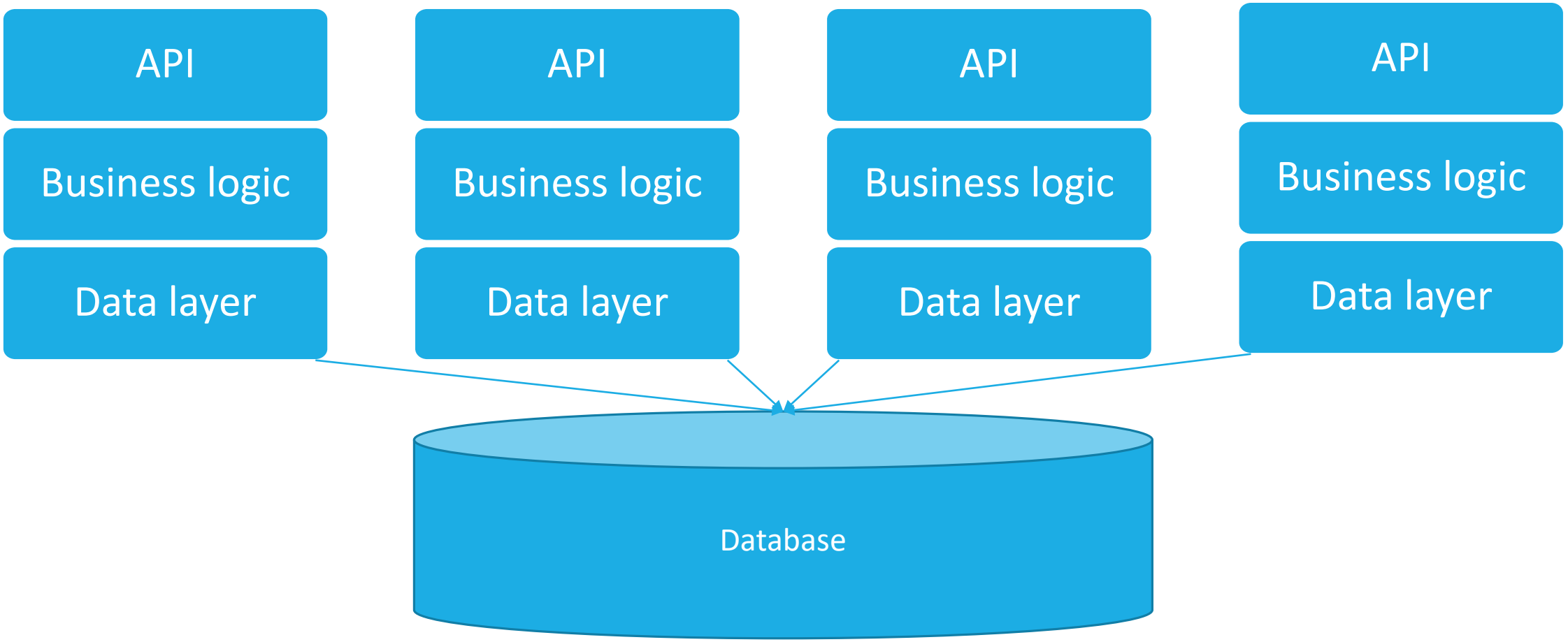
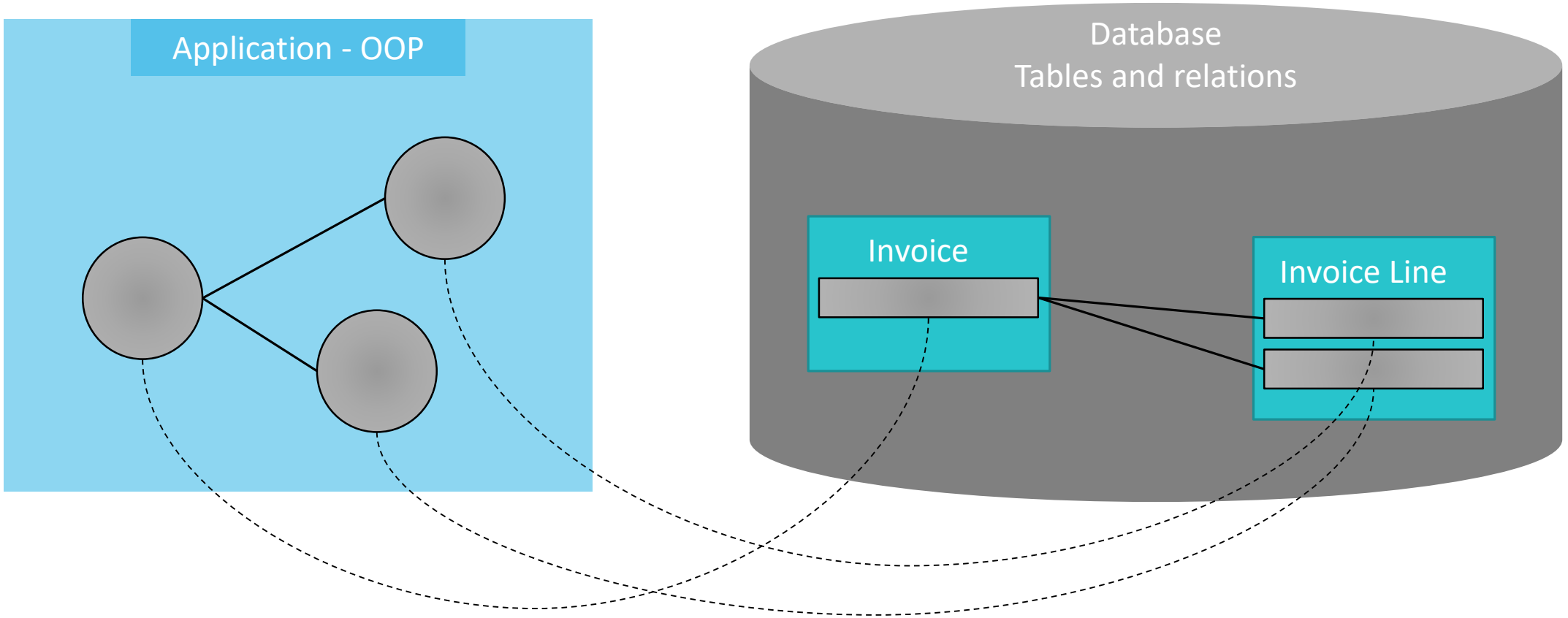
JPA

