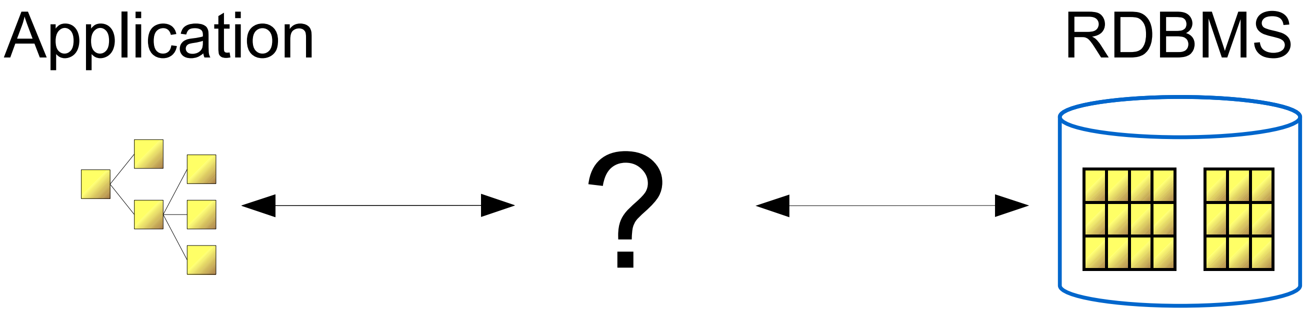
ObjectDB

Table of Content

- Introduction to ObjectDB and JPA
- Object Model: Types, Entities, Primary Keys.
- Database Connections
- JPA Fundamentals (CRUD Operations)
- Java Persistence Query Language (JPQL)
- JPA Query API and Criteria Query API
- References

Introduction

Object Relational Impedance Mismatch



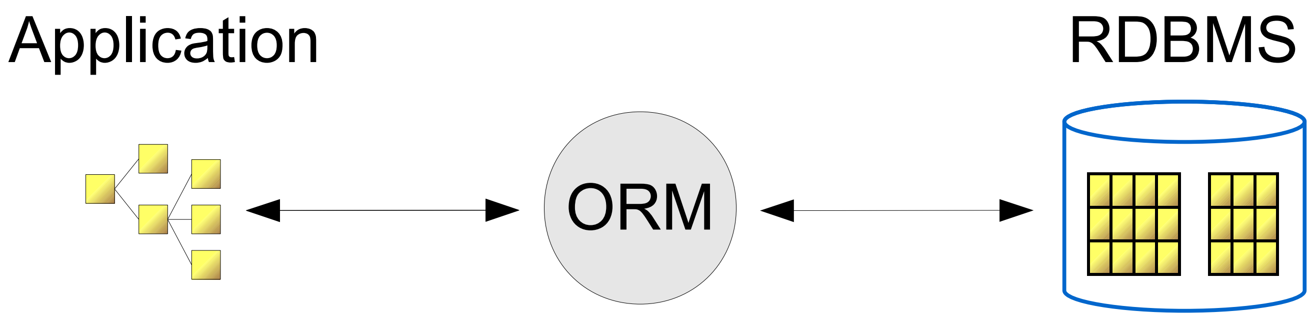
Application

?

RDBMS

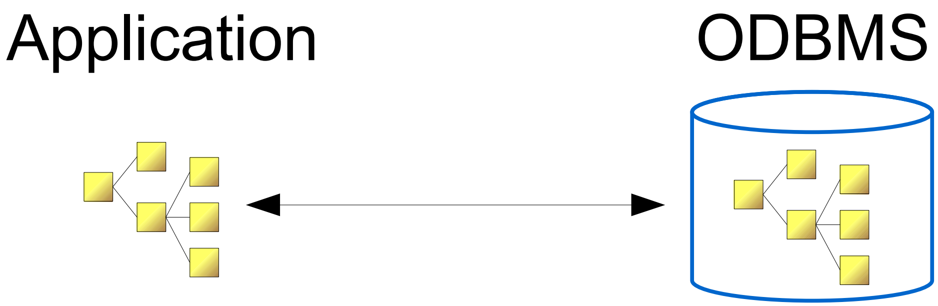
- Database uses tables and records.
- Application uses classes and objects.
- Solutions:
  - Application scope (ad hoc) transformations
  - Object Relational Mapping (ORM)
  - ODBMS (and other NoSQL Databases)

Object Relational Mapping (ORM)



