



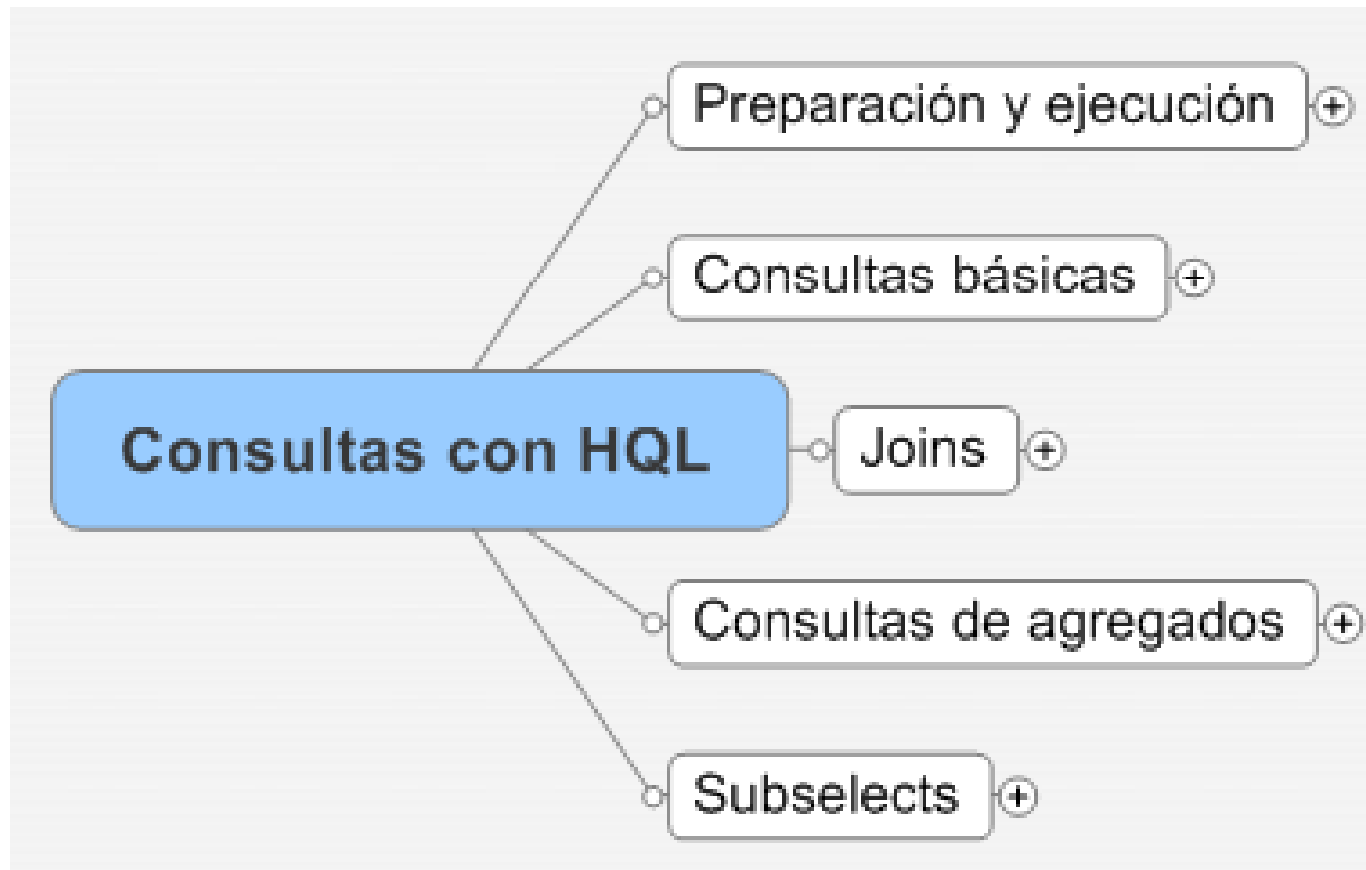
# Consultas en JPA-QL

---

Sistemas de Persistencia de  
Objetos



# Contenidos



# Preparación y ejecución





# Creación

---

## ■ HQL y JPA QL

```
session.createQuery("from Category c where c.name like 'Laptop%'");  
entityManager.createQuery(  
    "select c from Category c where c.name like 'Laptop%'"  
);
```

## ■ QBC y QBE (Query By Criteria)

```
session.createCriteria(Category.class)  
    .add( Restrictions.like("name", "Laptop%") );
```

## ■ SQL Directo

```
session.createSQLQuery(  
    "select {c.*} from CATEGORY {c} where NAME like 'Laptop%'"  
).addEntity("c", Category.class);
```

# Paginación

```
List list = em.createQuery("select u from User u")  
    .setFirstResult(10)  
    .setMaxResults(20)  
    .getResultList();
```

El primer resultado es el 0

Las Query permiten  
encadenamiento de  
métodos

Número máximo de filas a  
recuperar desde la fijada por  
`setFirstResult()`

Ejecuta la consulta y  
devuelve una `List()`  
de objetos `User`

```
for (Object o : list) {  
    User u = (User) o;  
    System.out.println(u);  
}
```



# Enlace de parámetros

- Lo que no se debe hacer

```
String queryString =  
    "from Item i where i.description like '" + search + "'";  
List result = em.createQuery(queryString).getResultList();
```

¿Qué hay en este string?

```
search = "foo' and callSomeStoredProcedure() and 'bar' = 'bar"
```

¿Qué pasa si  
escriben esto en  
un formulario?