MARTIN MUDRA

Architecture overview



JPA – Java Persistence API



Why do we need ORM

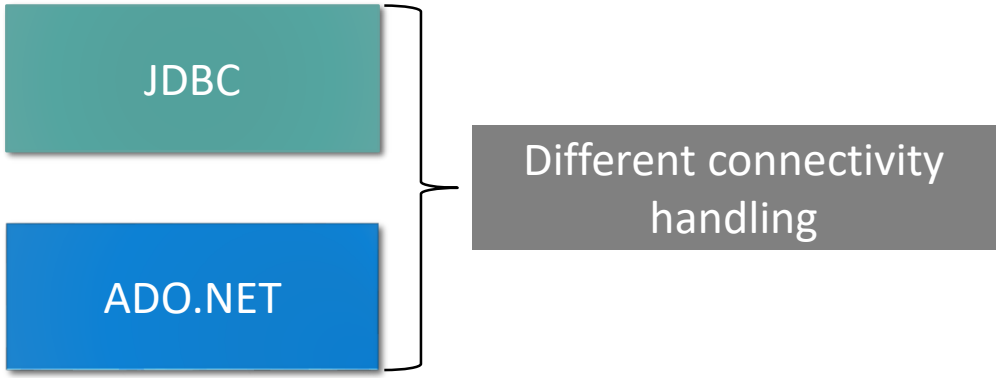
OOP works with classes and their instances

There are object databases that work with OOP natively

- Performance problems
- Standartization problems
- Problematic scalability

Commercial solutions are usually based on RDB

Native code



JDBC - Example

```
try (Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("SELECT * FROM MyTable"))
while ( rs.next() ) {
  int numColumns = rs.getMetaData().getColumnCount();

  myClass    result = new myClass();
  result.setPropertyA((int) rs.getObject(1));
  result.setPropertyB((String) rs.getObject(2));
  result.setPropertyC((int) rs.getObject(3));
  result.setPropertyD((int) rs.getObject(4));
  result.setPropertyE((int) rs.getObject(5));
  .
  .
  .
  results.Add(result);
}
```

