Querying Entities

Antonio Goncalves
JAVA CHAMPION

@agoncal www.antoniogoncalves.org

Previous Module

Relationships

- Join tables or join columns
- Direction, a cardinality, cascade events
- Lazily or eagerly

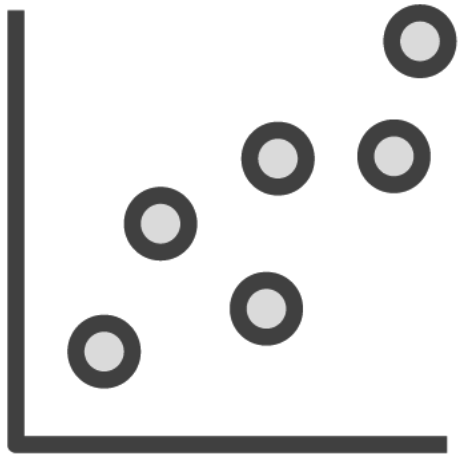
Inheritance

- 3 different strategies
- Inherit from entities or mapped superclasses

Overview

- Query entities
- Java Persistence Query Language
- Rich syntax
- Dynamic queries
- Named queries

Why Do We Need Queries?



- Getting data out of the database is crucial
- Search
- Sort
- Aggregate
- Analyze
- Reporting
- Business intelligence

Structured Query Language

Relational databases

SQL

SELECT statement

Retrieves data from one or more tables

Rich syntax

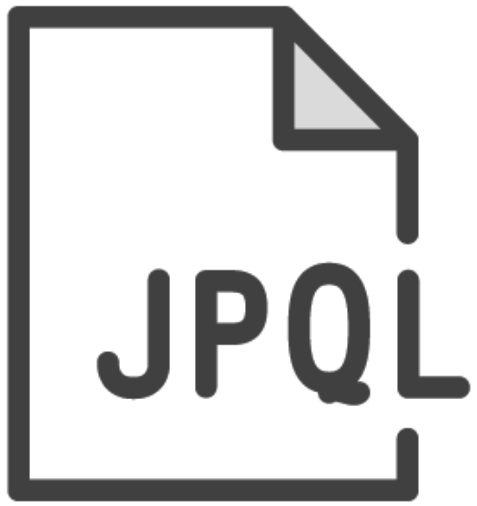
Clauses, expressions, predicates, statements

Structured Query Language

```
SELECT *
FROM   item
WHERE  unit_cost > 100.00
ORDER BY title;

SELECT   title AS Title, COUNT(*) AS Authors
FROM     item
  JOIN   book_author
  ON     item.id = book_author.book_fk
GROUP BY item.title;
```

Java Persistence Query Language



- Manipulate entities individually
- CRUD operations
- Finding by ID is limiting
- Retrieve an entity by criteria
- Inherent to relational databases
- JPA has JPQL

From SQL to JPQL

Query language with SQL heritage

SQL is relational database oriented

JPQL is object-oriented

Database independent query language

JPQL query translated into SQL

JDBC calls

Returns entities

From SQL to JPQL

```
SELECT     *
FROM       item
WHERE      unit_cost > 100.00
ORDER BY   title;
```

```
SELECT     i
FROM       Item i
WHERE      i.unitCost > 100
ORDER BY   i.title
```

From SQL to JPQL

```
SELECT    title AS Title, COUNT(*) AS Authors
FROM      item
  JOIN    book_author
  ON      item.id = book_author.book_fk
GROUP BY  item.title;
```

```
SELECT    b.title, COUNT(a)
FROM      Book b
LEFT JOIN b.authors a
GROUP BY  b
```

JPQL Syntax

```
SELECT     <select clause>
FROM       <from clause>
[WHERE     <where clause>]
[ORDER BY  <order by clause>]
[GROUP BY  <group by clause>]
[HAVING    <having clause>]
```