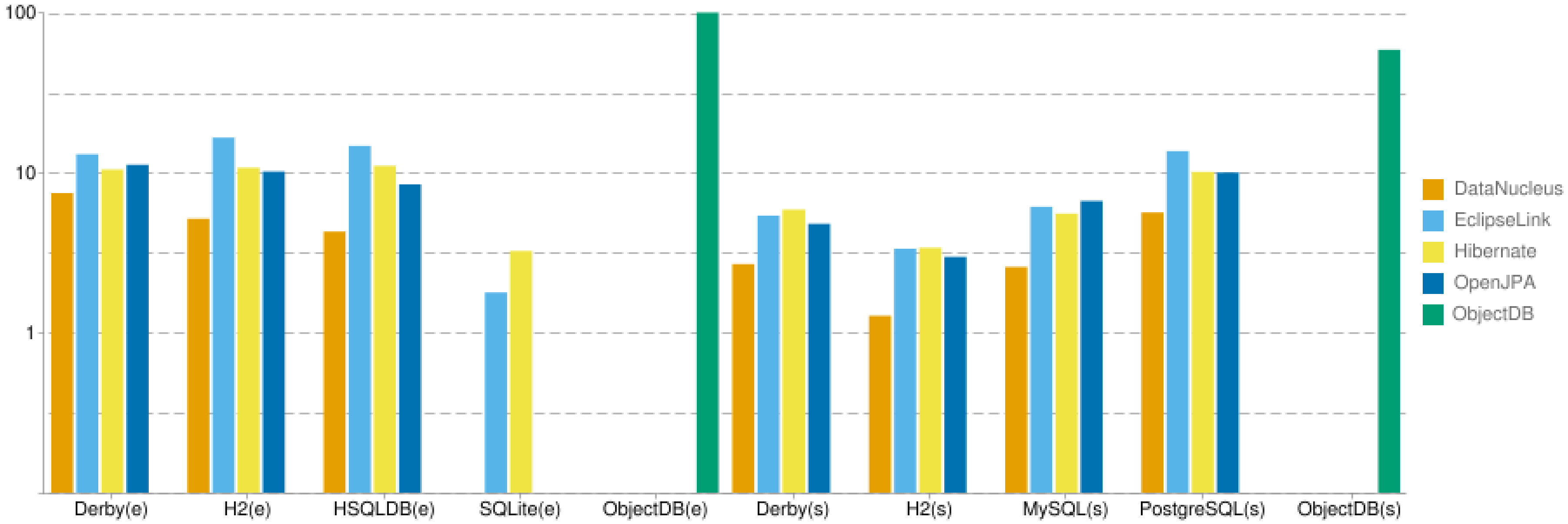
- Moderate conservative solution (Adapter) for the Object Relational Mismatch problem.
- JDO, EJB, JPA, Hibernate, EclipseLink, OpenJPA, DataNucleus, Entity Framework, NHibernate.
- Simplifies development (solves many issues).
- Doesn't solve performance issues.

Object Oriented Databases (ODBMS)



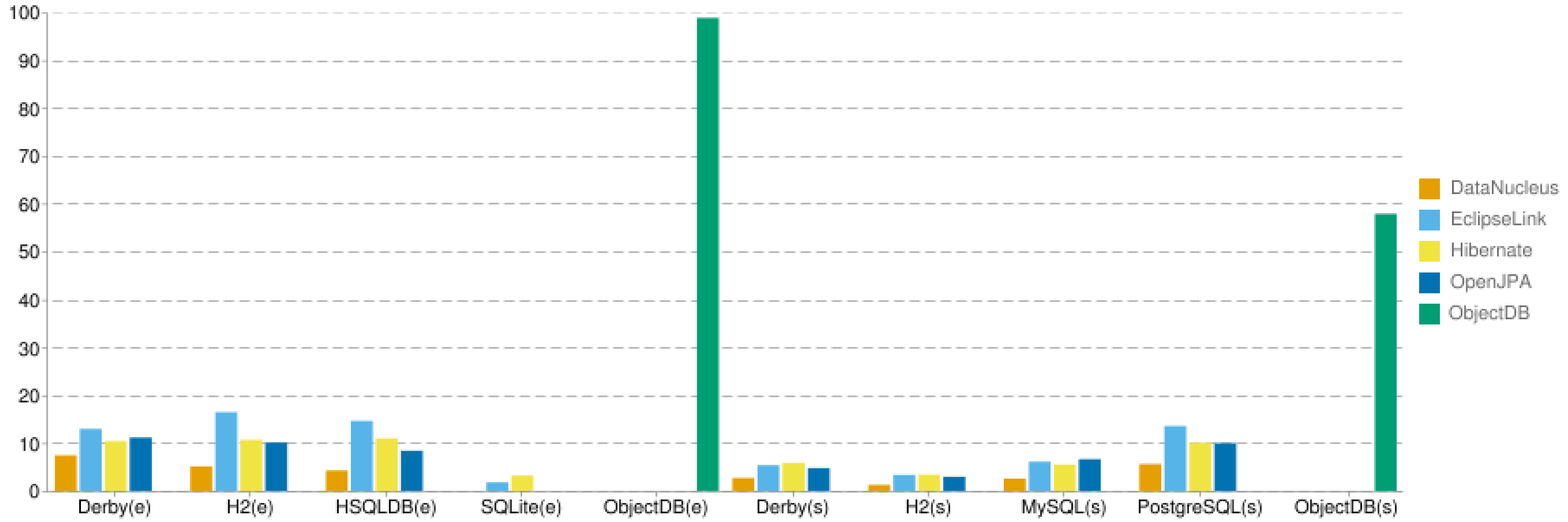
- No Object Relational Mismatch - application and database share the same class based schema.
- Very effective for complex data models.
- Standards: ODMG (and OQL), JDO.
- Versant, Objectivity/DB, GemStone, ObjectStore, InterSystems Cache, Matisse, db4o, ObjectDB.
- Small market share relative to RDBMS.

Performance (logarithmic scale)



http://jpab.org/All/All/All.html

Performance (linear scale)



http://jpab.org/All/All/All.html

ODBMS vs RDBMS vs Other NoSQL

	RDBMS + SQL (JDBC)	RDBMS + ORM (JPA)	ODBMS	Most NoSQL
Popularity	Very High	Very High	Low	Increasing
Tools	Many	Many	Few	Few
Independent Data Representation	Very High	Very High	Low	High
Consistency	ACID	ACID	ACID	Limited
Query Execution	Rich, Fast	Rich, Fast	Rich, Fast (Varies)	Limited
Complex Models	Complex, Slow	Slow	Fast	Varies
Object Oriented Integration	Low	High	Very High	Varies
Scalability	High	High	High	Very High

ObjectDB

- Commercial, free for small databases (up to 10 classes, one million objects).
- History:
  - 2003: version 1.0, JDO API.
  - 2010: version 2.0, JPA, JDO APIs.
  - Current version (2011) – 2.3.2.
- Embedded and Client-Server modes.
- Tools: Explorer, Doctor, Enhancer, Replication, Online Backup, BIRT Driver.