Es el problema de la **SQL injection**



# Enlace de parámetros

## ■ Enlace nominal (recomendado)

```
String queryString = "select item from Item item"
                    + " where item.description like :search"
                    + " and item.date > :minDate";

Query q = em.createQuery(queryString)
        .setParameter("search", searchString)
        .setParameter("minDate", mDate, TemporalType.DATE);
```

setParameter() sobrecargado para  
java.util.Date, java.util.Calendar y  
Object (ver documentación)

# Enlace de parámetros

## ■ Enlace posicional

```
String queryString = "select item from Item item"  
+ " where item.description like ?1"  
+ " and item.date > ?2";
```

```
Query q = em.createQuery(queryString)  
    .setParameter(1, searchString)  
    .setParameter(2, minDate, TemporalType.DATE);
```

El orden de parámetros  
no tiene por qué ser  
secuencial

¡Ojo! Se empieza en 1

setters  
sobrecargados





# Ajustes de rendimiento

Hint	Description
org.hibernate.timeout	Query timeout in seconds ( eg. new Integer(10) )
org.hibernate.fetchSize	Number of rows fetched by the JDBC driver per roundtrip ( eg. new Integer(50) )
org.hibernate.comment	Add a comment to the SQL query, useful for the DBA ( e.g. new String("fetch all orders in 1 statement") )
org.hibernate.cacheable	Whether or not a query is cacheable ( eg. new Boolean(true) ), defaults to false
org.hibernate.cacheMode	Override the cache mode for this query ( eg. CacheMode.REFRESH )
org.hibernate.cacheRegion	Cache region of this query ( eg. new String("regionName") )
org.hibernate.readOnly	Entities retrieved by this query will be loaded in a read-only mode where Hibernate will never dirty-check them or make changes persistent ( eg. new Boolean(true) ), default to false
org.hibernate.flushMode	Flush mode used for this query
org.hibernate.cacheMode	Cache mode used for this query



# Ajustes de rendimiento

---

```
Query q = em.createQuery(queryString)
    .setFlushMode(FlushModeType.COMMIT);
```

```
Query q = em.createQuery(queryString)
    .setHint("org.hibernate.cacheMode",
        org.hibernate.CacheMode.IGNORE);
```

- Los “hints” son todos dependientes de implementación, no estándar JPA
- Si un hint no es soportado es ignorado silenciosamente



# Hints: ejemplos

---

```
Query q = em.createQuery(queryString)
    .setFlushMode(FlushModeType.COMMIT);
```

```
Query q = em.createQuery(queryString)
    .setHint("org.hibernate.cacheMode",
        org.hibernate.CacheMode.IGNORE);
```

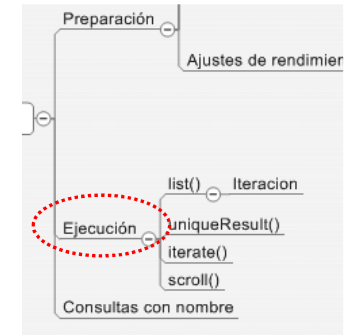
```
Query q = em.createQuery("select i from Item i")
    .setHint("org.hibernate.readOnly", true);
```

```
Query q = em.createQuery("select i from Item i")
    .setHint("org.hibernate.timeout", 60);
```

```
Query q = em.createQuery("select i from Item i")
    .setHint("org.hibernate.fetchSize", 50);
```

```
Query q = em.createQuery("select i from Item i")
    .setHint("org.hibernate.comment", "My Comment...");
```

# Ejecución



- Se produce al invocar a:

- `getResultList()`

- `getSingleResult()`

Excepción si más de uno o ninguno

```
Bid maxBid = (Bid) em.createQuery(  
    "select b from Bid b order by b.amount desc"  
).setMaxResults(1)  
.getSingleResult();
```

Así ya no... pero puede no haber ninguno



# Consultas con nombre

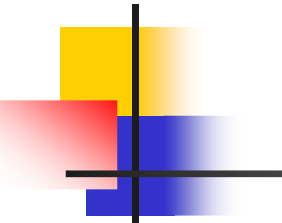
---

- Se carga el string de la consulta desde mapeos

- **createNamedQuery(...)**

```
em.createNamedQuery("findItemsByDescription")  
    .setParameter("desc", description);
```

- Query con anotaciones o en **orm.xml**



```
@NamedQueries({
    @NamedQuery(
        name = "findItemsByDescription",
        query = "select i from Item i where i.description like :desc"
    ),
    ...
})
@Entity
@Table(name = "ITEM")
public class Item { ... }
```

```
<entity-mappings ...>
```

```
...
```

```
<named-query name="findAllItems">
```

```
    <query>select i from Item i</query>
```

```
</named-query>
```

```
<entity class="Item">
```

```
...
```

```
<named-query name="findItemsByDescription">
```

```
    <query>
```

```
        select i from Item i where i.description like :desc
```

```
    </query>
```

```
    <hint name="org.hibernate.comment" value="My Comment"/>
```

```
    <hint name="org.hibernate.fetchSize" value="50"/>
```

```
    <hint name="org.hibernate.readOnly" value="true"/>
```

