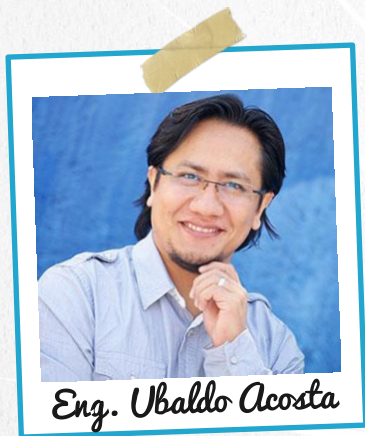


HIBERNATE & JPA COURSE

HQL AND JPQL QUERIES



By the expert: Eng. Ubaldo Acosta

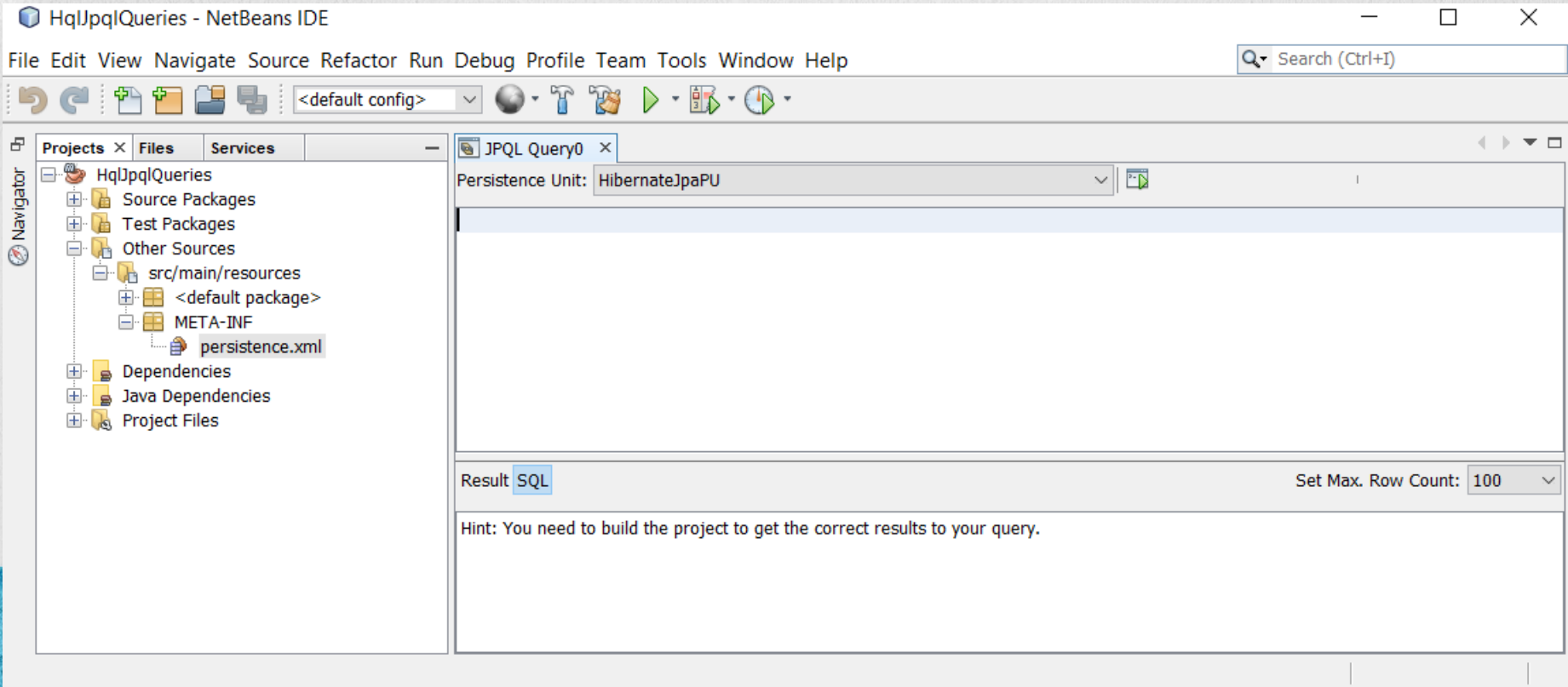


HIBERNATE & JPA COURSE

www.globalmentoring.com.mx

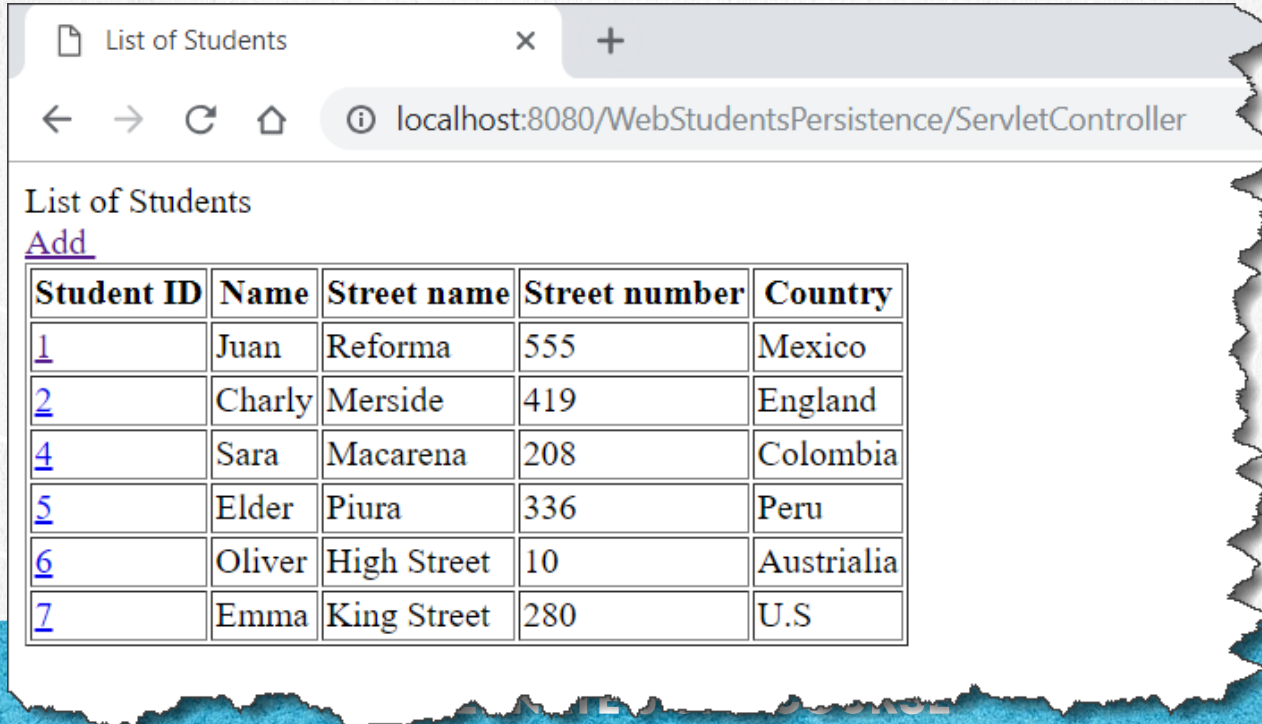
EXERCISE OBJECTIVE

Create queries with HQL / JPQL. At the end we should see:



ADD DATA TO THE DATABASE SMS_DB

We add some records to the database using the Web application that we created previously or directly in the mysql database of sms_db, the values may vary, but as an example are:



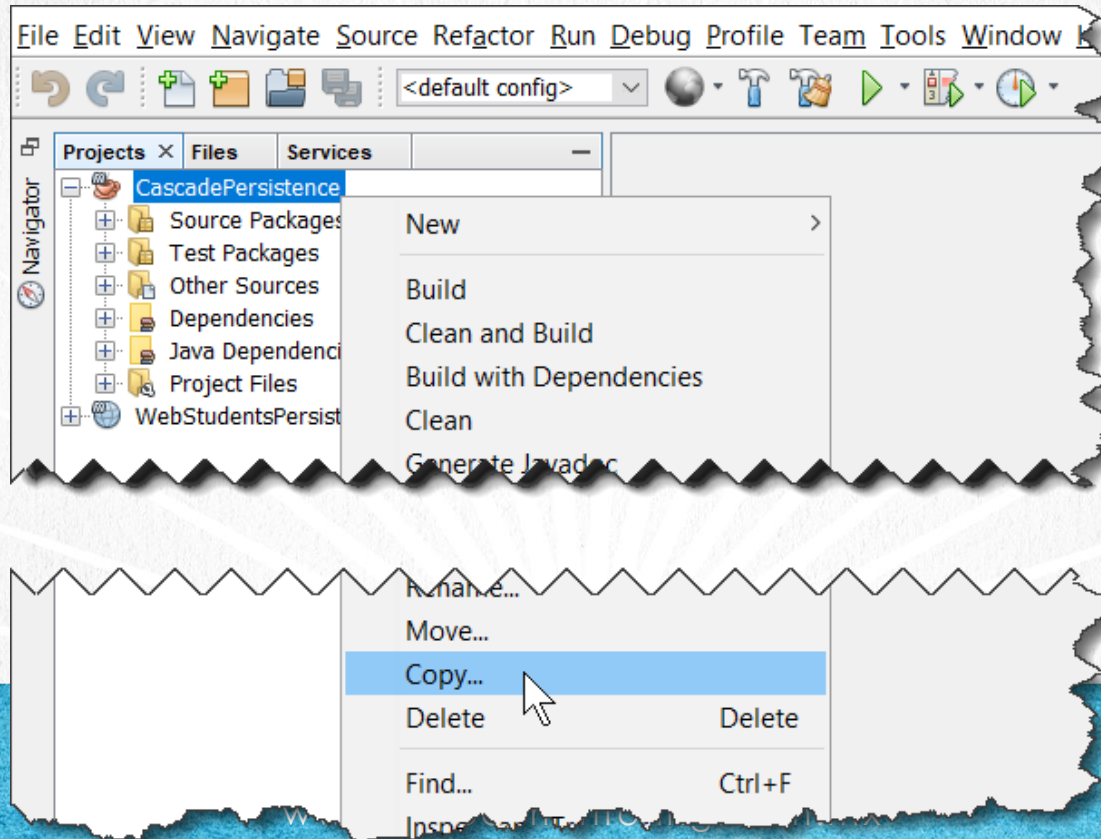
List of Students

[Add](#)