ObjectDB Sample Users

Anritsu Discover What's Possible™

Anritsu Corporation

AQUA

AQUA - Institute for applied research and quality promotion in health care

BAE SYSTEMS

BAE Systems Corporation
Europe's largest defense contractor

Ball

Ball Aerospace & Technologies

Honeywell

Honeywell International

hp

Hewlett-Packard (HP)

LIBRARY AND ARCHIVES CANADA

Canada's national collection of books, historical documents, government records, photos, films, maps, music...and more.

Los Alamos
NATIONAL LABORATORY
EST. 1943

MRC | National Institute for Medical Research

National Institute for Medical Research, UK

NAVSEA

NAVAL SEA SYSTEMS COMMAND
The U.S. Navy

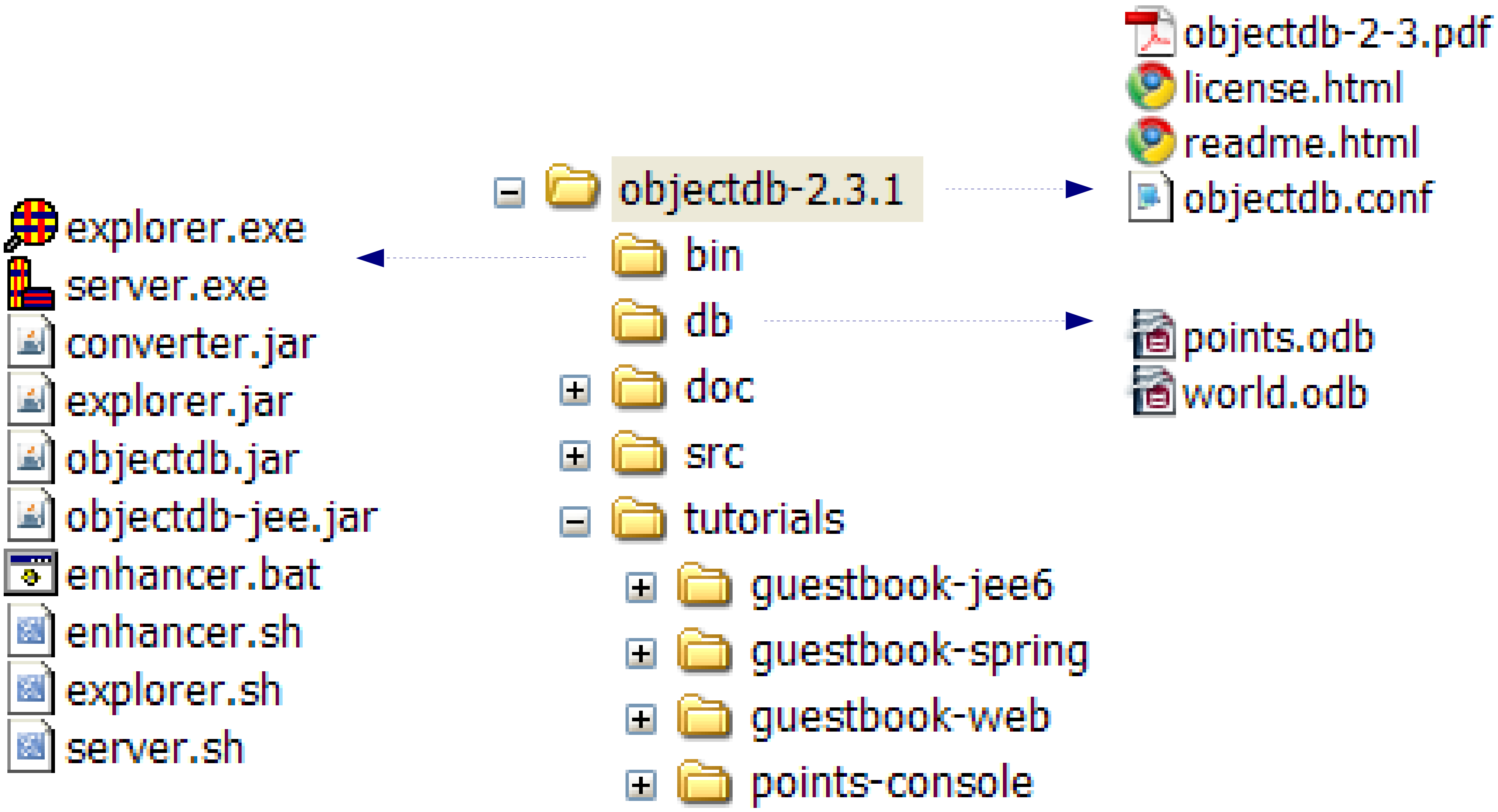
Novell.