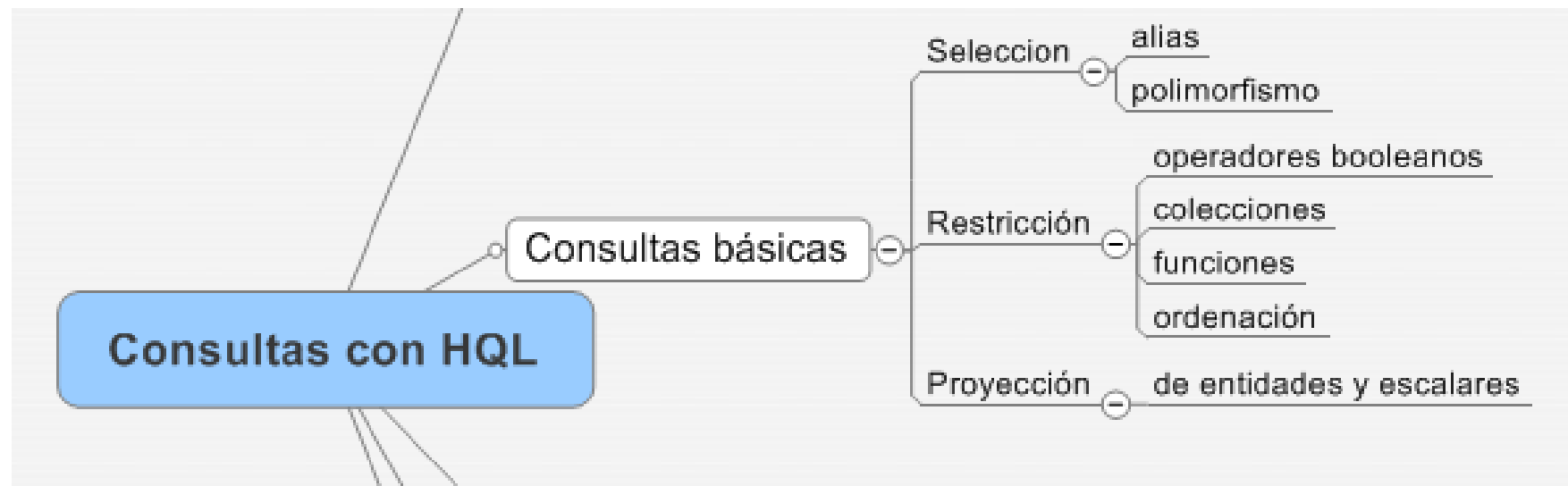
```
    <hint name="org.hibernate.timeout" value="60"/>
```

```
    </named-query>
```

```
</entity>
```

```
</entity-mappings>
```

# Consultas básicas





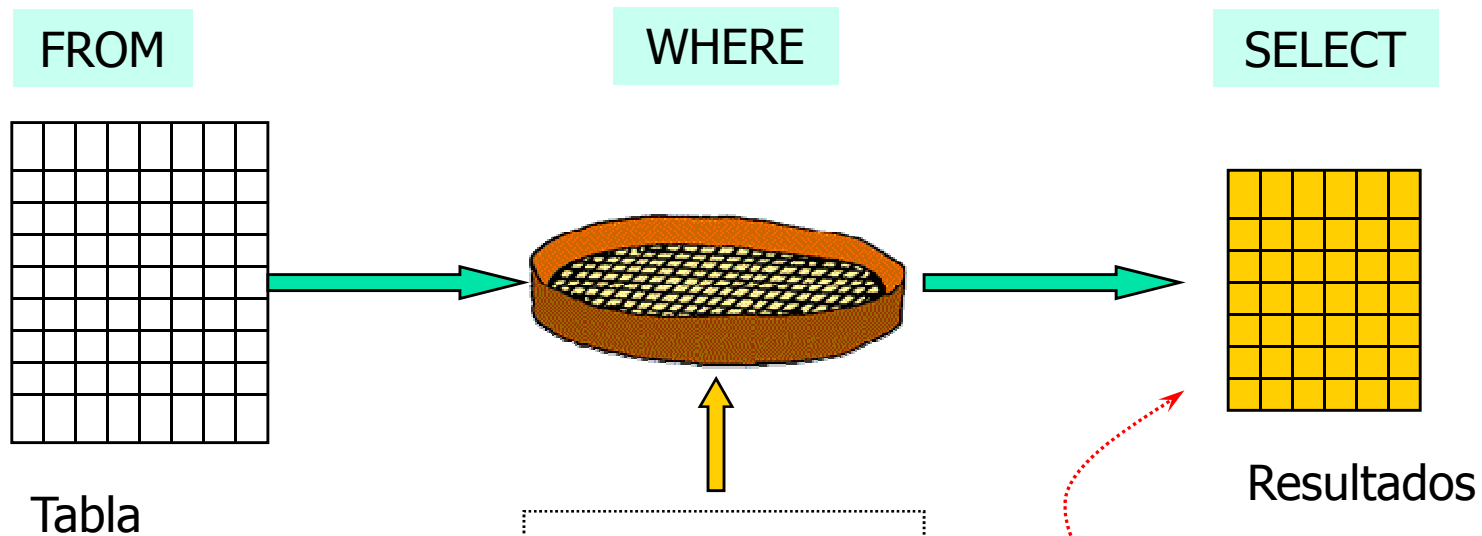
# Partes de una consulta

---

- Selección
  - Fuente de datos → FROM
  - Una sola o combinación de ellas
- Restricción
  - Filtrado de filas → WHERE
- Proyección
  - Selección de partes de las filas que pasan el filtro → SELECT



# Partes de una consulta



Puede que haya menos filas (WHERE) y puede que menos campos (SELECT)



# Selección (FROM)

- SELECT en JPA QL, no necesario en HQL

- `select i from Item i`

- Alias necesarios para condiciones sobre miembros

- `select i from Item as i`

- `select i from Item i`

- Las consultas son polimórficas

- `select b from BillingDetail b`

- `select o from java.lang.Object o`

- `select s from java.io.Serializable s`

¡Sube toda la BDD!