JPQL Syntax: Functions

```
SELECT     <select clause>
FROM       <from clause>
[WHERE     <where clause>]
[ORDER BY  <order by clause>]
[GROUP BY  <group by clause>]
[HAVING    <having clause>]

<function>   AVG, COUNT, MAX, MIN, SUM
```

JPQL Syntax: Operators

```
SELECT    <select clause>
FROM      <from clause>
[WHERE    <where clause>]
[ORDER BY <order by clause>]
[GROUP BY <group by clause>]
[HAVING   <having clause>]

<operators>  =, >, >=, <, <=, <>, [NOT] BETWEEN, [NOT] IN,
             [NOT] LIKE, IS [NOT] NULL, IS [NOT] EMPTY,
             [NOT] MEMBER [OF]
```

JPQL Syntax: Expressions

```
SELECT    <select clause>
FROM      <from clause>
[WHERE    <where clause>]
[ORDER BY <order by clause>]
[GROUP BY <group by clause>]
[HAVING   <having clause>]

<num exp.>    ABS, SQRT, MOD, SIZE, INDEX

<string exp.> CONCAT, SUBSTRING, TRIM, LOWER, UPPER,
              LENGTH, LOCATE

<date exp.>   CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP
```

Simplest JPQL Query

```
SELECT      b
FROM        Book b
WHERE       b.unitCost > 100
```

Select Clause

```
SELECT      b
FROM        Book b
```

Select Clause

```
SELECT      b.title, b.unitCost, b.isbn
FROM        Book b
```

Select Clause

```
SELECT      COUNT(b)
FROM        Book b
```

Select Clause

```
SELECT      AVG(b.unitCost)
FROM        Book b
```

Select Clause

```
SELECT      b.publisher
FROM        Book b
```

Select Clause

```
SELECT      b.publisher.name
FROM        Book b
```

Select Clause

```
SELECT    DISTINCT(b.publisher.name)
FROM      Book b
```

From Clause

```
SELECT      b
FROM        Book b
```

Where Clause

```
SELECT      b
FROM        Book b
WHERE       b.unitCost > 29
```

Where Clause

```
SELECT      b
FROM        Book b
WHERE       b.unitCost > 29 AND b.nbOfPage < 100
```

Where Clause

```
SELECT      b
FROM        Book b
WHERE       b.unitCost > 29 AND b.nbOfPage BETWEEN 50 AND 90
```

Where Clause

```
SELECT      b
FROM        Book b
WHERE       b.title LIKE '%java%'
```

Where Clause

```
SELECT      b
FROM        Book b
WHERE       LOWER(b.title) LIKE '%java%'
```

Order by Clause

```
SELECT     b
FROM       Book b
WHERE      LOWER(b.title) LIKE '%java%'
ORDER BY   b.title
```

Order by Clause

```
SELECT     b
FROM       Book b
WHERE      LOWER(b.title) LIKE '%java%'
ORDER BY   b.title ASC
```

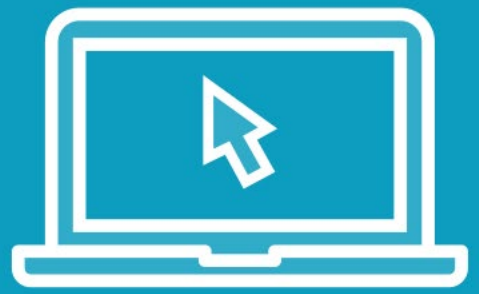
Order by Clause

```
SELECT     b
FROM       Book b
WHERE      LOWER(b.title) LIKE '%java%'
ORDER BY   b.title DESC
```

Order by Clause

```
SELECT     b
FROM       Book b
WHERE      LOWER(b.title) LIKE '%java%'
ORDER BY   b.title DESC, b.nbOfPage ASC
```