Novell. Making IT Work As One™

RBC | RBC Royal Bank

Royal Bank of Canada

ObjectDB Distribution



objectdb-2.3.1

bin

db

doc

src

tutorials

guestbook-jee6

guestbook-spring

guestbook-web

points-console

objectdb-2-3.pdf

license.html

readme.html

objectdb.conf

points.odb

world.odb

explorer.exe

server.exe

converter.jar

explorer.jar

objectdb.jar

objectdb-jee.jar

enhancer.bat

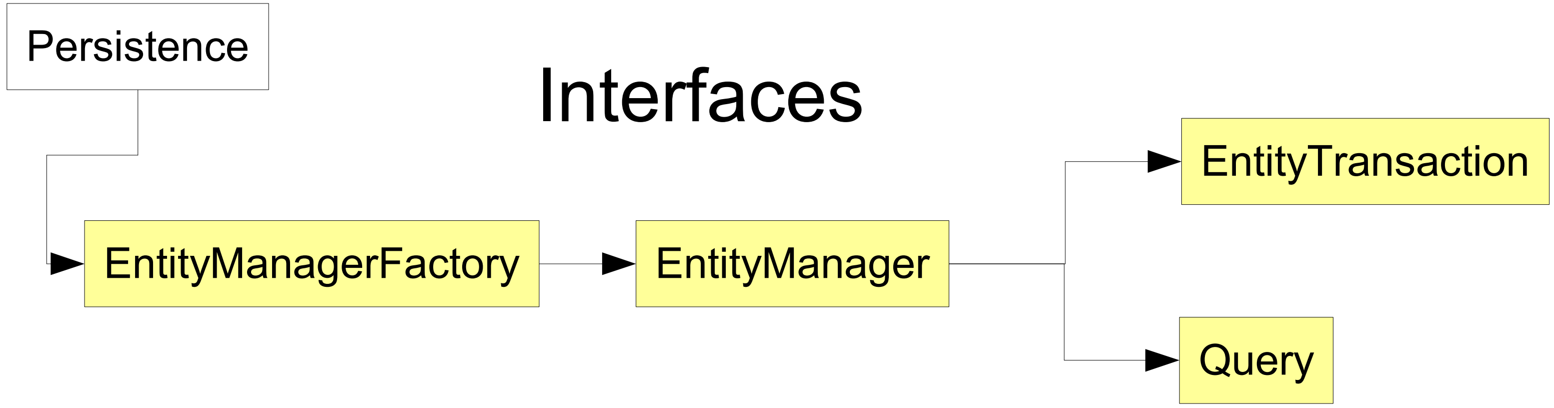
enhancer.sh

explorer.sh

server.sh

API Basics (JPA)

import javax.persistence.*;



