeType.*AUTO*); [default]**
  - Basel
  - *All pending changes in persistence context are synchronized with database before a query is executed*

8) Persistence Context & Database

```
em.getTransaction().begin();
Customer c = em.find(Customer.class, 1);
c.getAddress().setCity("Zuerich");
// flush not necessary as default flush-mode = AUTO
Query q = em.createQuery("select a.city from Address a");
List<?> cities = q.getResultList();
for(Object city : cities)
        System.out.println(city);
System.in.read();
em.getTransaction().rollback();
```

```
Zürich


Wettingen
```

- **Synchronizing PersistenceContext with Database # COMMIT**
- **Whether uncommitted changes are visible depends on TX isolation level**