Demo



Execute JPQL queries

On several entities

JPQL syntax

Select statements

Dot navigation

Queries

JPQL statements

Executed in queries

Dynamic query

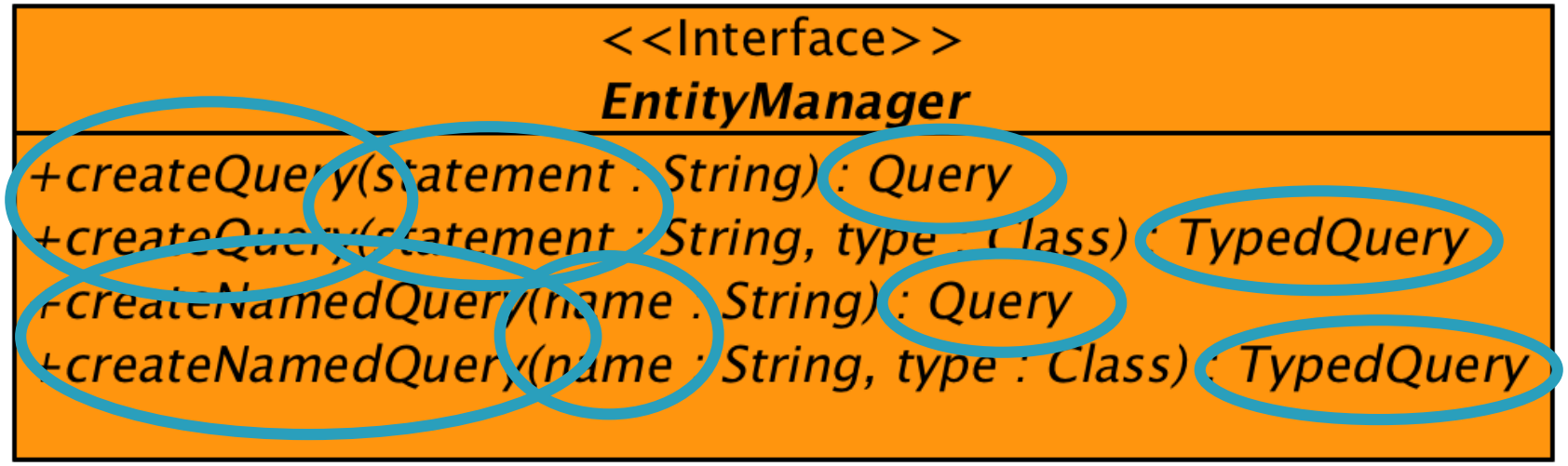
Named query

Query

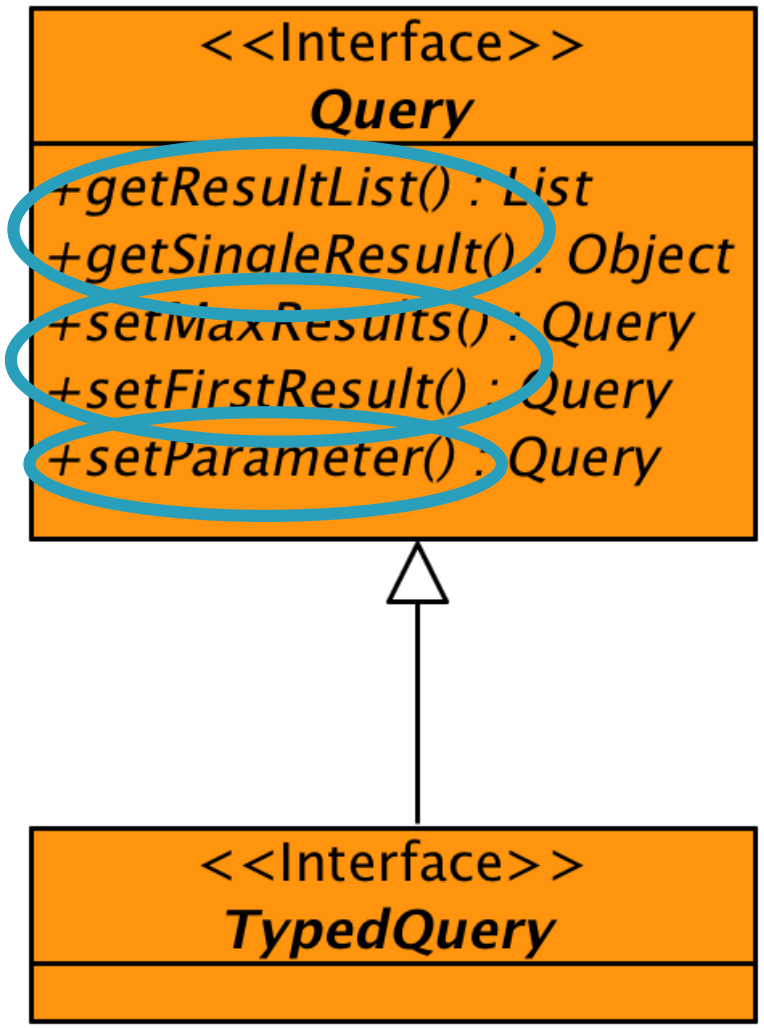
TypedQuery

EntityManager

EntityManager



Query and TypedQuery



Dynamic Query

```
Query query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700");
```

```
Book book = (Book) query.getSingleResult();
```

✗NonUniqueResultException

Dynamic Query

```
Query query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700");
```

```
List books = query.getResultList();
```

Dynamic TypedQuery

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700",
         Book.class);



List<Book> books = query.getResultList();
```

Stream TypedQuery

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700",
         Book.class);
```

```
Stream<Book> books = query.getResultList().stream();
```

Stream TypedQuery

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700",
         Book.class);


Stream<Book> books = query.getResultStream();
```

Binding Parameters

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700",
         Book.class);



List<Book> books = query.getResultList();
```

Binding Parameters

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > ?1 AND b.nbOfPage < ?2",
         Book.class);

query.setParameter(1, unitCost);
query.setParameter(2, nbOfPage);
List<Book> books = query.getResultList();
```

Binding Parameters

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.unitCost > :cost AND b.nbOfPage < :pages",
         Book.class);

query.setParameter("cost", unitCost);
query.setParameter("pages", nbOfPage);