Entity Manager

Takes care of work with entities, connections

Instance resolving

Resource Injection

```
@PersistenceContext (unitName="school_persistence_unit")
private EntityManager em;
//…follows work with em
```

Factory method

```
javax.persistence.EntityManager em;
 em = javax.persistence.Persistence
          .createEntityManagerFactory("school_persistence_unit")
          .createEntityManager();
//…follows work with em
```

Entity Manager - Configuration

Configuration is stored in file persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
 <persistence-unit name="school_persistence_unit" transaction-type="JTA“>
  <provider>oracle.toplink.essentials.PersistenceProvider</provider>
  <jta-data-source>jdbc/school</jta-data-source>
  <properties>
   <property name="toplink.ddl-generation" value="drop-and-create-tables"/>
  </properties>
 </persistence-unit>
</persistence>
```

Name of PU – userd for DI

JPA provider - imlementation

JNDI – Java Naming and Directory Interface

Stragy for table generation

More PU can be defined in one file

Provider - implementation

- Hibernate, Eclipselink (Oracle TopLink), OpenJPA, DataNucleus

Entity manager - methods

persist(Object entity)
- saves entity into DB

refresh(Object entity)
- reloads the entity from DB

merge(T entity)
- merges / connects an object to a persistent context

remove(Object entity)
- deletes from DB

find(ClassentityClass, Object primaryKey)
- finds an entity of T type using primary key

flush()
- makes sure to write to DB

create*Query(String sql, …)
- query to DB

Code example - entity

```
@Entity
public class TeacherEntity implements Serializable
{
  private static final long serialVersionUID = 1L;

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private String firstName;
  private String lastName;

  @OneToMany(mappedBy = "supervisor")
  private Collection students;
  …getters and setters…
}
```

POJO entity

Primary key

Collection of related entities

Properties – getters and setters

Code example EJB

```
@Stateless
public class SchoolSessionBean implements SchoolSessionBean
{
    @PersistenceContext(unitName = "school_persistence_unit")
    private EntityManager em;

    public StudentEntity addStudent(final String firstName, final String lastName)
    {
        StudentEntity newStudent = new StudentEntity();
        newStudent.setFirstName(firstName);
        newStudent.setLastName(lastName);

        em.persist(newStudent);
        return newStudent;
    }
}
```

Session EJB

DI – Entity Manager

Create new instance

Persist to DB

Hibernate / NHibernate

Implementation of JPA

Mature ORM Frameworks

Hibernate – since 2001

NHibernate – 2005 (version 1.0 mirrored functionality of hibernate 2.1)

Schema generation

- DDL

Entity manager – internally using Hibernate sessions

Mapping

Entity

- public / protected constructor (empty parameters)
- Non final / virtual methods (all)
- Public / protected getters and setters for fields (Properties)

POJO / POCO to RDB

XML

- *.hbm.xml

Property / fields

Attributes / Annotations

- @LOB – BLOB type
- @Transient – not persisted

Fluent

- Refactorable

XML Mapping example

```xml
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping assembly="ITJakub.DataEntities"
    namespace="ITJakub.DataEntities.Database.Entities"
    xmlns="urn:nhibernate-mapping-2.2">
  <class name="Author" table="[Author]">
    <id name="Id" column="Id">
      <generator class="identity" />
    </id>
    <property name="Name"/>