Student ID	Name	Street name	Street number	Country
1	Juan	Reforma	555	Mexico
2	Charly	Merside	419	England
4	Sara	Macarena	208	Colombia
5	Elder	Piura	336	Peru
6	Oliver	High Street	10	Australia
7	Emma	King Street	280	U.S

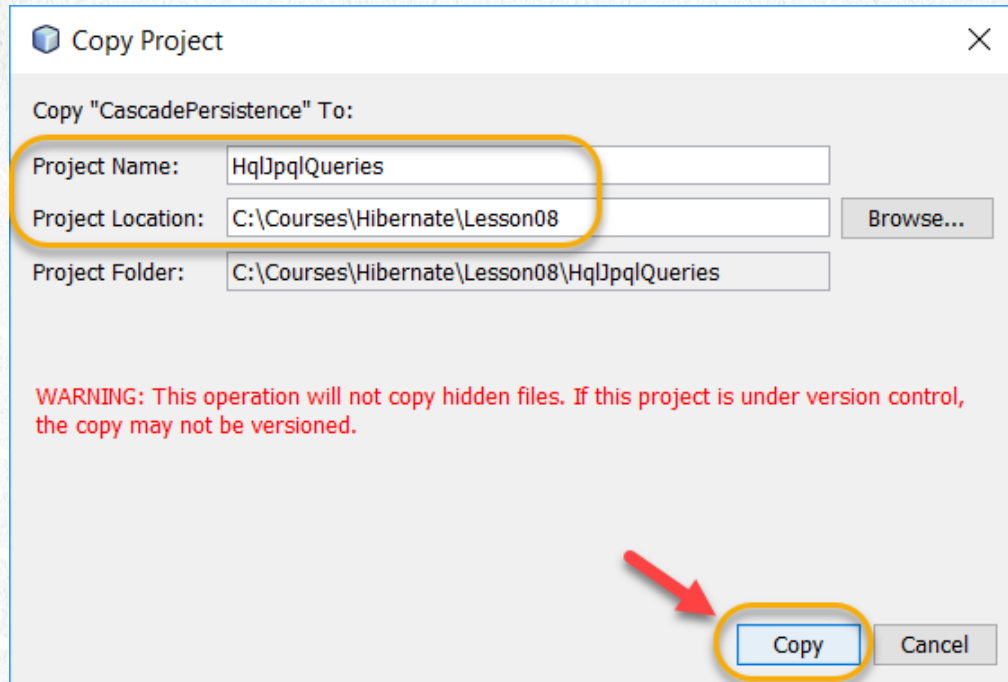
1. COPY THE PROJECT

We copy the project CascadePersistence:



1. COPY THE PROJECT

We created the new project HqJpqlQueries:

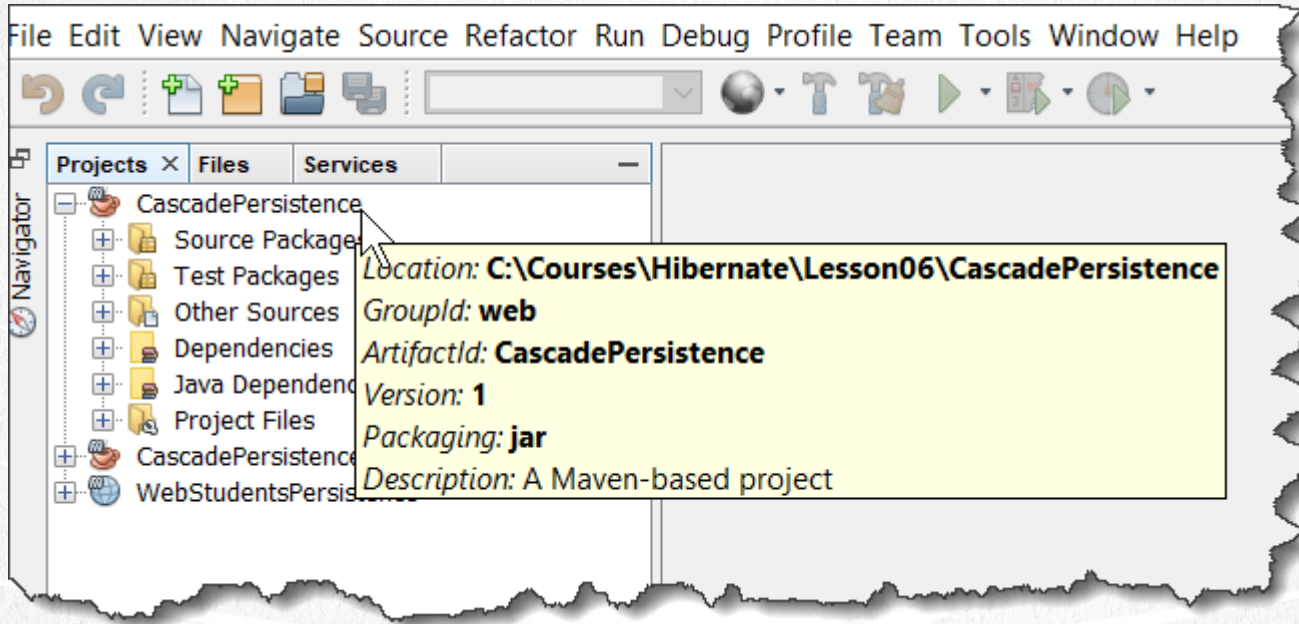


HIBERNATE & JPA COURSE

www.globalmentoring.com.mx

2. CLOSED THE PROJECT NOT USED ANYMORE

We close the project that we no longer use:

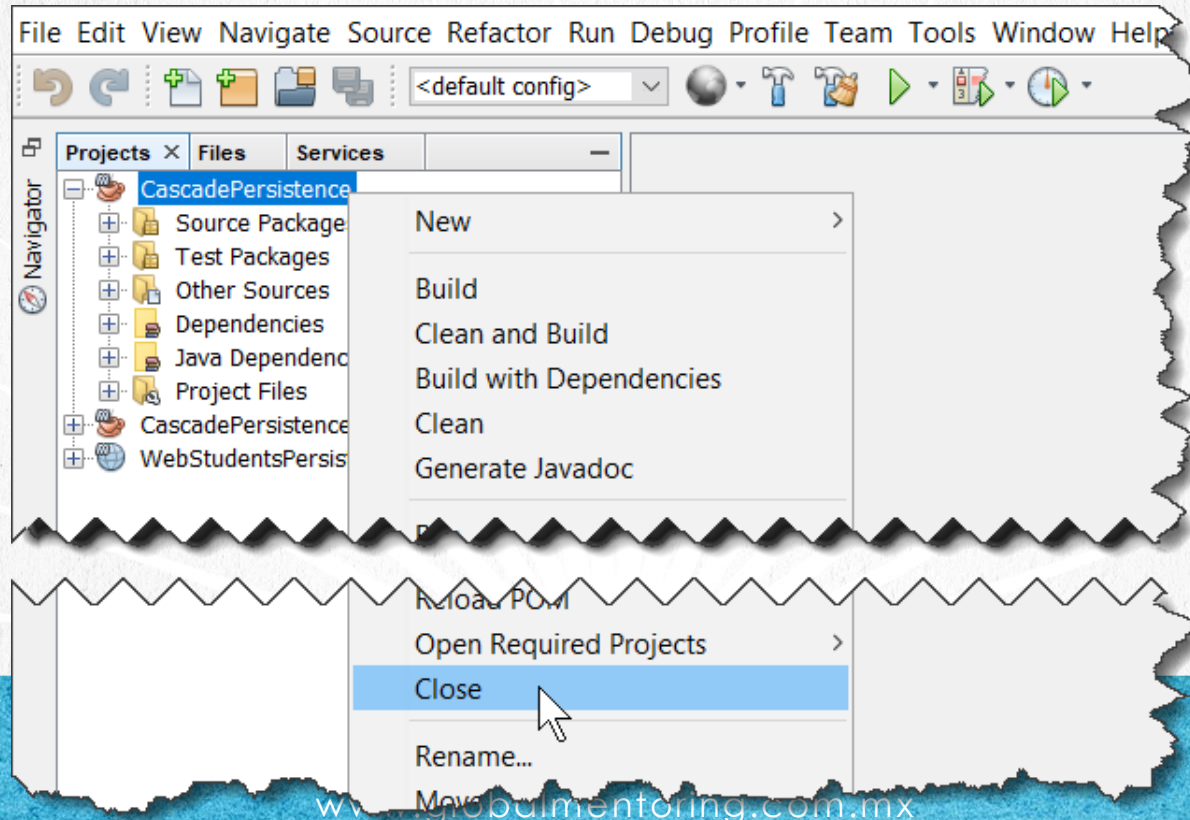


HIBERNATE & JPA COURSE

www.globalmentoring.com.mx

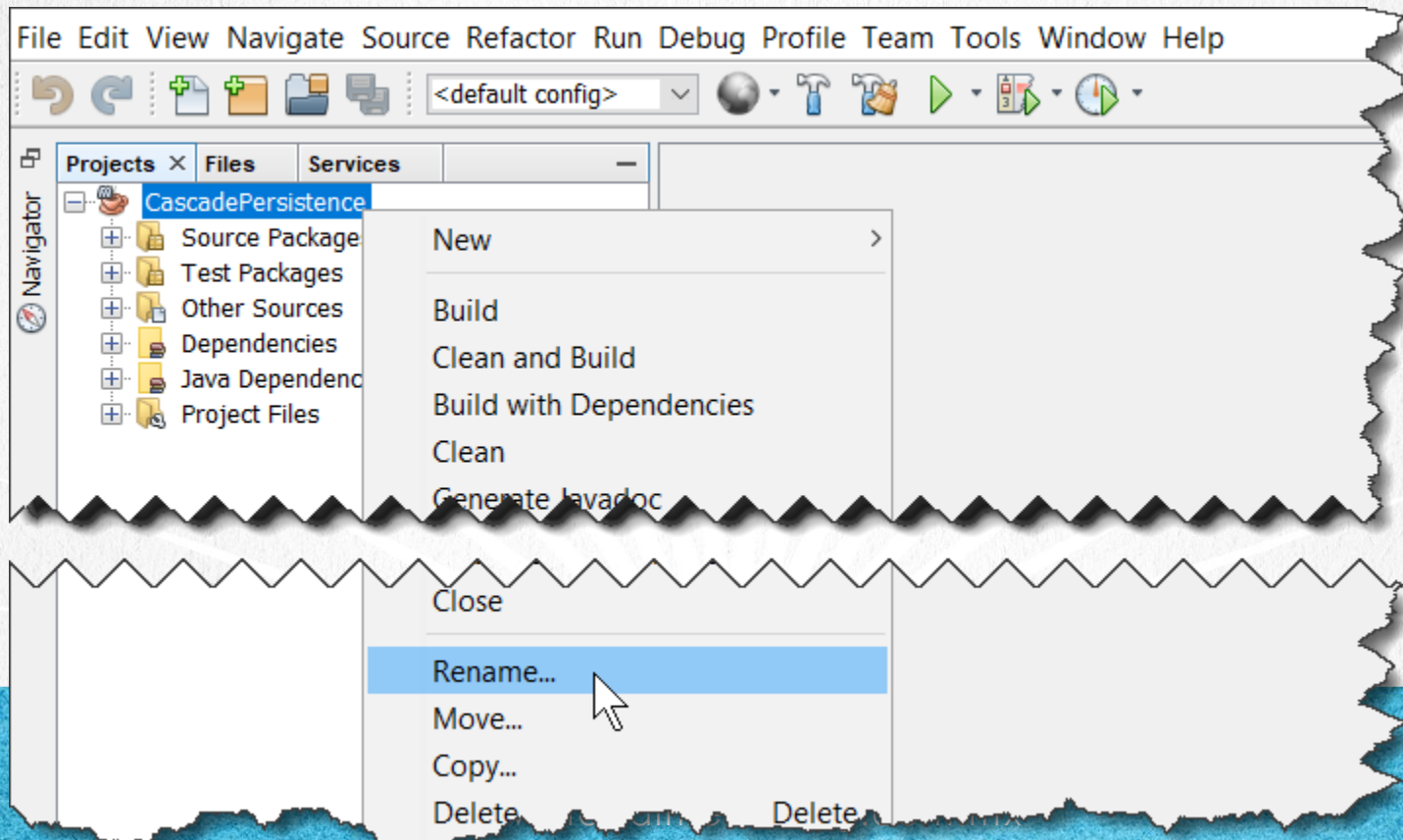
2. CLOSED THE PROJECT NOT USED ANYMORE

We close the project that we no longer use:



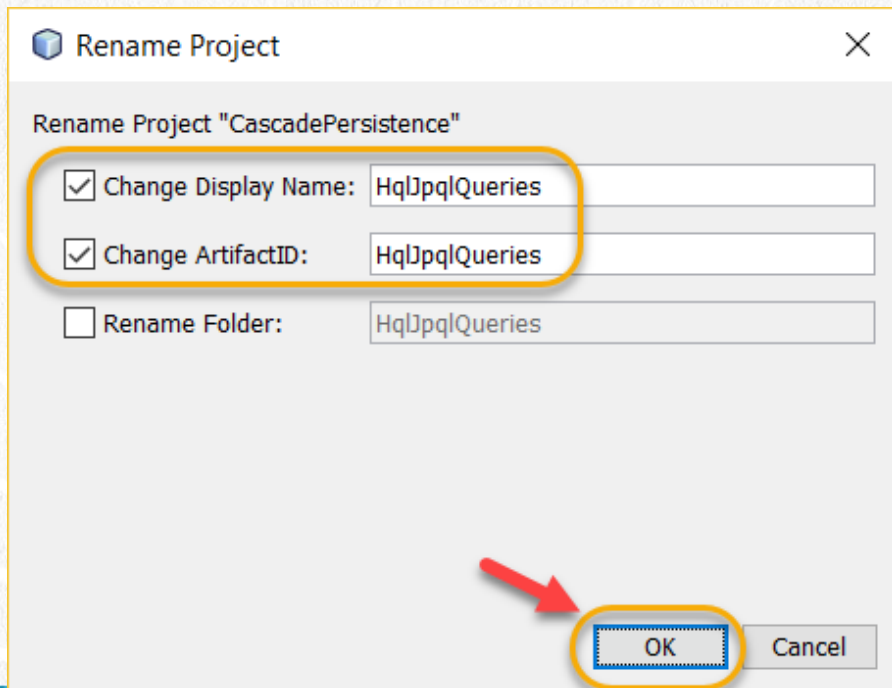
3. RENAME THE PROJECT

Rename the project:



3. RENAME THE PROJECT

Rename the Project:



Rename Project

Rename Project "CascadePersistence"