List<Book> books = query.getResultList();
```

Binding Date Parameters

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate",
         Book.class);

query.setParameter("pubDate", LocalDate.now());

List<Book> books = query.getResultList();
```

Binding Legacy Date Parameters

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate",
         Book.class);

query.setParameter("pubDate", new Date(), DATE);

List<Book> books = query.getResultList();
```

Binding Legacy Date Parameters

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate",
         Book.class);

query.setParameter("pubDate", new Date(), TIMESTAMP);

List<Book> books = query.getResultList();
```

Pagination

```
TypedQuery<Book> query = em.createQuery(
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate",
         Book.class);

query.setParameter("pubDate", new Date(), TIMESTAMP);
query.setMaxResults(10);
List<Book> books = query.getResultList();
```

Why Dynamic?

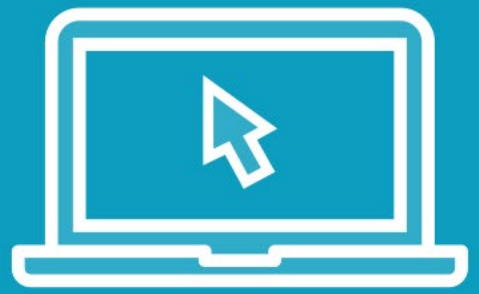
```
String statement = "SELECT b FROM Book b
                    WHERE  b.unitCost > :cost ";

if (hasPages)
  statement += "AND b.nbOfPage < :pages ";
if (hasDate)
  statement += "OR b.publicationDate < :pubDate";

TypedQuery<Book> query = em.createQuery(statement,Book.class);
query.setParameter("cost", unitCost);
query.setParameter("pages", nbOfPage);
query.setParameter("pubDate", publicationDate);

Stream<Book> books = query.getResultStream();
```

Demo



Query service

Dynamic queries

Query and TypedQuery APIs

Query parameters

Named Query

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
public class Book {
  // Attributes and Constructors
}
```

```
Query query = em.createNamedQuery("ExpensiveBooks");
```