    <bag name="BookVersions" table="BookVersion_Author" inverse="true" lazy="true">
      <key column="Author" />
      <many-to-many class="BookVersion" column="BookVersion" />
    </bag>
  </class>
</hibernate-mapping>
```

Namespace / package

Class

Id generation strategy

M ku N relationship

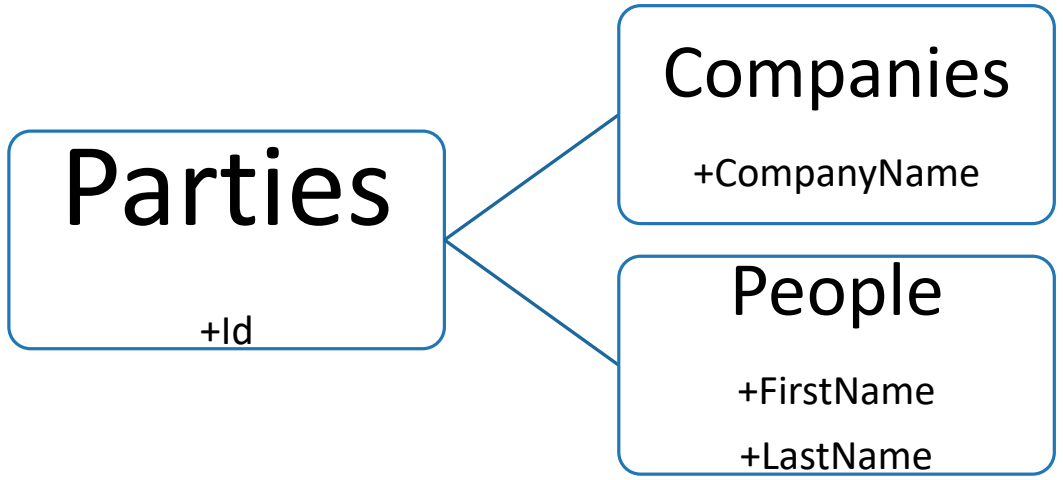
Inheritance – IS-a relations

Concrete table per class
- Select all for base class – multiple queries

Subclass (Table per hierarchy)
- Discriminator – string, number
- Discriminator-value attribute

Joined-subclass (Table per type)
- Left outer joins

Union-subclass
- Impossible to use identity (Hi-Lo etc.)



Primary key generation

Static generation – Identity

Auto

Table

HI-LO

- Hi – synchronized generation
- Lo - seed

Statistically unique – UUID / GUID

Hi-lo algorithm

Identifiers given:

$$\langle (hi - 1) * incrementSize + 1, hi * IncrementSize + 1)$$

Lo

$$\langle 0, increment\ Size)$$

Query language (H + NH)

Java Persistence query language – based on HQL

Dialect independent

Simmilar to SQL but on objects

Supports

- Selectors, updates, deletes
- Parameters
- Joins
- Aggregation function – AVG, MAX etc.
- Custom extensions for pagination
  - Take
  - Skip

```
Query query =
  session.createQuery("from Cat c where c = :id ");
query.setParameter("id", "7277");
List list = query.list();
```

```
DELETE Cat c WHERE c.Id = 151
```

```
FROM Cat c INNER JOIN c.Mate m LEFT OUTER JOIN c.Kittens k ORDER BY k.Name
```

Criteria (H + NH)

Elimination of magic strings – Properties?

Selectors only

Parameters

Subqueries, ordering.

Usefull tools

- Restrictions
- Transformers
- (N)HibernateUtil

```
IList<Cat> cats = sess.CreateCriteria<Cat>()
    .Add( Expression.Like("Name", "Fritz%") )
    .Add( Expression.Or(
        Expression.Eq( "Age", 0 ),
        Expression.IsNull("Age")
    ) )
    .List<Cat>();
```

```
List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.in( "name", new String[] { "Fritz", "Izi", "Pk" } ) )
    .add( Restrictions.disjunction()
        .add( Restrictions.isNull("age") )
        .add( Restrictions.eq("age", new Integer(0) ) )
        .add( Restrictions.eq("age", new Integer(1) ) )
        .add( Restrictions.eq("age", new Integer(2) ) )
    ) )    .list();
```

QueryOver – (NH)

Selectors only

Elimination of magic strings

Refactorable

Supports

- Joins
- Detached queryOver – subqueries
- Ordering
- Pagination etc.