☒ Change Display Name: HqJpqlQueries

☒ Change ArtifactID: HqJpqlQueries

☐ Rename Folder: HqJpqlQueries

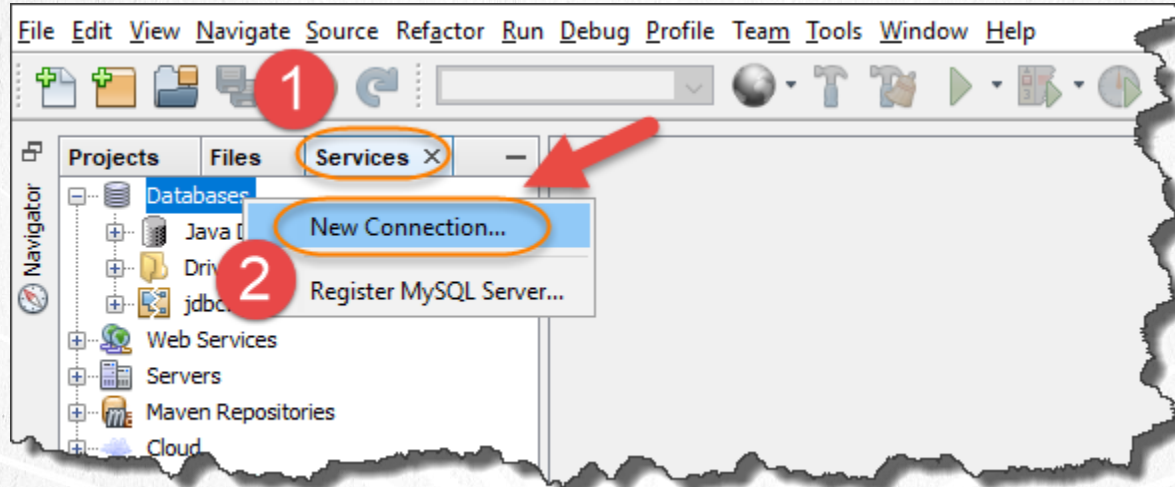
OK Cancel

HIBERNATE & JPA COURSE

www.globalmentoring.com.mx

4. CREATE A MYSQL CONNECTION

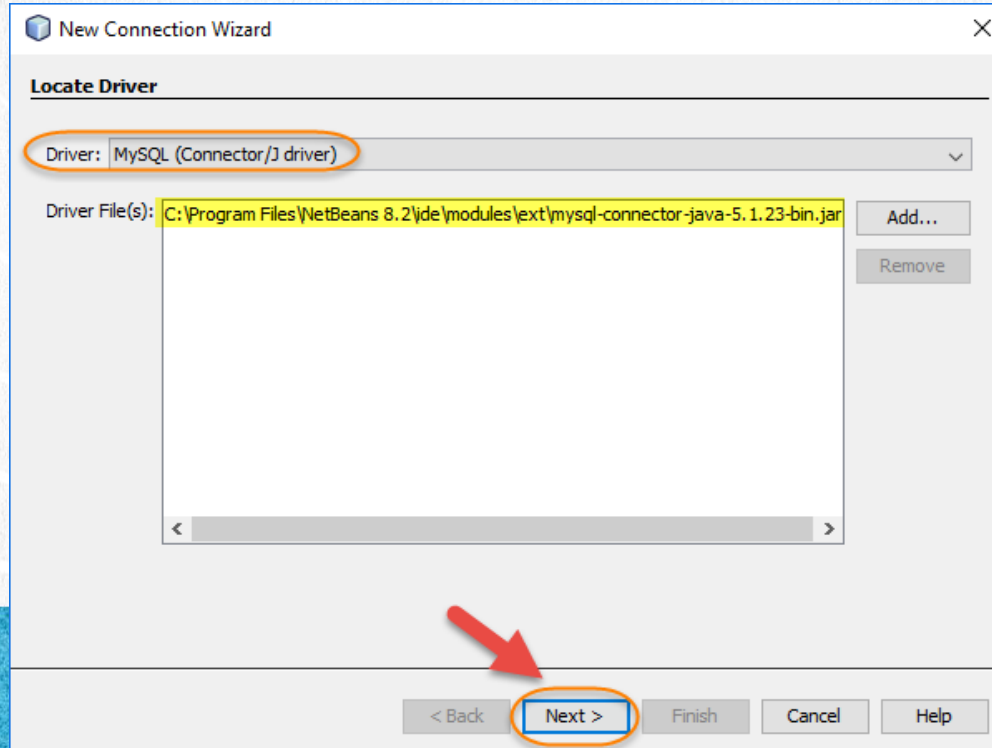
We created a connection to Mysql in the Services section:



4. CREATE A MYSQL CONNECTION

If you do not recognize the mysql.jar automatically, we add it from the following link:

<http://icursos.net/cursos/JavaJDBC/drivers/mysql-driver.jar>



4. CREATE A MYSQL CONNECTION

We add the values requested by the connection to the mysql sms_db database:

New Connection Wizard

Customize Connection

Driver Name: MySQL (Connector/J driver)

Host: localhost Port: 3306

Database: sms_db

User Name: root

Password: password: admin

☒ Remember password

Connection Properties Test Connection

JDBC URL: jdbc:mysql://localhost:3306/sms_db?userSSL=false

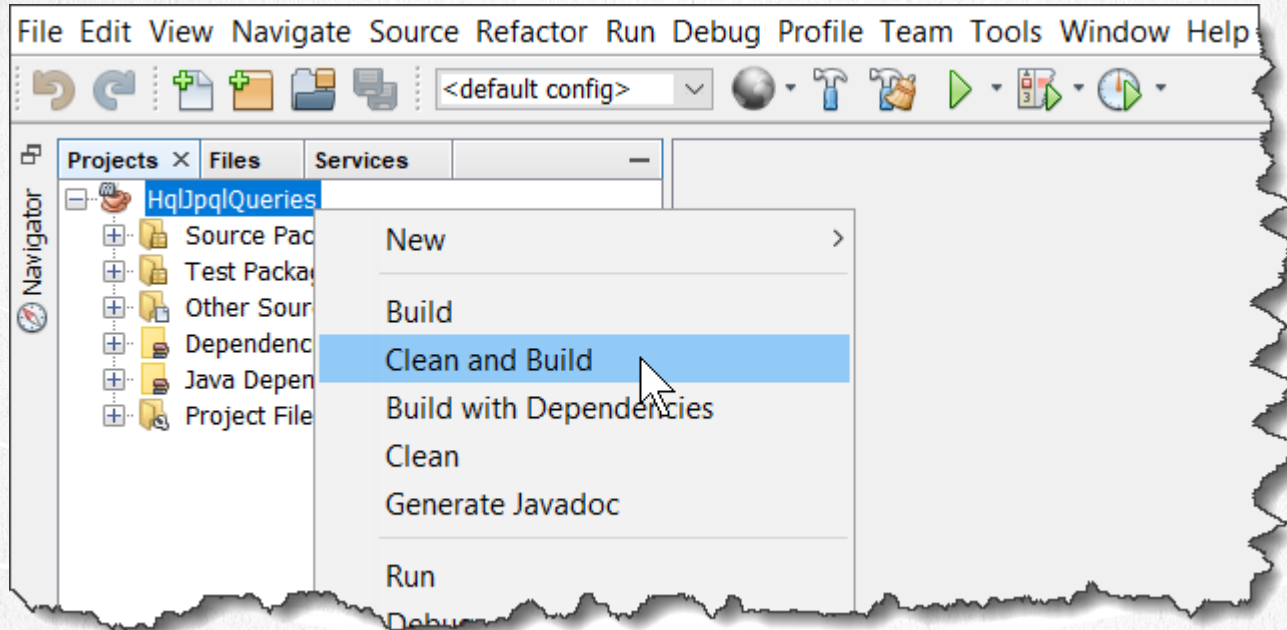
Connection Succeeded.

< Back Next > Finish Cancel Help

Very Important: All values must match the values contained in the **persistence.xml** file of the Java project

5. EXECUTE CLEAN & BUILD

We do a clean & build of the project, and we have to do it every time we open a JPQL console to ensure that we have the latest version in the Java and JPA code:

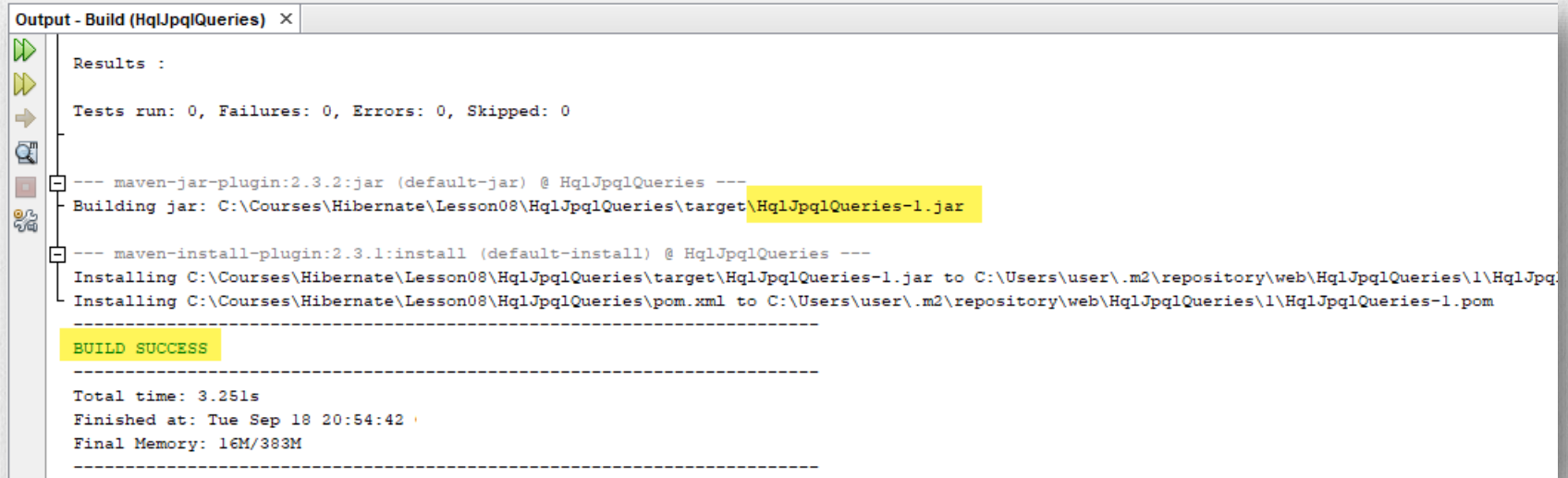


HIBERNATE & JPA COURSE

www.globalmentoring.com.mx

5. EXECUTE CLEAN & BUILD

We do a clean & build of the project, and we have to do it every time we open a JPQL console to ensure that we have the latest version in the Java and JPA code:



```
Output - Build (HqlJpqlQueries) X
Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

--- maven-jar-plugin:2.3.2:jar (default-jar) @ HqlJpqlQueries ---
Building jar: C:\Courses\Hibernate\Lesson08\HqlJpqlQueries\target\HqlJpqlQueries-1.jar

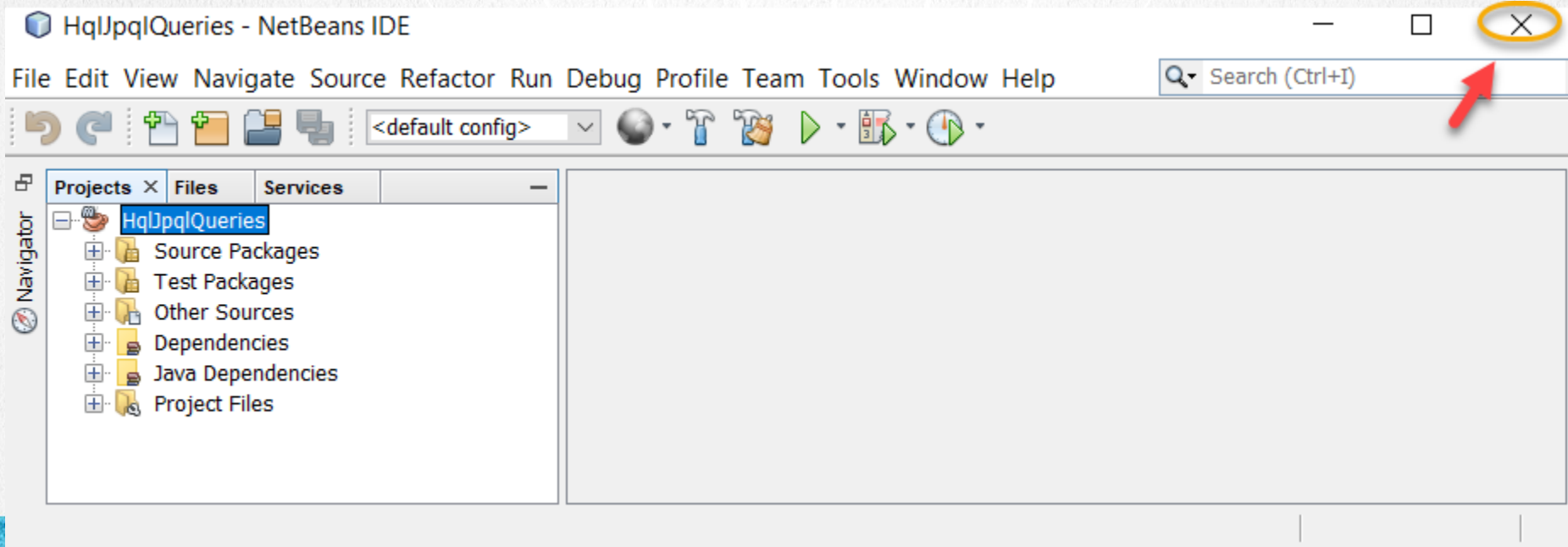
--- maven-install-plugin:2.3.1:install (default-install) @ HqlJpqlQueries ---
Installing C:\Courses\Hibernate\Lesson08\HqlJpqlQueries\target\HqlJpqlQueries-1.jar to C:\Users\user\.m2\repository\web\HqlJpqlQueries\1\HqlJpqlQueries-1.jar
Installing C:\Courses\Hibernate\Lesson08\HqlJpqlQueries\pom.xml to C:\Users\user\.m2\repository\web\HqlJpqlQueries\1\HqlJpqlQueries-1.pom

BUILD SUCCESS

Total time: 3.251s
Finished at: Tue Sep 18 20:54:42
Final Memory: 16M/383M
```


6. RESTART THE IDE

We close and reopen Netbeans so that it recognizes the persistence.xml file as a valid file to execute JPQL queries with the new changes:

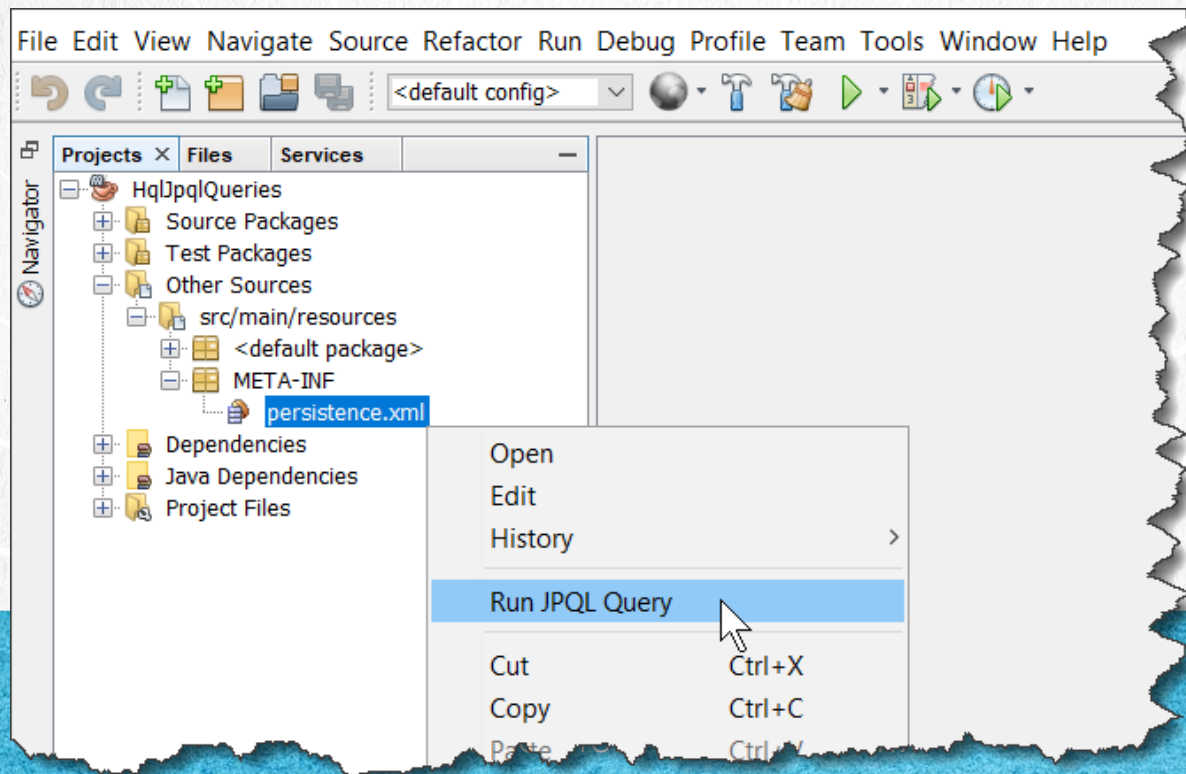


HIBERNATE & JPA COURSE

www.globalmentoring.com.mx

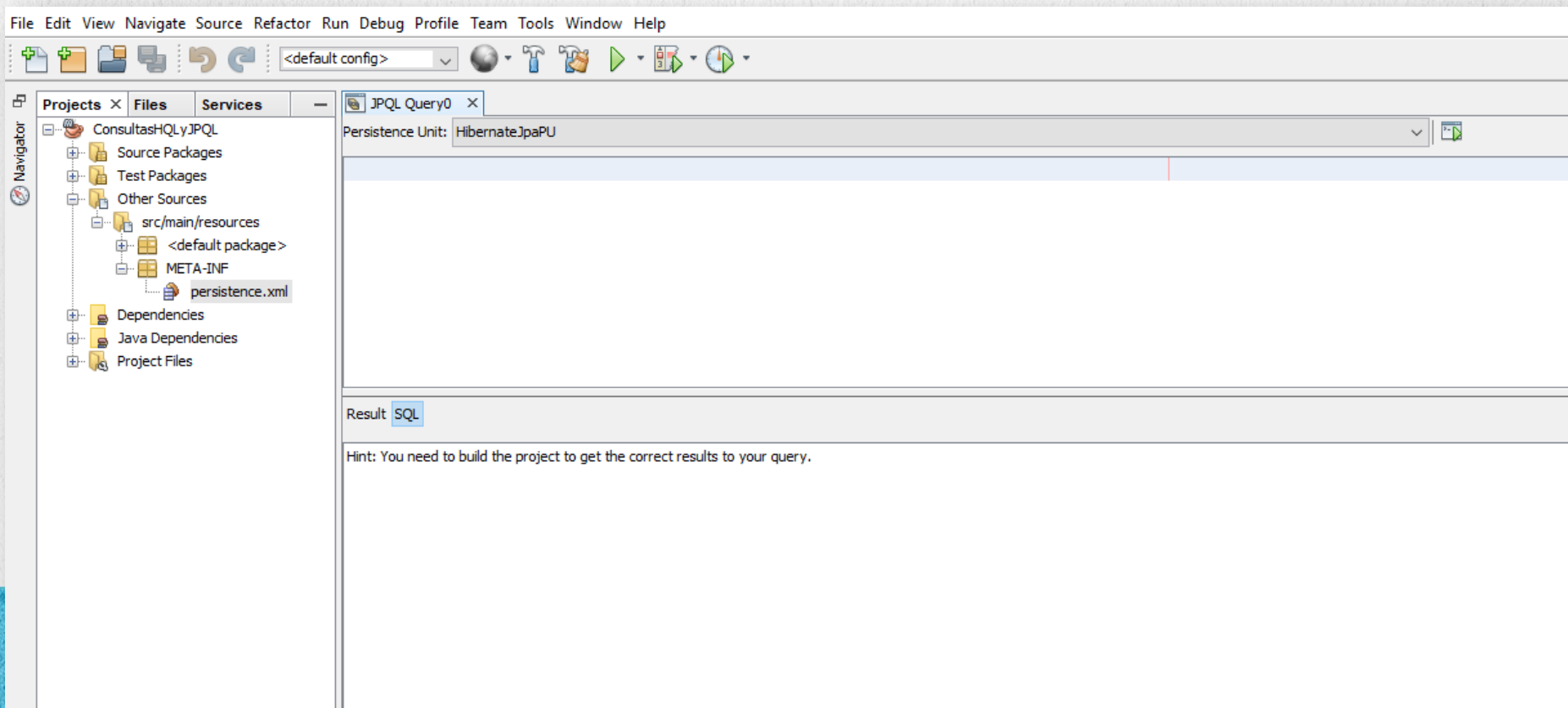
7. OPEN THE JPQL CONSOLE

Once the IDE is opened again, we open the console to execute JPQL queries. If for some reason it fails, check that the connection string of the persistence.xml file is the same as the one created in Netbeans, according to the steps indicated above:



7. OPEN THE JPQL CONSOLE

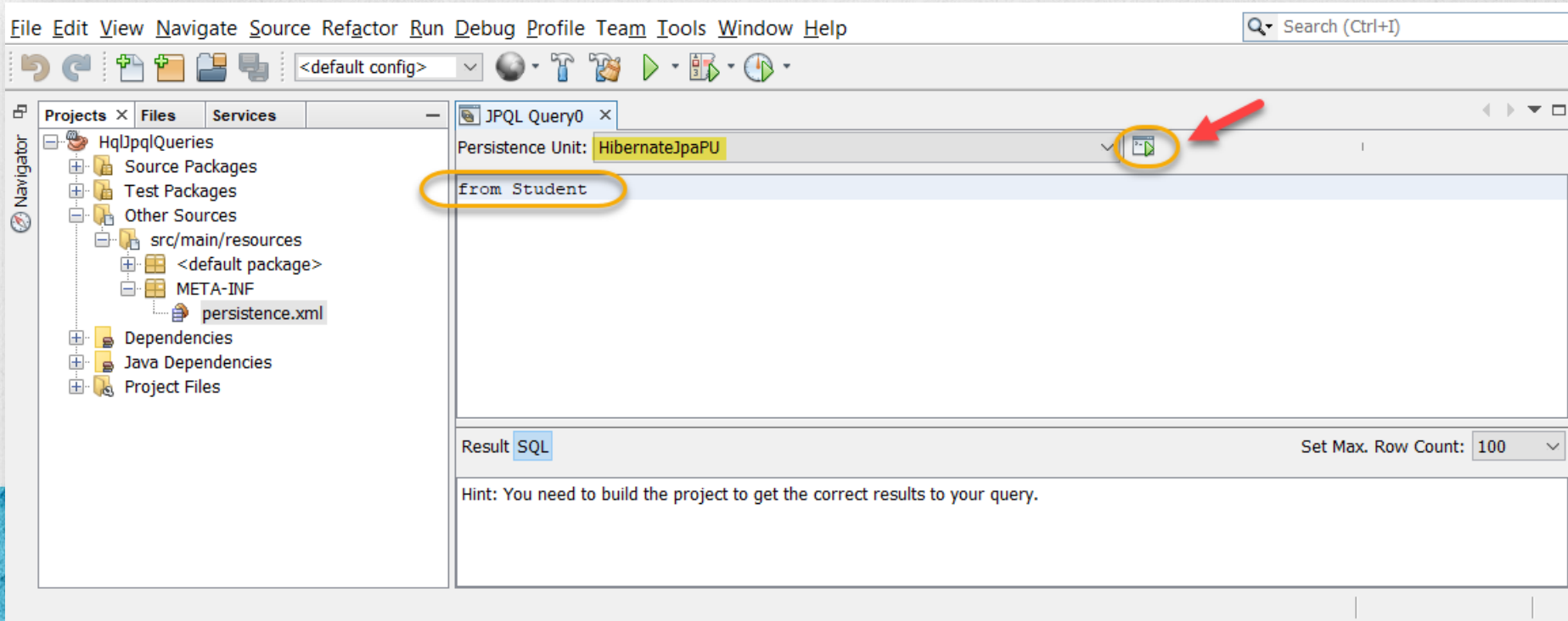
We open the console to execute JPQL queries:



7. WRITE THE JPQL QUERY

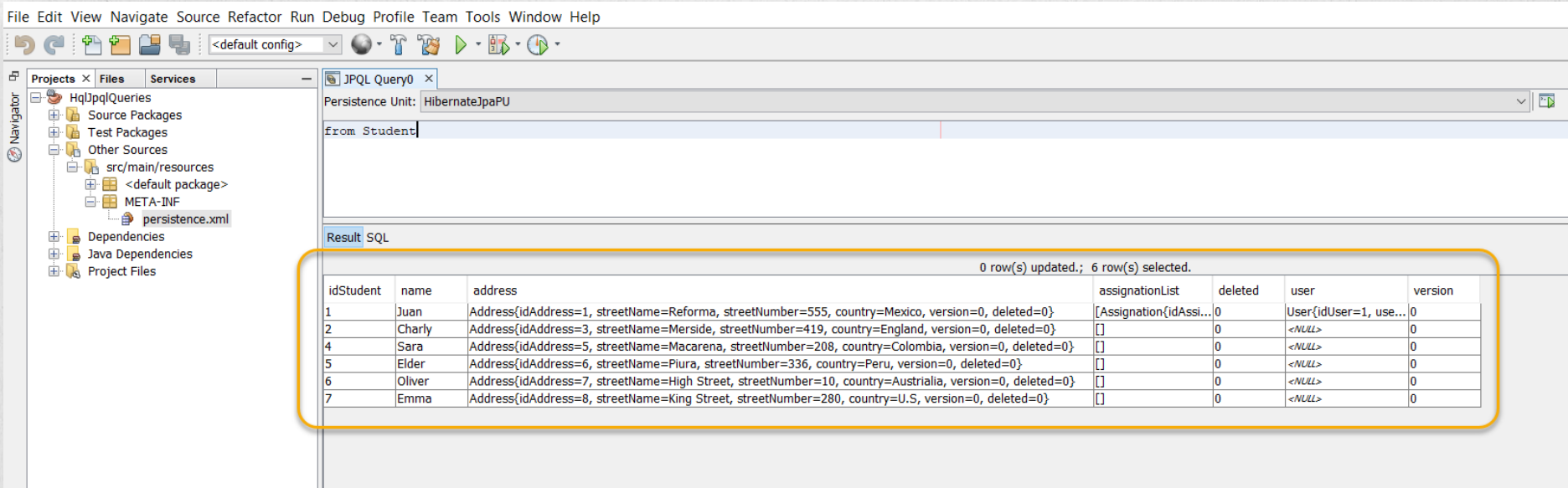
Write the query: `from Student`

And we run the query by pressing the Run JPQL Query button as shown:



8. RUN THE JPQL CONSOLE

We observe the result of executing the query. The data may vary according to the information in the database.



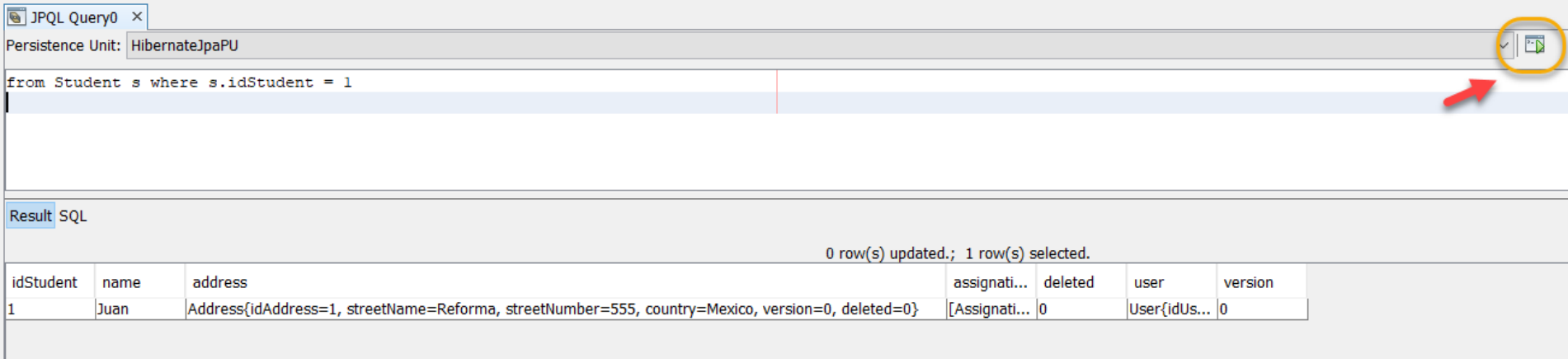
The screenshot shows the Eclipse IDE interface. The left sidebar contains the 'Projects' view with a tree structure including 'HqlJpqlQueries', 'Source Packages', 'Test Packages', 'Other Sources', 'src/main/resources', '<default package>', 'META-INF', 'persistence.xml', 'Dependencies', 'Java Dependencies', and 'Project Files'. The main editor area is titled 'JPQL Query0' and shows the query 'from Student' with 'Persistence Unit: HibernateJpaPU'. Below the query, the 'Result SQL' tab is active, displaying a table of results. The table has 7 rows and 7 columns: 'idStudent', 'name', 'address', 'assignmentList', 'deleted', 'user', and 'version'. The first row is highlighted in orange. The status bar at the top indicates '0 row(s) updated.; 6 row(s) selected.'

idStudent	name	address	assignmentList	deleted	user	version
1	Juan	Address{idAddress=1, streetName=Reforma, streetNumber=555, country=Mexico, version=0, deleted=0}	[Assignment{idAssi...	0	User{idUser=1, use...	0
2	Charly	Address{idAddress=3, streetName=Merside, streetNumber=419, country=England, version=0, deleted=0}	[]	0	<NULL>	0
4	Sara	Address{idAddress=5, streetName=Macarena, streetNumber=208, country=Colombia, version=0, deleted=0}	[]	0	<NULL>	0
5	Elder	Address{idAddress=6, streetName=Piura, streetNumber=336, country=Peru, version=0, deleted=0}	[]	0	<NULL>	0
6	Oliver	Address{idAddress=7, streetName=High Street, streetNumber=10, country=Australia, version=0, deleted=0}	[]	0	<NULL>	0
7	Emma	Address{idAddress=8, streetName=King Street, streetNumber=280, country=U.S, version=0, deleted=0}	[]	0	<NULL>	0

9. EXECUTE THE JPQL QUERY

Based on the previous steps, we execute the following queries. You may have to modify the values as they exist in your database, so the values may vary:

```
from Student s where s.idStudent = 1
```