Persistence

Interfaces

EntityManagerFactory

EntityManager

EntityTransaction

Query

Annotations

@Entity

@Embeddable

@Id

@GeneratedValues

Enums

FlushModeType

LockModeType

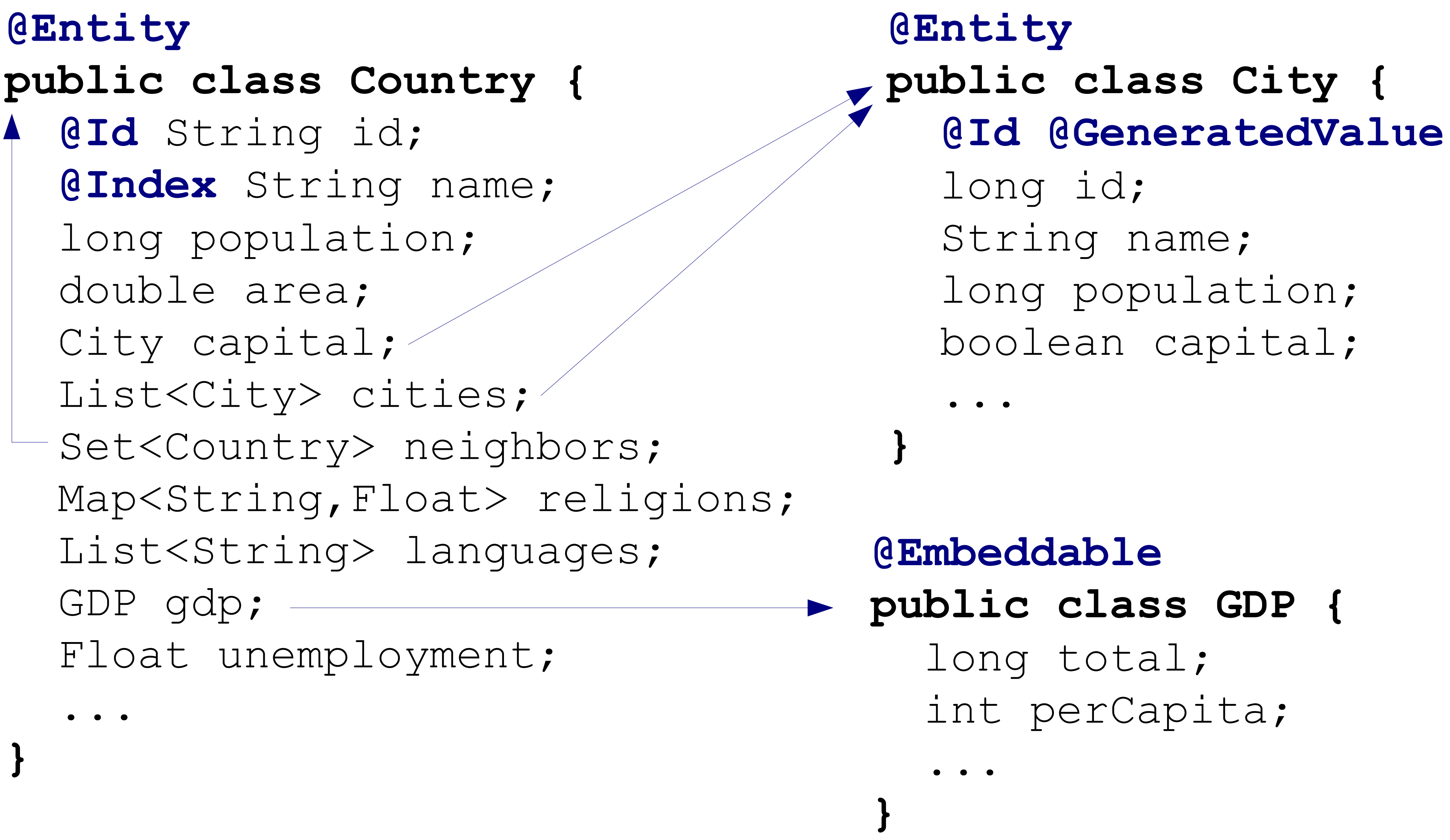
Exceptions

PersistenceException

TransactionRequiredException

Object Model (Types)

World Object Model

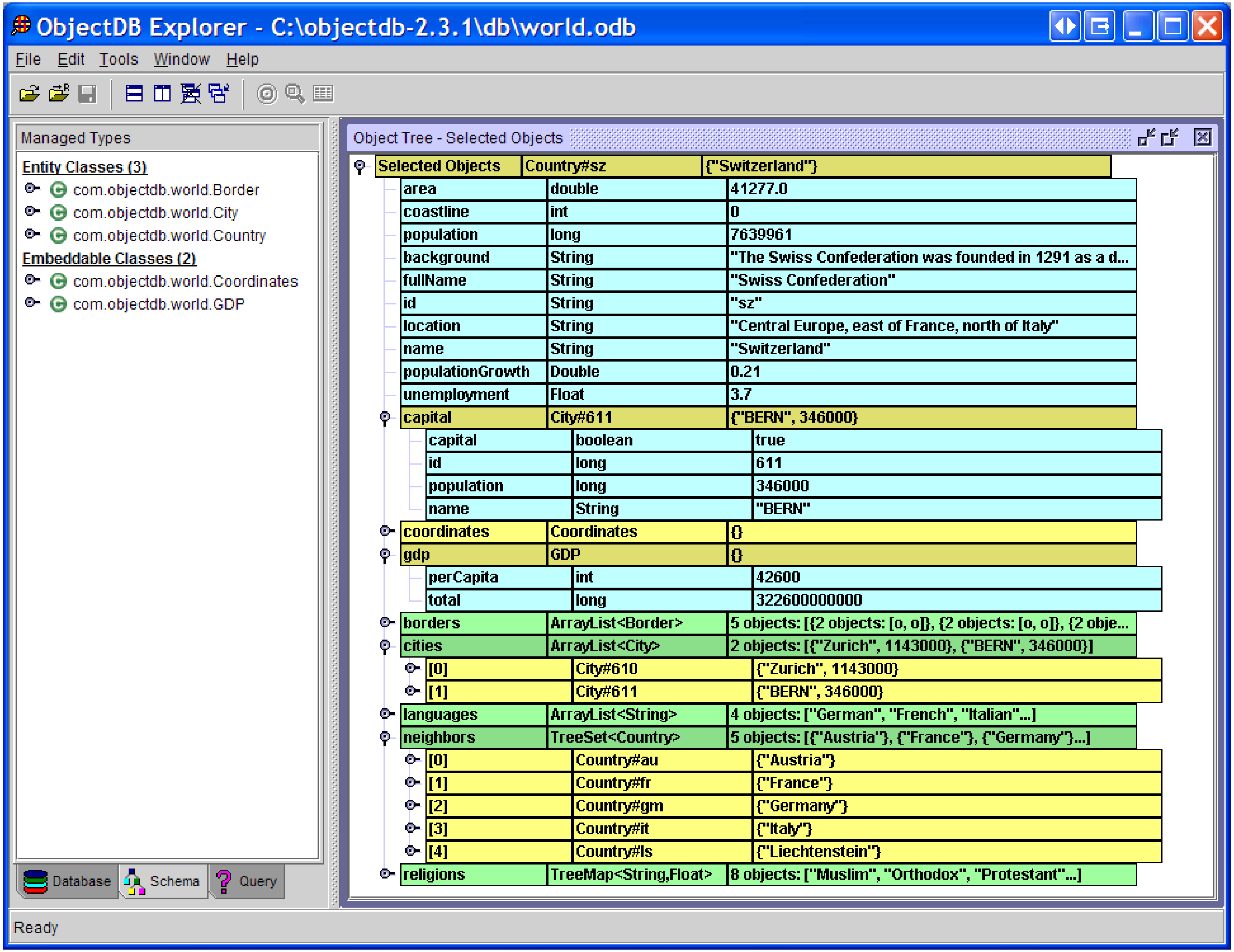


```
@Entity
public class Country {
  @Id String id;
  @Index String name;
  long population;
  double area;
  City capital;
  List<City> cities;
  Set<Country> neighbors;
  Map<String,Float> religions;
  List<String> languages;
  GDP gdp;
  Float unemployment;
  ...
}
```

```
@Entity
public class City {
  @Id @GeneratedValue
  long id;
  String name;
  long population;
  boolean capital;
  ...
}
```

```
@Embeddable
public class GDP {
  long total;
  int perCapita;
  ...
}
```