```
session.QueryOver<Book>().Select(book => book.Id).List<long>();
```

```
session.QueryOver<Author>()
.Where(author => author.Name == name)
.SingleOrDefault<Author>();
```

Other query methods

Native SQL

Stored methods

Named queries

- SQL
- Mappings / annotations / config files
- Parameters naming problems

LINQ to SQL

```
@NamedQueries(
 { @NamedQuery(name = "Teacherentity.findAll", query = "SELECT t FROM Teacherentity t"),
   @NamedQuery(name = "Teacherentity.findById", query = "SELECT t FROM Teacherentity t WHERE t.id = :id"),
   @NamedQuery(name = "Teacherentity.findByFirstname", query = "SELECT t FROM Teacherentity t WHERE t.firstname =
       = :firstname")
})
```

Cascade

None
- Do not cascade

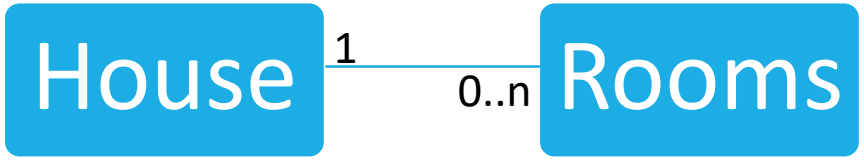
Save-update
- Check associations and save/update any object that require it even Many-to-many

Delete
- Delete all the objects in associations

Delete-orphan
- Delete and check other objects that reference me and delete orphans
- JPA – orphanRemoval

All
- Save/update/delete

All-delete-orphan
- Save/update/delete-orphan



Lazy

Fetch

True/ false/ proxy

- Default - true

Fetch-mode

- Select
- Join – limited to 1 relationship