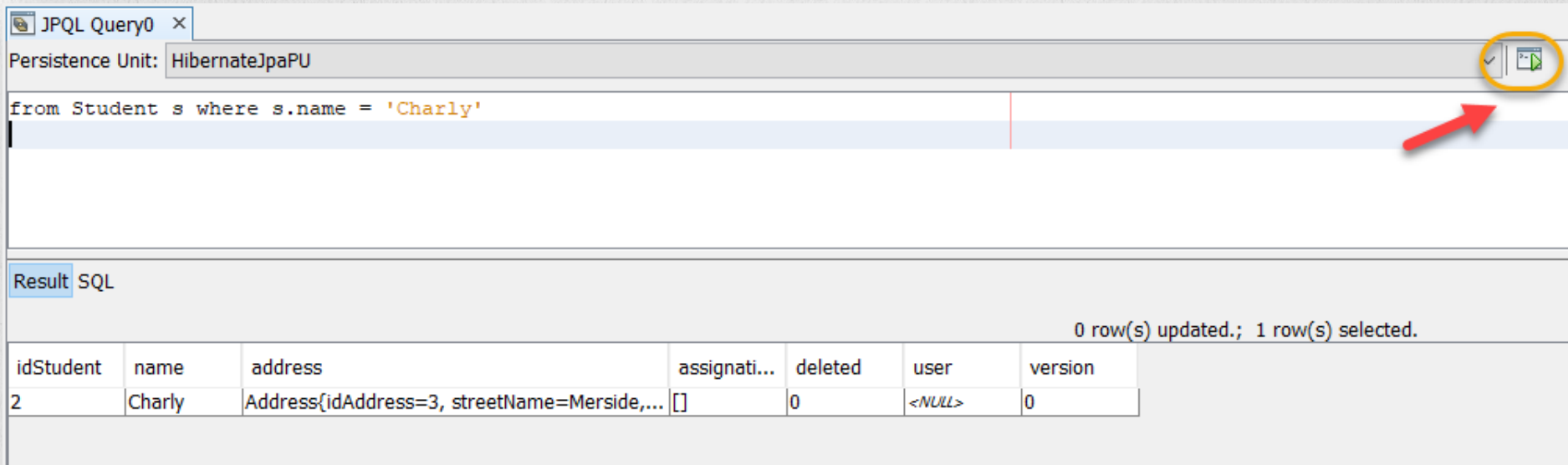
The screenshot shows a web-based JPQL Query tool interface. At the top, there's a tab labeled "JPQL Query0". Below it, the "Persistence Unit" is set to "HibernateJpaPU". The query text area contains the JPQL query: `from Student s where s.idStudent = 1`. To the right of the query area, there are two icons: a checkmark and a document with a magnifying glass. A red arrow points to the document icon. Below the query area, there's a "Result" tab and an "SQL" tab. The "Result" tab is active, showing the execution status: "0 row(s) updated.; 1 row(s) selected." Below this, a table displays the query results.

idStudent	name	address	assignati...	deleted	user	version
1	Juan	Address{idAddress=1, streetName=Reforma, streetNumber=555, country=Mexico, version=0, deleted=0}	[Assignati...	0	User{idUs...	0

10. EXECUTE THE JPQL QUERY

We execute the following query:

```
from Student s where s.name = 'Charly'
```



JPQL Query0 x

Persistence Unit: HibernateJpaPU

```
from Student s where s.name = 'Charly'
```

Result SQL

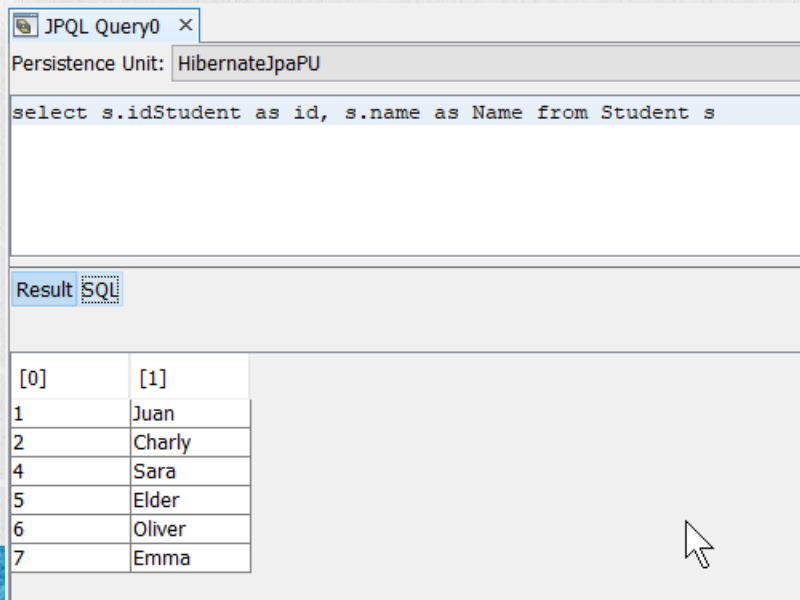
0 row(s) updated.; 1 row(s) selected.

idStudent	name	address	assignati...	deleted	user	version
2	Charly	Address{idAddress=3, streetName=Merside,...	[]	0	<NULL>	0

11. EXECUTE THE JPQL QUERY

We execute the following query:

```
select s.idStudent as id, s.name as Name from Student s
```



The screenshot shows a software interface for executing JPQL queries. At the top, there's a tab labeled 'JPQL Query0'. Below it, the 'Persistence Unit' is set to 'HibernateJpaPU'. The query text area contains the JPQL statement: 'select s.idStudent as id, s.name as Name from Student s'. Below the query area, there are two tabs: 'Result' and 'SQL', with 'Result' currently selected. The results are displayed in a table with two columns: '[0]' and '[1]'. The table contains seven rows of data, representing students with IDs 1 through 7 and names Juan, Charly, Sara, Elder, Oliver, and Emma. A mouse cursor is visible at the bottom right of the results table.

[0]	[1]
1	Juan
2	Charly
4	Sara
5	Elder
6	Oliver
7	Emma

An array of
Object type of
2-column is
created

12. EXECUTE THE JPQL QUERY

We execute the following query:

```
select s, s.idStudent from Student s
```

JPQL Query0 x

Persistence Unit: HibernateJpaPU

```
select s, s.idStudent from Student s
```

An array of Object type of 2-column is created

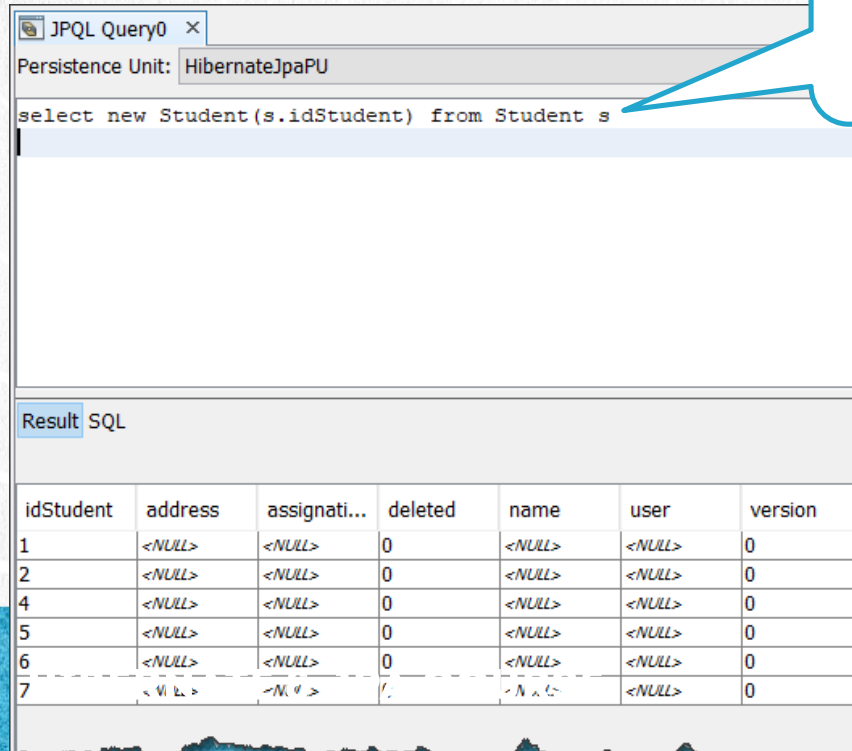
Result	SQL	1st Col	2nd Col				
[0].idStu...	[0].name	[0].addr...	[0].assig...	[0].deleted	[0].user	[0].version	[1]
1	Juan	Address{i...	[Assignati...	0	User{idUs...	0	1
2	Charly	Address(i...	[]	0	<NULL>	0	2
4	Sara	Address(i...	[]	0	<NULL>	0	4
5	Elder	Address(i...	[]	0	<NULL>	0	5
6	Oliver	Address(i...	[]	0	<NULL>	0	6
7	Emma	Address(i...	[]	0	<NULL>	0	7

13. EXECUTE THE JPQL QUERY

We execute the following query:

```
select new Student(s.idStudent) from Student s
```