World Database in the Explorer


ObjectDB Explorer - C:\objectdb-2.3.1\db\world.odb
File
Edit
Tools
Window
Help
Managed Types
Object Tree - Selected Objects
Database
Schema
Query
Ready

Entity Classes (3)

- com.objectdb.world.Border
- com.objectdb.world.City
- com.objectdb.world.Country

Embeddable Classes (2)

- com.objectdb.world.Coordinates
- com.objectdb.world.GDP

Selected Objects	Country#sz	{"Switzerland"}
area	double	41277.0
coastline	int	0
population	long	7639961
background	String	"The Swiss Confederation was founded in 1291 as a d...
fullName	String	"Swiss Confederation"
id	String	"sz"
location	String	"Central Europe, east of France, north of Italy"
name	String	"Switzerland"
populationGrowth	Double	0.21
unemployment	Float	3.7
capital	City#611	{"BERN", 346000}
capital	boolean	true
id	long	611
population	long	346000
name	String	"BERN"
coordinates	Coordinates	{}
gdp	GDP	{}
perCapita	int	42600
total	long	322600000000
borders	ArrayList<Border>	5 objects: [{2 objects: [o, o]}, {2 objects: [o, o]}, {2 obje...
cities	ArrayList<City>	2 objects: [{"Zurich", 1143000}, {"BERN", 346000}]
[0]	City#610	{"Zurich", 1143000}
[1]	City#611	{"BERN", 346000}
languages	ArrayList<String>	4 objects: ["German", "French", "Italian"...]
neighbors	TreeSet<Country>	5 objects: [{"Austria"}, {"France"}, {"Germany"}...]
[0]	Country#au	{"Austria"}
[1]	Country#fr	{"France"}
[2]	Country#gm	{"Germany"}
[3]	Country#it	{"Italy"}
[4]	Country#ls	{"Liechtenstein"}
religions	TreeMap<String,Float>	8 objects: ["Muslim", "Orthodox", "Protestant"...]

Persistent State

- Entity & Embeddable Fields:
  - Persistent Fields
  - Transient Fields (transient, @Transient, final)
  - Inverse (Mapped By) Fields
- Entity Fields:
  - Primary Key Fields (@Id, @EmbeddedId)
  - Version Field (@Version)
- Access Mode: Field / Property

Supported Types

- User Defined Entity and Embeddable classes
- Java Value Types:
  - Primitives (int, float, …)
  - Wrappers (Integer, Float, …)
  - String, BigInteger, BigDecimal
  - Date, Time, Timestamp, Calendar
- Enum types (system and user defined)
- Java Collections and Maps
- Any other serializable type

Primary Key

- Every entity object can be uniquely identified by type + primary key - e.g. in em.find(cls, pk).
- Primary Key Modes:
  - Implicit Primary Key
  - Single Field (@Id)
  - Composite - Multiple Fields (@Id, @IdClass)
  - Composite – Embedded Id (@EmbeddedId)
- Most comparable value types are supported.
- Set by Application or by @GeneratedValue.

Composite Primary Key

```
@Entity
@IdClass(ProjectId.class)  ──►  Class ProjectId {
public class Project {            int departmentId;
  @Id int departmentId;           long projectId;
  @Id long projectId;           }
  ...
}
```

```
@Entity
public class Project {               @Embeddable
  @EmbeddedId ProjectId id;  ──►  Class ProjectId {
  ...                                int departmentId;
}                                    long projectId;
                                   }
```

Indexes

- Indexes are used in queries for:
  - Retrieval (range / scan).
  - Ordering results.
  - MAX and MIN aggregate expressions.
- JDO Annotations for indexes:

  @Index, @Indices, @Unique, @Uniques.
- Supported Indexes:
  - Single Field / Composite
  - Indexed collections
  - Multi Part Path Index

Schema Evolution

- Automatic Schema Evolution when:
  - New field is added.
  - Existing field is removed.
  - Field type is modified.
  - Class hierarchy is changed.
  - Fields are moved in class hierarchy.
- Semi-automatic Schema Evolution when:
  - Package is renamed.
  - Class is renamed or moved.
  - Field is renamed.

Database Connections

Connection Management

Obtaining an EntityManager

```
EntityManagerFactory emf =
  Persistence.createEntityManagerFactory(
    "$objectdb/db/world.odb");

EntityManager em = emf.getEntityManager();
```

Cleanup

```
em.close();

emf.close();
```

Connection URLs

Embedded URLs:

```
C:\db\world.odb   $objectdb/db/world.odb
db/world.odb      objectdb:db/world.tmp
```

Client Server URLs:

```
objectdb://localhost/my/world.odb
objectdb://localhost:6136/my/world.odb
```

URL Parameters

```
objectdb://localhost/world.odb;user=a;password=a
objectdb:myDbFile.tmp;drop
```

persistence.xml

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

   <persistence-unit name="world-pu">

     <provider>com.objectdb.jpa.Provider</provider>

     <class>com.objectdb.world.Country</class>
     <class>com.objectdb.world.Border</class>
     <class>com.objectdb.world.City</class>
     <class>com.objectdb.world.Coordinates</class>
     <class>com.objectdb.world.GDP</class>

     <properties>
       <property name="javax.persistence.jdbc.url"
                 value="objectdb://localhost/world.odb"/>
       <property name="javax.persistence.jdbc.user" value="admin"/>
       <property name="javax.persistence.jdbc.password" value="admin"/>
     </properties>