También  
polimorfismo sobre  
interfaces



# Restricción (WHERE)

## ■ WHERE para filtrar filas

```
select u from User u where u.email = 'foo@hibernate.org'
```

```
select i from Item i where i.isActive = true
```

```
select bid from Bid bid where bid.amount between 1 and 10
```

```
select bid from Bid bid where bid.amount > 100
```

```
select u from User u where u.email in ('foo@bar', 'bar@foo')
```

```
select u from User u where u.email is null
```

```
select i from Item i where i.successfulBid is not null
```

```
select u from User u where u.firstname like 'G%'
```

```
select u from User u where u.firstname not like '%Foo B%'
```

```
select u from User u where u.firstname not like '\\%Foo%' escape='\\'
```



# Restricción (WHERE)

---

```
select u from User u where u.firstname like 'G%'
```

```
select u from User u where u.firstname not like '%Foo B%'
```

```
select u from User u where u.firstname not like '%Foo%' escape='\'
```

```
select bid from Bid bid where ( bid.amount / 0.71 ) - 100.0 > 0.0
```

```
select user from User user
    where user.firstname like 'G%' and user.lastname like 'K%'
```

```
select u from User u
    where ( u.firstname like 'G%' and u.lastname like 'K%' )
    or u.email in ('foo@hibernate.org', 'bar@hibernate.org' )
```

# Operadores de comparación y precedencia

Operator
.
+, -
*, /
+, -
=, <>, <, >, >=, <=, [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL, IS [NOT] EMPTY, [NOT] MEMBER [OF]
NOT, AND, OR





# Restricciones sobre colecciones (WHERE)

- En el WHERE
- Se pueden complementar con funciones

```
select i from Item i where i.bids is not empty
```

```
select i, c from Item i, Category c where i.id = '10' and i member of c.items
```

```
select u from User u where lower(u.email) = 'foo@hibernate.org'
```

```
select user from User user  
    where concat(user.firstname, user.lastname) like 'G% K%'
```

```
select i from Item i where size(i.bids) > 3
```



# Funciones

Hibernate	Function
	BIT_LENGTH(s)
	CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP()
	SECOND(d), MINUTE(d), HOUR(d), DAY(d), MONTH(d), YEAR(d)
	CAST(t as Type)
	INDEX(joinedCollection)
	MINELEMENT(c), MAXELEMENT(c), MININDEX(c), MAXINDEX(c), ELEMENTS(c), INDICES(c)

Registered in org.hibernate.Dialect

JPA	Function
	UPPER(s), LOWER(s)
	CONCAT(s1, s2)
	SUBSTRING(s, offset, length)
	TRIM( [[BOTH LEADING TRAILING] char [FROM]] s)
	LENGTH(s)
	LOCATE(search, s, offset)
	ABS(n), SQRT(n), MOD(dividend, divisor)
	SIZE(c)

```
from User u where lower(u.email) = 'foo@hibernate.org'
```

```
from User user
    where concat(user.firstname, user.lastname) like 'G% K%'
```

```
from Item i where size(i.bids) > 3
```



# Ordenación

---

- De la forma usual

```
select u from User u order by u.username
```

```
select u from User u order by u.username desc
```

```
select u from User u order by u.lastname asc, u.firstname asc
```



# Proyección



```
Query q = em.createQuery("select i, b from Item i, Bid b");
```

```
Iterator pairs = q.getResultList().iterator();
```

```
while ( pairs.hasNext() ) {
```

```
    Object[] pair = (Object[]) pairs.next();
```

```
    Item item = (Item) pair[0];
```

```
    Bid bid = (Bid) pair[1];
```

```
    // . . .
```

```
}
```

(Esta consulta es inútil ya que da un producto cartesiano)

Cada fila es un vector de los elementos proyectados (Item y Bid)



# Proyección de escalares

```
String query = "select i.id, i.description, i.initialPrice" +  
               "from Item i " +  
               "where i.endDate > current_date()";
```