✗NonUniqueResultException

```
Book book = (Book) query.getSingleResult;
```

Named Query

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
public class Book {
  // Attributes and Constructors
}
```

```
Query query = em.createNamedQuery("ExpensiveBooks");
```

```
List books = query.getResultList();
```

Named TypedQuery

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
public class Book {
  // Attributes and Constructors
}
```

```
TypedQuery<Book> query = em.createNamedQuery (
                         "ExpensiveBooks", Book.class);
```

```
List<Book> books = query.getResultList();
```

Stream Named TypedQuery

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
public class Book {
  // Attributes and Constructors
}
```

```
TypedQuery<Book> query = em.createNamedQuery (
                         "ExpensiveBooks", Book.class);
```

```
Stream<Book> books = query.getResultStream();
```

Binding Parameters

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > ?1 AND b.nbOfPage < ?2")
public class Book {
  // Attributes and Constructors
}
```

```
TypedQuery<Book> query = em.createNamedQuery (
                         "ExpensiveBooks", Book.class);
```

```
List<Book> books = query.getResultList();
```

Binding Parameters

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > :cost AND b.nbOfPage < :pages")
public class Book {
  // Attributes and Constructors
}
```

```
TypedQuery<Book> query = em.createNamedQuery (
                         "ExpensiveBooks", Book.class);
query.setParameter("cost", unitCost);
query.setParameter("pages", nbOfPage);

List<Book> books = query.getResultList();
```

Pagination

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > :cost AND b.nbOfPage < :pages")
public class Book {
  // Attributes and Constructors
}
```

```
TypedQuery<Book> query = em.createNamedQuery (
                         "ExpensiveBooks", Book.class);
query.setParameter("cost", unitCost);
query.setParameter("pages", nbOfPage);
query.setMaxResults(10);
List<Book> books = query.getResultList();
```

Multiple Named Queries

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
@NamedQuery(name = "PublishedBooks", query =
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate")
@NamedQuery(name = "All",          query = "SELECT b FROM Book b")
public class Book {



 // Attributes and Constructors
}
```

Unique Name

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
@NamedQuery(name = "PublishedBooks", query =
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate")
@NamedQuery(name = "All",          query = "SELECT b FROM Book b")
public class Book {



 // Attributes and Constructors
}
```

Unique Name

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
@NamedQuery(name = "PublishedBooks", query =
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate")
@NamedQuery(name = "Book.All",     query = "SELECT b FROM Book b")
public class Book {



 // Attributes and Constructors
}
```

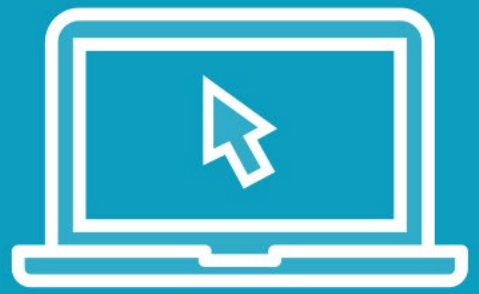
Unique Name

```
@Entity
@NamedQuery(name = "ExpensiveBooks", query =
        "SELECT b FROM Book b
         WHERE  b.unitCost > 29 AND b.nbOfPage < 700")
@NamedQuery(name = "PublishedBooks", query =
        "SELECT b FROM Book b
         WHERE  b.publicationDate < :pubDate")
@NamedQuery(name = Book.FIND_ALL, query = "SELECT b FROM Book b")
public class Book {

  public static final String FIND_ALL = "Book.All";

 // Attributes and Constructors
}
```

Demo



Query service

Named queries

On Book entity

Query or the TypedQuery APIs

Parameters

Summary

- Query entities
- Java Persistence Query Language
- Object oriented
- JPQL syntax
- Dynamic queries
- Named queries

Next Module

Entity life cycle

- Transient
- Managed
- Detached

Callback methods

Listeners

Business logic