   </persistence-unit>

</persistence>
```

In Java EE (EJB) and Spring

Java EE

```
@Stateless
public class MyEJB {
  // Injected database connection:
  @PersistenceContext private EntityManager em;
  ...
}
```

Spring MVC

```
@Component
public class MyComponent {
  // Injected database connection:
  @PersistenceContext private EntityManager em;
  ...
}
```

CRUD Operations

Creating Entities

```
em.getTransaction().begin();

Country country = new Country();
country.setId("br");
  // A new unmanaged entity is constructed.

em.persist(country);
  // The new entity object becomes managed.

em.getTransaction().commit();
  // Updates are committed to the database.
```

Retrieving Entities

Retrieval by Id

```
Country country = em.find(Country.class, "br");
```

Retrieval by Query

```
TypedQuery<Country> query =
  em.createQuery("select c from Country c");
List<Country> results = query.getResultList();
```

Retrieval by Navigation

```
for (Country c : results) {
  System.out.println(c.getCapital().getName());
}
```

Updating Entities

```
em.getTransaction().begin();

Country c = em.find(Country.class, "br");
  // A managed object is retrieved.

c.setPopulation(192376496)
  // The managed object is updated.

em.getTransaction().commit();
  // Updates are committed to the database.
```

- Efficient change detection for enhanced classes.
- Snapshot comparison for non enhanced classes.

Deleting Entities

```
em.getTransaction().begin();

Country c = em.find(Country.class, "br");
  // A managed object is retrieved.

em.remove(c);
  // The object is marked as removed.

em.getTransaction().commit();
  // Updates are committed to the database.
```

Cascading and Fetch Types

- Operations are cascaded by annotating relationships:

```
@OneToMany(cascade=CascadeType.ALL,
           fetch=FetchType.EAGER)
private List<City> cities;
```

- Relationship annotations:

  @OneToOne, @ManyToOne, @OneToMany, @ManyToMany.

- Cascading Types:

  ALL, PERSIST, REMOVE, REFRESH, DETACH, MERGE.

- Fetch Types: EAGER, LAZY.

Advanced Topics

- Lock Management: Optimistic, Pessimistic.
- Detach and Merge
- Lifecycle Events: @PrePersist, @PostPersist, @PostLoad @PreUpdate, @PostUpdate, @PreRemove, @PostRemove.
- Bidirectional Relationships
- L2 Cache
- Metamodel API
- Configuration

Queries (JPQL)

A Basic JPQL Query

```
SELECT c FROM Country [AS] c
```

- FROM declares query identification variables (at least one) representing entity scopes.
- SELECT declares the results.
- SELECT and FROM are required.
  WHERE, GROUP BY, HAVING and ORDER BY are optional.
- Queries are case insensitive except names of entities, attributes and relationships.

JPQL Literals

- String literals are Unicode characters enclosed in single quotes (e.g. **'College'**), where single quote itself is represented by two single quotes (e.g. **'College''s'**)
- Numeric literals follow the Java and SQL syntax (e.g. **100**, **5.7**) including optional type suffixes (e.g. **1.5F**).
- enum literals follow the Java syntax, where class name must be specified.
- Boolean literals are **TRUE** and **FALSE**

Operators and Precedence

- **Navigation (.)**
- **Arithmetic Operators**
  +, - (unary)
  * (multiplication), / (division)
  + (addition), - (subtraction)
- **Comparison Operators**
  =, <, <= , >, >=, <>, [NOT] BETWEEN, ⇒
  [NOT] LIKE, [NOT] IN, IS [NOT] NULL, ⇒
  IS [NOT] EMPTY, [NOT] MEMBER [OF]
- **Logical operators**
  NOT, AND, OR

WHERE

Basic String Expression

```
SELECT c FROM Country c
WHERE c.name = 'Brazil'
```

Basic Numeric Expression

```
SELECT c FROM Country c
WHERE c.area > 1000000
```

Logical Operators

```
SELECT c FROM Country c
WHERE (NOT c.area > 1000000) OR
      (c.population >= 10000000 AND
       c.population <= 50000000)
```

Parameters

Positional Parameters

```
SELECT c FROM Country c
WHERE c.area > ?1 AND
      c.population <= ?2
```

Named Parameters

```
SELECT c FROM Country c
WHERE c.area > :area AND
      c.population <= :population
```

Same parameter can be used more than once in the query string.

Relationship Navigation

-
  ```
  SELECT c FROM Country c
  WHERE c.capital.name = 'Paris'
  ```

-
  ```
  SELECT c
  FROM Country c, JOIN c.cities city
  WHERE city.name = 'Paris'
  ```

-
  ```
  SELECT c
  FROM Country c, JOIN c.capital city
  WHERE city.name = 'Paris'
  ```

-
  ```
  SELECT c FROM Country c
  WHERE c.cities.name = 'Paris'
  ```

JOIN Operation Types

Inner Join

```
SELECT c, l FROM Country c
[INNER] JOIN c.languages l
```

Left Outer Join

```
SELECT c, l FROM Country c
LEFT [OUTER] JOIN c.languages l
```

Fetch Join

```
SELECT c FROM Country c
LEFT [OUTER] JOIN FETCH c.languages
```

Comparison Expressions

```
SELECT c FROM Country c WHERE …
   c.area <= :minArea
   c.area BETWEEN :a1 AND :a2
   c.population.size / c.area > :ratio
   c.name <> :name
   c.name LIKE '_raz%'
```

String Functions