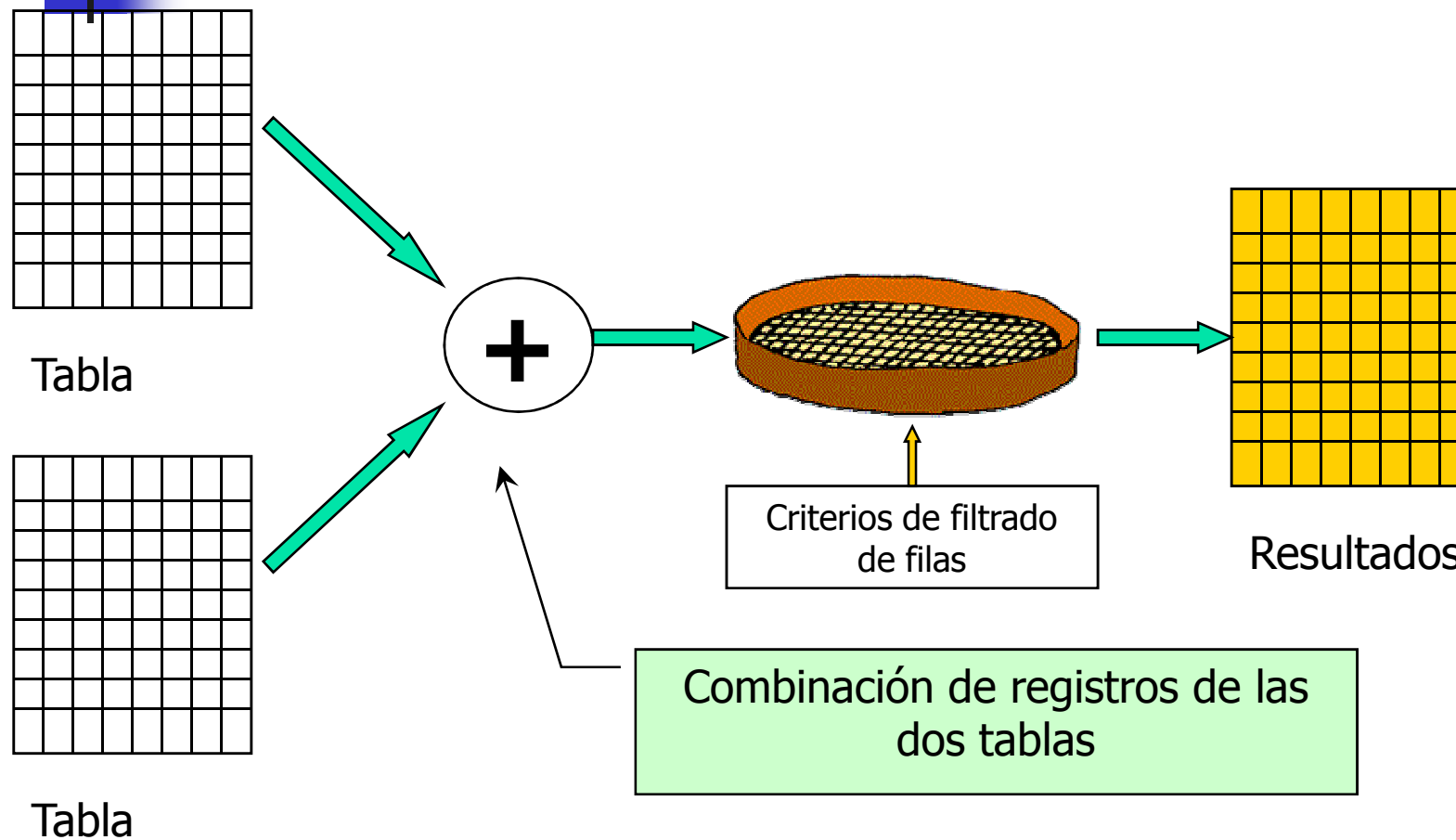
```
List list = em.createQuery(query)  
               .getResultList();  
  
for (Object o: list) {  
    Object[] pair = (Object[]) o;  
    Long id = (Long) pair[0];  
    String desc = (String) pair[1];  
    BigDecimal price = (BigDecimal) pair[2];  
  
    //...  
}
```

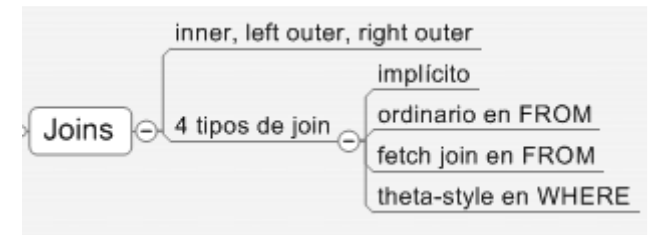
En la select pueden ir  
atributos de clases...

... y resultados de funciones  
(las ya vistas)

```
select distinct item.description from Item item  
select item.startDate, current_date() from Item item  
  
select item.startDate, item.endDate, upper(item.name)  
from Item item
```

# Consulta sobre varias tablas





# Joins: inner, left y right outer

ITEM			BID		
ITEM_ID	DESCRIPTION	...	BID_ID	ITEM_ID	AMOUNT
1	Item Nr. One	...	1	1	99.00
2	Item Nr. Two	...	2	1	100.00
3	Item Nr. Three	...	3	1	101.00
			4	2	4.99

Todos los Items con sus Bids

**select i.\*, b.\* from ITEM i  
inner join BID b on i.ITEM\_ID = b.ITEM\_ID**

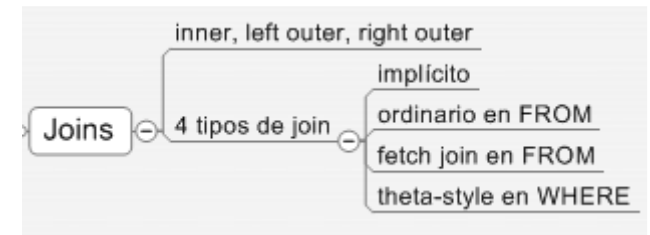
ITEM_ID	DESCRIPTION	...	BID_ID	ITEM_ID	AMOUNT
1	Item Nr. One	...	1	1	99.00
1	Item Nr. One	...	2	1	100.00
1	Item Nr. One	...	3	1	101.00
2	Item Nr. Two	...	4	2	4.99

Los Items que tienen Bids

**select i.\*, b.\* from ITEM i  
left outer join BID b on i.ITEM\_ID = b.ITEM\_ID**

ITEM_ID	DESCRIPTION	...	BID_ID	ITEM_ID	AMOUNT
1	Item Nr. One	...	1	1	99.00
1	Item Nr. One	...	2	1	100.00
1	Item Nr. One	...	3	1	101.00
2	Item Nr. Two	...	4	2	4.99
3	Item Nr. Three	...	NULL	NULL	NULL

# Joins implícitos en asociaciones



- Cuando se accede a propiedades a lo largo de un camino (path)

```
select bid from Bid bid where bid.item.description like '%Foo%'
```

```
select bid from Bid bid where bid.item.buyer.firstname like '%name%'
```

Bid join Item

Item join User

Acceso a propiedad

```
select distinct bid.item.buyer.firstname from Bid bid
```