N+1 Problem

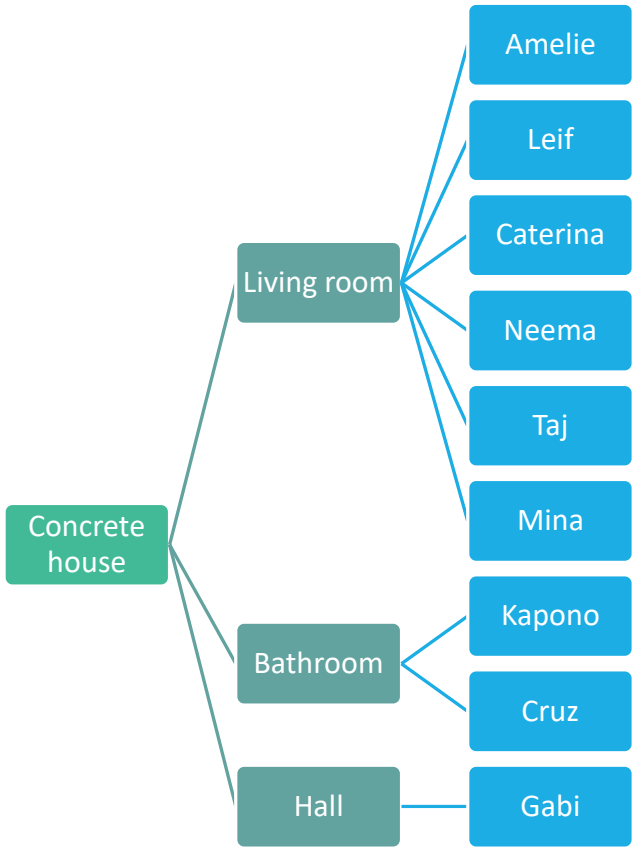
Select / delete / update

Cascade problems

Lazy initialization problems

- Batch-size
- Futures
- Load all at one roundtrip

Transaction



Isolations

Config settings – default value

Per-transaction settings

Deadlocks

Serializable
- RangeLock – multiversion lock
- Usually loss of concurrency

Repeatable reads
- Phantom reads
  - Two same queries in one transaction returns different result
- Keep read/write lock to selected data

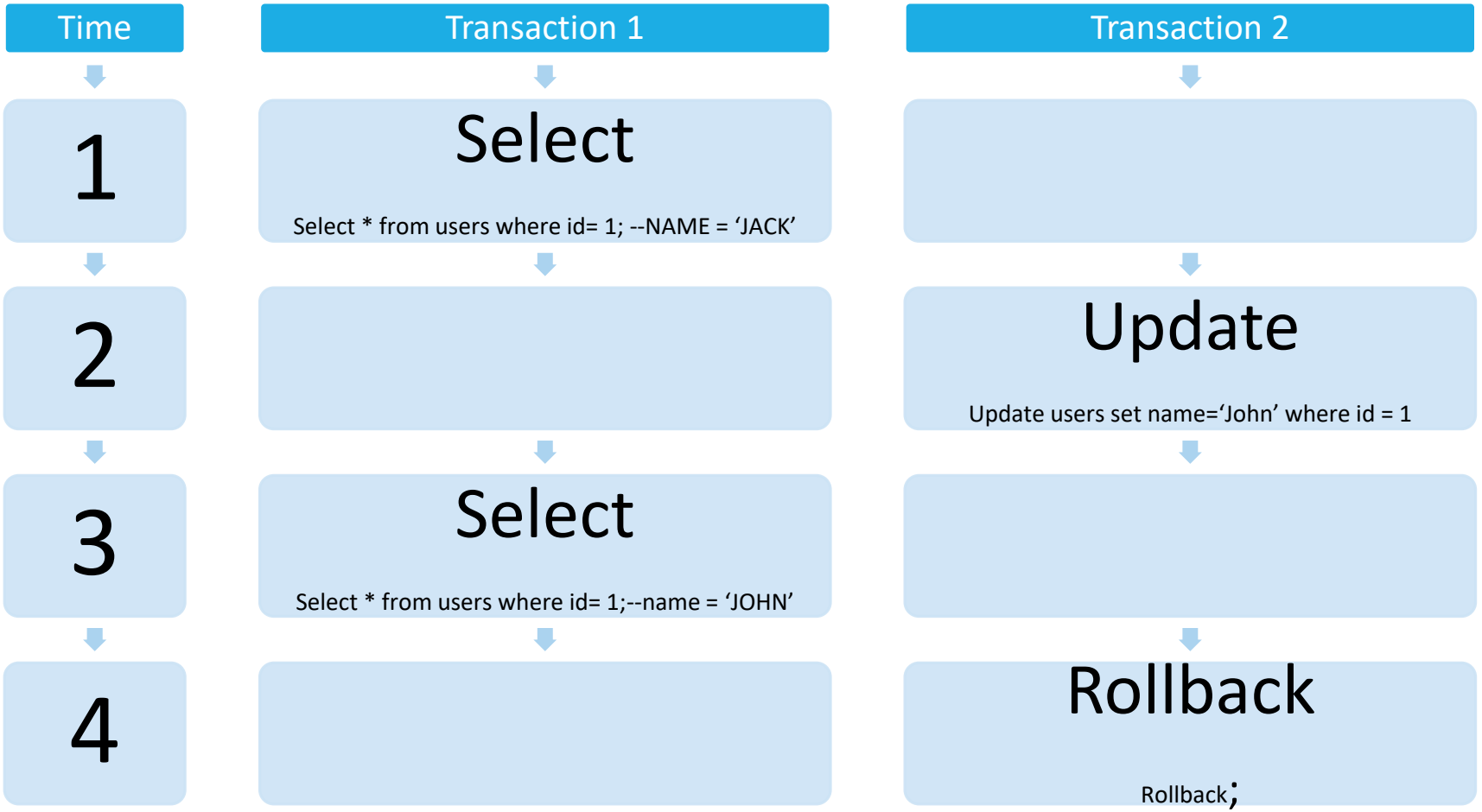
Read committed
- Keeps write-locks until transaction end
- Relase read-lock immidately after SELECT

Read uncommitted
- Dirty reads

Phantom reads

Time	Transaction 1	Transaction2
1	Select range SELECT * FROM users WHERE age BETWEEN 10 AND 30	
2		Insert & commit INSERT INTO users(…) VALUES (…, 'John', 27); COMMIT;
3	Select range & Commit SELECT * FROM users WHERE age BETWEEN 10 AND 30; COMMIT; --Get different values than prev read	

Dirty reads


Time
Transaction 1
Transaction 2
1
Select
Select * from users where id= 1; --NAME = 'JACK'
2
Update
Update users set name='John' where id = 1
3
Select
Select * from users where id= 1;--name = 'JOHN'
4
Rollback
Rollback;