```
SELECT c FROM Country c WHERE …
  CONCAT(c.name, "123") = 'Brazil123'
  LENGTH(c.name) = :length
  LOCATE(c.name, 'a') > 0
  LOCATE(c.name, 'a', 2) > 0
  LOWER(c.name) = :name
  UPPER(c.name) = :name
  SUBSTRING(c.name 3, 3) = 'azi'
  TRIM(c.name) = :name
  TRIM(LEADING 'B' FROM c.name) = 'razil'
  TRIM(TRAILING 'l' FROM c.name) = 'Brazi'
```

Other Functions

```
SELECT c FROM Country c WHERE ...
   MOD(c.population.size, 2) = 0
   ABS(c.area) <= 1000000
   SQRT(c.area) <= 1000

   c.gdp.date > CURRENT_DATE
   (... CURRENT_TIME, CURRENT_TIMESTAMP)

   SIZE(c.languages) >= 3
```

Other Conditional Expressions

```
SELECT c FROM Country c WHERE ...

  c.capital IS [NOT] NULL

  c.languages IS [NOT] EMPTY
  :lang [NOT] MEMBER [OF] c.languages

  c.name IN ('France', 'Germany')
```

Selection and Projection

```
SELECT c.name FROM Country c
SELECT c.capital FROM Country c
SELECT c.name, c.capital FROM Country c

SELECT l
FROM Country c JOIN c.languages l
WHERE c.population > 10000000

SELECT DISTINCT l
FROM Country c JOIN c.languages l
WHERE c.population > 10000000
```

Constructor Expressions

```
SELECT NEW results.NameAndArea(
  c.name, c.area) FROM Country c

public class NameAndArea {
  private String name;
  private int area;
  public NameAndArea(String n, int a) {
    name = n;
    area = a;
  }
  String getName() { return name; }
  int getArea() { return area; }
}
```

ORDER BY

```
SELECT c FROM Country c
ORDER BY c.name ASC
```

```
SELECT c FROM Country c
ORDER BY c.capital.name DESC
```

```
SELECT c FROM Country c
ORDER BY c.name, c.capital.name
```

Aggregate Functions

```
SELECT COUNT(c) FROM Country c
  => Long
SELECT SUM(c.area) FROM Country c
  => Long, Double, BigInteger, BigDecimal, null
SELECT AVG(c.area) FROM Country c
  => Double
SELECT MAX(c.name) FROM Country c
  => String, Date or Numeric Type
SELECT MIN(c.name) FROM Country c
  => String, Date or Numeric Type
```

GROUP BY / HAVING

```
SELECT SUBSTRING(c.name, 1, 1), COUNT(c)
FROM Country c
GROUP BY SUBSTRING(c.name, 1, 1)
```

```
SELECT SUBSTRING(c.name, 1, 1), COUNT(c)
FROM Country c
WHERE LENGTH(c.name) >= 3
GROUP BY SUBSTRING(c.name, 1, 1)
HAVING COUNT(c) >= 5
ORDER BY SUBSTRING(c.name, 1, 1)
```

Update and Delete Queries

-
  ```
  UPDATE Country AS c
  SET c.name = TRIM(c.name)
  ```

-
  ```
  UPDATE Country SET name = TRIM(name)
  ```

-
  ```
  UPDATE Country SET name = TRIM(name)
  WHERE population > 1000
  ```

-
  ```
  DELETE FROM Country AS c
  WHERE c.population > 1000
  ```

-
  ```
  DELETE FROM Country
  WHERE population > 1000
  ```

The Query API

Static Query Execution

```
@NamedQuery(name="findLarge",
  query = "SELECT c FROM Country c " +
          "WHERE c.area > :area")

@NamedQueries(
  {@NamedQuery(...), @NamedQuery(...)})

List<Country> results = (List<Country>)
  em.createNamedQuery("findLarge").
  setParameter("area", 100000).
  getResultList();
```

Dynamic Query Execution

```
String query =
  "SELECT c FROM Country c " +
  "WHERE c.area > :area AND " +
  "c.population <= :population";
List<Country> results =
  (List<Country>)em.createQuery(query).
    setParameter("area", 100000).
    setParameter("population", 1000000).
    getResultList();

Country c = (Country)em.createQuery(query).
  setParameter("area", 100000).
  setParameter("population", 1000000).
  getSingleResult();
```

Criteria Queries

```
CriteriaBuilder cb =
  em.getCriteriaBuilder();
CriteriaQuery<Country> q =
  cb.createQuery(Country.class);

Root<Country> c = q.from(Country.class);
q.select(c);
ParameterExpression<Integer> p =
  cb.parameter(Integer.class);
q.where(cb.le(c.get("population"), p));
```

The Query API

```
Query setFlushMode(
  FlushModeType flushMode)
    // AUTO or COMMIT

Query setFirstResult(int startPos)
Query setMaxResults(int maxResult)
Query setHint(
  String hintName, Object value)

List getResultList()
Object getSingleResult()
int executeUpdate()
```

The Query API (cont)

```
Query setParameter(
  String name, Object value)
Query setParameter(String name,
  Calendar value, TemporalType tempType)
Query setParameter(String name,
  Date value, TemporalType tempType)

Query setParameter(
  int position, Object value)
Query setParameter(int pos,
  Calendar value, TemporalType tempType)
Query setParameter(int pos,
  Date value, TemporalType tempType)
```

References

References

ObjectDB Website

http://www.objectdb.com

Tutorials

http://www.objectdb.com/tutorial

Manual

http://www.objectdb.com/java/jpa

Forum

http://www.objectdb.com/database/forum