También se puede usar en **select**



# Joins implícitos

---

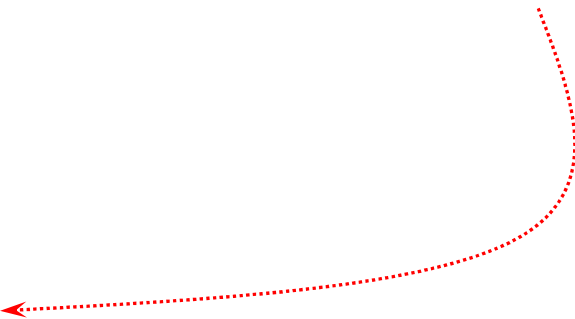
- Solo se permiten en caminos (path) que pasen a través de asociaciones **many-to-one** o **one-to-one**
- El final del camino **NO puede ser multivaluado**
  - P.e. **item.bids.amount** es ilegal

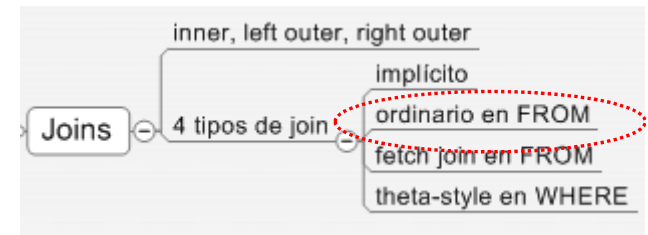


# Joins implícitos en SQL

```
select bid from Bid bid
where bid.item.seller.address.city = 'Oviedo'

select
  bid0_.id as id10_,
  bid0_.currency as currency10_,
  bid0_.value as value10_,
  bid0_.bidder_id as bidder6_10_,
  bid0_.created as created10_,
  bid0_.item_id as item7_10_,
  bid0_.successful as successful10_
from
  Bid bid0_,
  Item item1_,
  User user2_
where
  bid0_.item_id=item1_.id
and item1_.seller_id=user2_.id
and user2_.city='Oviedo'
```





# Joins en FROM

- Cuando el camino de asociaciones resulta en un conjunto

```
select b from Bid b join b.item.categories c
where c.name like 'A.%'
```

many-to-many

```
select i from Item i join i.bids b
where i.description like '%Foo%'
and b.amount > 100
```

one-to-many





# Joins en FROM

---

- También left y right join

```
select i, b from Item i left join i.bids b  
  where i.description like '%name%'
```

Los Item `%name%` y sus  
Bids aunque haya Item  
que no tienen Bids

```
select i, b from Bid b right join b.item i  
  where i.description like '%name%'
```



# Join explícito en SQL

**select**

```
item0_.id as id8_0_  
bids1_.id as id10_1_  
item0_.buyer_id as buyer12_8_0_  
item0_.created as created8_0_  
item0_.description as descript3_8_0_  
item0_.endDate as endDate8_0_  
item0_.initialPrice_currency as initialP5_8_0_  
item0_.initialPrice_value as initialP6_8_0_  
item0_.name as name8_0_  
item0_.reservePrice_currency as reserveP8_8_0_  
item0_.reservePrice_value as reserveP9_8_0_  
item0_.seller_id as seller14_8_0_  
item0_.startDate as startDate8_0_  
item0_.successfulBid_id as success13_8_0_  
item0_.version as version8_0_  
bids1_.currency as currency10_1_  
bids1_.value as value10_1_  
bids1_.bidder_id as bidder6_10_1_  
bids1_.created as created10_1_  
bids1_.item_id as item7_10_1_  
bids1_.successful as successful10_1_
```

**from**

Item item0\_

**left outer join**

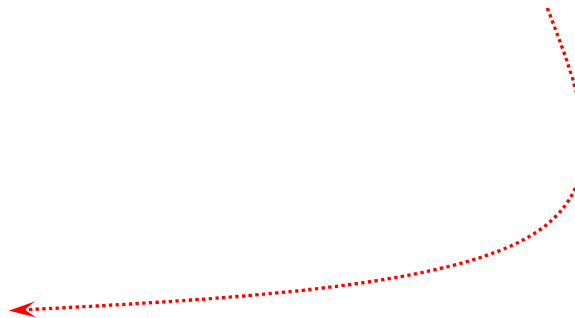
Bid bids1\_

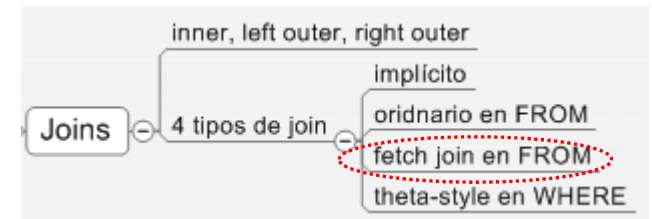
**on** item0\_.id=bids1\_.item\_id

**where**

item0\_.description **like** '%name%'

```
select i, b from Item i left join i.bids b  
where i.description like '%name%'
```





# Fetch join en FROM

- Salvo mapeo en contra todas las colecciones se cargan **lazy**
- La configuración de mapeo se puede sobrecargar para una consulta concreta si se usa **fetch join** para colecciones
- El efecto es que se cargan todos los elementos de la colección asociada al momento (**eager fetching**)
- Es un ajuste fundamental en el rendimiento de algunas consultas

# Fetch join en FROM

```
select i, b
  from Item i left join fetch i.bids b
 where i.description like '%name%'
```

```
List list = em.createQuery(
    "select i " +
    "from Item i left join fetch i.bids" +
    "where i.description like '%1%'"
).getResultList();
```

```
for (Object o: list) {
    Item i = (Item) o;
    for (Bid b: i.getBids()) {
        // . . .
    }
}
```

Se cargan los Item que pasan la restricción y sus colecciones asociadas de Bids de forma agresiva (eager), no lazy

El resultado es una List() de Item. Ya no son pares.



