



CS544 EA

Hibernate

JPQL: WHERE clause

WHERE Clause

- WHERE lets you add constraints to the result
 - Refining which rows end up in the list

```
TypedQuery<Person> query  
    = em.createQuery("from Person p where p.lastName = 'Johnson'", Person.class);
```

Selects all the people
whose last name is Johnson

- JPQL supports the same expressions as SQL
 - As well as some OO specific expressions

```
TypedQuery<Person> query  
    = em.createQuery("from Person p where p.accounts[0].balance > 100", Person.class);
```

People whose first account
has a balance is > 100

JPQL Expressions

Type	Operators
Literals	'string', 128, 4.5E+3, 'yyyy-mm-dd hh:mm:ss'
Arithmetic	+, -, *, /
Comparison	=, <>, >=, <=, !=, like
Logical	and, or, not
Grouping	(,)
Concatenation	
Values	in, not in, between, is null, is not null, is empty, is not empty
Case	case ... when ... then ... else ... end, case when ... then ... else ... end

JPQL Functions

- JPQL also provides several built-in functions
 - These work regardless of underlying DB

Type	Functions
Temporal	current_date(), current_time(), current_timestamp(), second(...), minute(...), hour(...), day(...), month(...), year(...)
String	concat(... , ...), substring(), trim(), lower(), upper(), length(), str()
Collection	Index(), size(), minindex(), maxindex()

Indexed Collection Expressions

- **[]** can be used to access indexed collections
 - Only: **Map** and **@OrderColumn List**

```
TypedQuery<Person> query  
    = em.createQuery("from Person p where p.accounts[0].balance > 100", Person.class);
```

Account list has to have
@OrderColumn

```
TypedQuery<Person> query  
    = em.createQuery("from Person p where p.pets['mimi'].species = 'Cat'", Person.class);
```

Map with String key

Query Parameters

- **Never concatenate** JPQL Strings!
 - Opens the door for **JPQL (SQL) injection**
 - Also makes your query messy

```
TypedQuery<Person> pplQuery  
    = em.createQuery("from Person p where p.firstName = '" + firstName + "'", Person.class);
```

- Use named parameters instead:

```
TypedQuery<Person> pplQuery  
    = em.createQuery("from Person p where p.firstName = :first", Person.class);  
pplQuery.setParameter("first", firstName);
```

Separates instruction
and data

Safely replace
placeholder

Placeholder

Temporal Parameters

- Specify the **exact type** for temporal types
 - Using either `java.util.Calendar` or `java.util.Date`
 - Java 8 `LocalDate` not yet supported

```
TypedQuery<Person> q  
    = em.createQuery("from Person p where p.birthDate < :date", Person.class);  
Calendar cal = Calendar.getInstance();  
cal.set(2000, 0, 1); // 2000-01-01  
q.setParameter("date", cal, TemporalType.DATE);
```

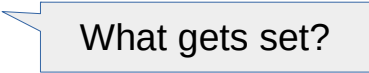
Overloaded to receive
`java.util.Date` or `java.util.Calendar`

Specify the temporal type

Positional Parameters

- Possible but **not recommended**
 - Uses ? as placeholder instead of unique names
 - Easily breaks if you add more parameters later
 - A lot less self documenting!

```
TypedQuery<Person> q
    = em.createQuery("from Person where firstName = ? and lastName = ?", Person.class);
q.setParameter(0, "Jackson");
q.setParameter(1, "Jarvis");
List<Person> ppl = q.getResultList();
```



.singleResult()

- Returns a single object instead of a List
 - Make sure there is **exactly one** result!
 - NoResultException, NonUniqueResultException

```
TypedQuery<Person> q = em.createQuery("from Person where id = 1", Person.class);  
Person p = q.getSingleResult();
```

Guaranteed to be single result

```
TypedQuery<Person> q2 = em.createQuery("from Person", Person.class);  
q2.setMaxResults(1);  
Person p2 = q2.getSingleResult();
```

Guaranteed to be single result

Special Attribute: .id

- Your @Id property can be referred to as .id
 - **Even if it's called something else**
 - Except if another property (not @Id) is called id

```
TypedQuery<Employee> q = em.createQuery("from Employee where id = 1", Employee.class);  
Employee e = q.getSingleResult();
```

```
@Entity  
public class Employee {  
    @Id  
    @GeneratedValue  
    private Long employeeId;  
    private String firstName;  
    private String lastName;
```

Special Attribute: .class

- You can **compare the class name** with .class
 - To restrict to a certain class with =
 - Or remove a certain class with != / <>

```
List<Account> accounts = em.createQuery("from Account a "  
    + "where a.class <> CheckingAccount "  
    + "and a.owner.firstName = 'Frank'", Account.class)  
    .getResultList();
```

- The type() function does the same

```
List<Account> accounts = em.createQuery("from Account a "  
    + "where type(a) = CheckingAccount "  
    + "and a.owner.firstName = 'Frank'", Account.class)  
    .getResultList();
```