# Fetch join en FROM

- También se puede usar para cargar de forma agresiva el extremo **one** de asociaciones **one-to-one** y **many-to-one**

```
from Bid bid
  left join fetch bid.item
  left join fetch bid.bidder
where bid.amount > 100
```

many-to-one

Si no pone **left** también carga de forma agresiva item y bidder pero solo los bids que tienen item y bidder

Nota: JPA por defecto carga eager asociaciones **...-to-one**, es útil si hay mapeo fetch=**LAZY** y se quiere forzar



# Fetch join: recovecos

---

- No se puede usar un alias en SELECT ni WHERE
- No se puede hacer **fetch join** más de una colección (problema del producto cartesiano)
- La estrategia del mapeo se ignora
- Se pueden cargar duplicados
- **setMaxResults(...)** y **setFirstResult(...)** se desaconsejan

# Fetch join recovecos

from Item i  
left join fetch i.bids b  
where b = ...



select b  
from Item i  
left join fetch i.bids b



✓ left join fetch i.bids b join fetch b.bidder

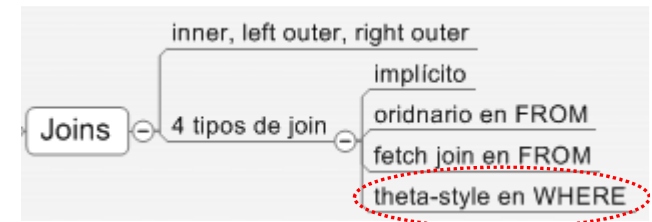
No se puede usar  
un alias en SELECT  
ni WHERE

select distinct i  
from Item i  
left join fetch i.bids b join fetch b.bidder

Se pueden cargar  
duplicados, para evitarlos ...

```
List list = em.createQuery(  
    "select i " +  
    "from Item i left join fetch i.bids b " +  
    "where i.description like '%1%'" )  
    .getResultList();
```

```
for (Object o: new HashSet(list)) {  
    Item i = (Item) o;  
    for (Bid b: i.getBids()) {  
        // ...  
    }  
}
```



# Theta-style en WHERE

- El ajuste del join se hace en el WHERE
- Es práctico para consultas sobre clases no asociadas

```
Iterator i = em.createQuery(  
    "from User user, LogRecord l  
    " where user.username = log.  
    .getResultList()  
    .iterator ();  
while ( i.hasNext() ) {  
    Object[] pair = (Object[]) i.next();  
    User user = (User) pair[0];  
    LogRecord log = (LogRecord) pair[1];
```

```
select user, log  
from User user, LogRecord log  
where user.username = log.username
```

Da pares



# Comparación de identificadores

```
select i from Item i, User u
  where i.seller = u and u.username = 'steve'
```

```
select i from Item i, User u
  where i.seller.id = u.id and u.username = 'steve'
```


```
select i
  from Item i join i.seller u
  where u.username = 'steve'
```

equivalentes

```
select b from Bid b where b.item.id = 1
select b from Bid b where b.item.description like '%Foo%'
```

Diferencia: la primera no carga Item,  
la segunda sí

# Comparación de id en ejecución



```
Query q = em.createQuery(
    "select c from Comment c " +
    "where c.fromUser.id = :user"
);
q.setParameter("user", new Long(1));
```

```
List result = q.getResultList();
for (Object o: result) {
    Comment c = (Comment) o;
    // . . .
}
```

```
User givenUser = em.find(User.class, new Long(1));
Query q = em.createQuery(
    "select c from Comment c " +
    "where c.fromUser = :user"
);
q.setParameter("user", givenUser);
```

```
List result = q.getResultList();
for (Object o: result) {
    Comment c = (Comment) o;
    // . . .
}
```



# Consultas de agregados

---





# Funciones en SELECT

---

**count()** **min()** **max()** **sum()** **avg()**

```
select count(i) from Item i
```

```
select count(i.successfulBid) from Item i
```

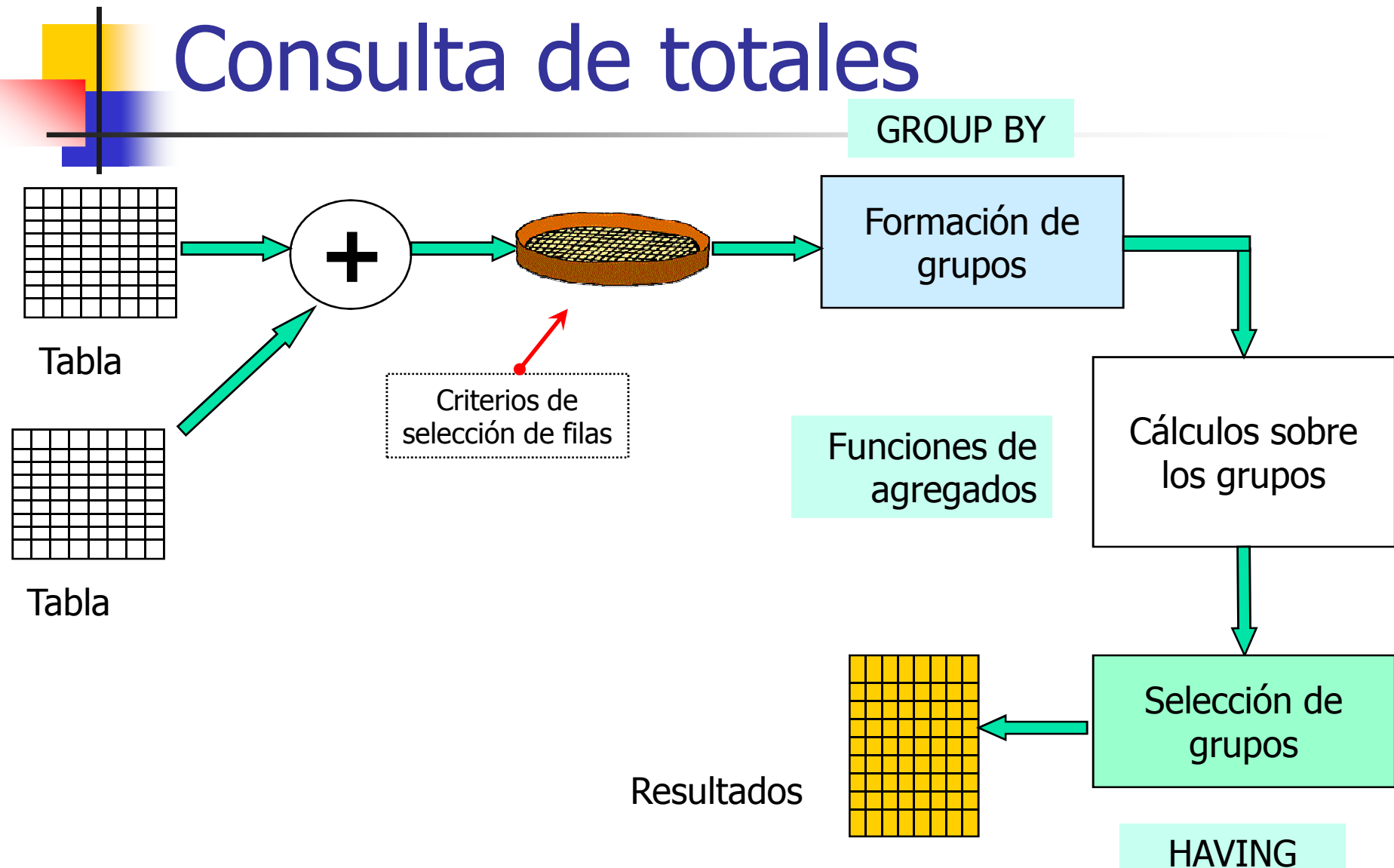
```
select sum(i.successfulBid.amount) from Item i
```

```
select min(bid.amount), max(bid.amount)
       from Bid bid where bid.item.id = 1
```

```
select count(distinct i.description) from Item i
```

```
String query = "select count(i) from Item i";
Long count = (Long) session.createQuery(query)
                        .uniqueResult();
```

# Consulta de totales



# Agrupamiento

## ■ Cláusula GROUP BY (como en SQL)

```
select u.lastname, count(u)  
  from User u  
  group by u.lastname
```

```
select bid.item.id, avg(bid.amount)  
  from Bid bid  
  group by bid.item.id
```

```
select bid.item.id, count(bid), avg(bid.amount)  
  from Bid bid  
  where bid.item.successfulBid is null  
  group by bid.item.id
```

Como en SQL cualquier propiedad o alias que aparezca en **SELECT** fuera de una función de agregado debe aparecer también en la cláusula **GROUP BY**



# Restricción de grupos con HAVING

- Mismas reglas que en SQL

```
select user.lastname, count(user)
  from User user
 group by user.lastname
 having user.lastname like 'A%'

select item.id, count(bid), avg(bid.amount)
  from Item item
    join item.bids bid
 where item.successfulBid is null
 group by item.id
 having count(bid) > 10
```

Solo puede aparecer en **HAVING** una función de agregado o una propiedad (o alias) usado en **GROUP BY**

# Instanciación dinámica en SELECT

```
select new ItemBidSummary(  
    bid.item.id, count(bid), avg(bid.amount)  
)  
from Bid bid  
where bid.item.successfulBid is null  
group by bid.item.id
```

Las consultas que no devuelven entidades pueden tener rendimiento al no meter resultados en contexto de persistencia

- Cada fila devuelve un objeto de la clase que se especifica
- La clase debe existir y no necesita estar mapeada





# Subselects

---

- En SQL una subselect puede ir en SELECT, FROM o WHERE
- En JPA QL sólo puede ir en el WHERE
- Las debe soportar la BDD
  - MySQL en versiones anteriores a 4.?? no tiene subselects

# Subselects

```
select u
  from User u
 where 10 < (
    select count(i)
    from u.items i
    where i.successfulBid is not null
  )
```

Correlada: puede tener peor rendimiento

```
select bid
  from Bid bid
 where bid.amount + 1 >= (
    select max(b.amount)
    from Bid b
  )
```

No correlada: no tiene impacto de rendimiento

Siempre entre paréntesis



# Cuantificación

---

- Una subselect puede devolver una sola fila (normalmente agregados) o varias
- En el caso de varias se usan con cuantificación
  - ALL, ANY (o SOME), IN



# Cuantificación ejemplos

---

```
from Item i
  where 100 > all (
    select b.amount
    from i.bids b)
```

```
from Item i
  where 100 <= any (
    select b.amount
    from i.bids b)
```

```
from Item i
  where 100 = some (
    select b.amount
    from i.bids b)
```

```
from Item i
  where 100 in (
    select b.amount
    from i.bids b)
```

# Funciones con subselect implícitas

---

SIZE(c)

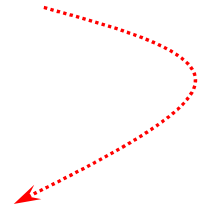
---

MINELEMENT(c), MAXELEMENT(c),  
MININDEX(c), MAXINDEX(c),  
ELEMENTS(c), INDICES(c)

---

```
from Category c
  where :givenItem in elements(c.items)
```

```
from Category c
  where :givenItem in
    (select i from c.items i)
```



```
List result =
  session.createQuery(
    "from Category c " +
    "where :givenItem in " +
    "elements(c.items)")
    .setEntity("givenItem", item)
    .list();
```