We use the
idStudent
constructor



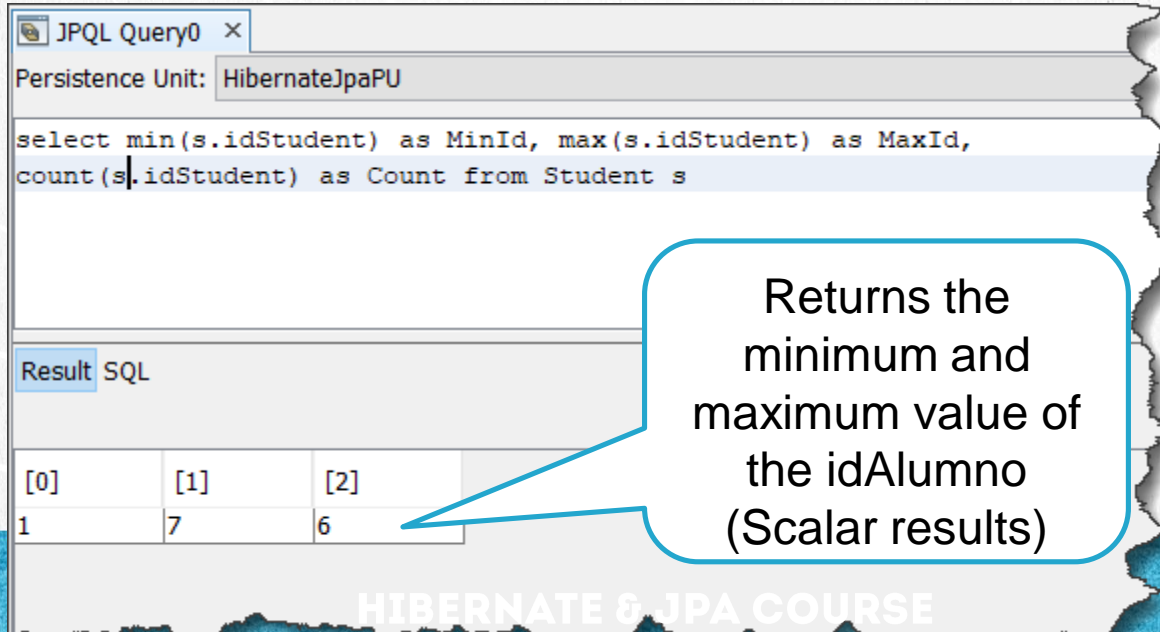
The screenshot shows a software interface for executing JPQL queries. At the top, a tab labeled 'JPQL Query0' is active. Below it, the 'Persistence Unit' is set to 'HibernateJpaPU'. The query text area contains the JPQL statement: `select new Student(s.idStudent) from Student s`. A blue callout bubble points to this query with the text 'We use the idStudent constructor'. Below the query area, there is a section titled 'Result SQL' which displays a table of results. The table has 7 columns: 'idStudent', 'address', 'assignati...', 'deleted', 'name', 'user', and 'version'. The first six rows of data show 'idStudent' values 1, 2, 4, 5, 6, and 7, with corresponding 'address' and 'assignati...' values, and a 'deleted' status of 0. The 'name', 'user', and 'version' columns contain '<NULL>' for all rows.

idStudent	address	assignati...	deleted	name	user	version
1	<NULL>	<NULL>	0	<NULL>	<NULL>	0
2	<NULL>	<NULL>	0	<NULL>	<NULL>	0
4	<NULL>	<NULL>	0	<NULL>	<NULL>	0
5	<NULL>	<NULL>	0	<NULL>	<NULL>	0
6	<NULL>	<NULL>	0	<NULL>	<NULL>	0
7	<NULL>	<NULL>	0	<NULL>	<NULL>	0

14. EXECUTE THE JPQL QUERY

We execute the following query:

```
select min(s.idStudent) as MinId, max(s.idStudent) as MaxId,  
count(s.idStudent) as Count from Student s
```



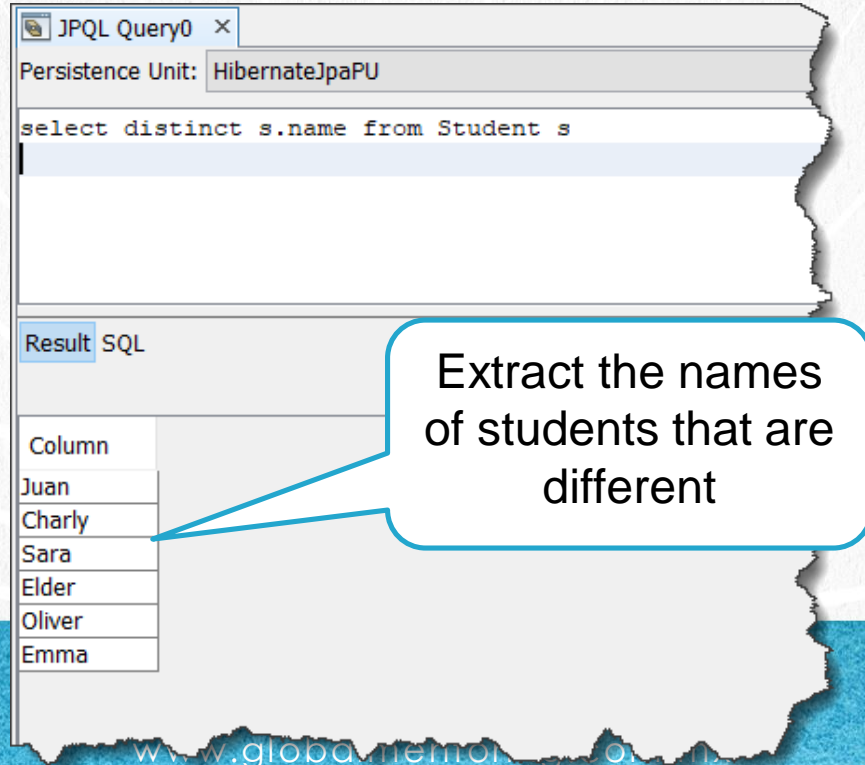
The screenshot shows a window titled "JPQL Query0" with a persistence unit of "HibernateJpaPU". The query being executed is: `select min(s.idStudent) as MinId, max(s.idStudent) as MaxId, count(s.idStudent) as Count from Student s`. Below the query, there is a "Result" tab and an "SQL" tab. The "Result" tab is active, showing a table with three columns: "[0]", "[1]", and "[2]". The values in the table are 1, 7, and 6 respectively. A callout box points to the value 6 in the "[2]" column, stating: "Returns the minimum and maximum value of the idAlumno (Scalar results)".

[0]	[1]	[2]
1	7	6

15. EXECUTE THE JPQL QUERY

We execute the following query:

```
select distinct s.name from Student s
```



The screenshot shows a window titled "JPQL Query0" with a persistence unit of "HibernateJpaPU". The query entered is "select distinct s.name from Student s". The results are displayed in a table with the following names: Juan, Charly, Sara, Elder, Oliver, and Emma. A callout box points to the "Charly" result with the text "Extract the names of students that are different".

Column
Juan
Charly
Sara
Elder
Oliver
Emma

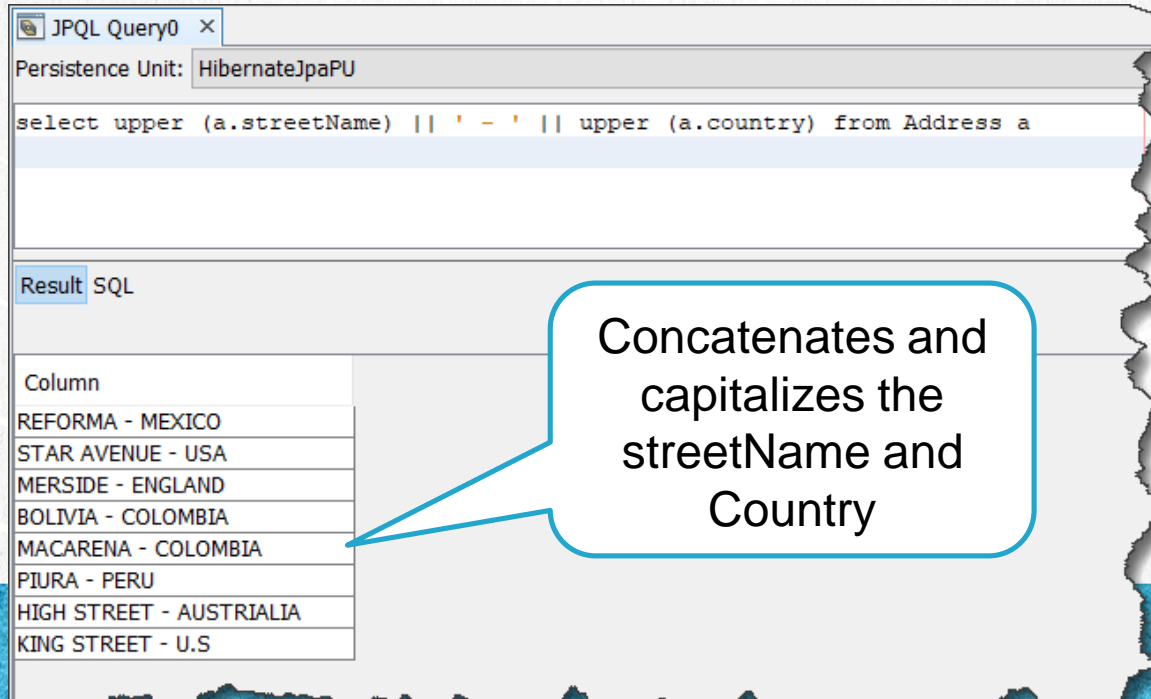
Extract the names of students that are different

www.globalmemory.com

16. EXECUTE THE JPQL QUERY

We execute the following query:

```
select upper (a.streetName) || ' - ' || upper (a.country) from Address a
```



The screenshot shows a window titled "JPQL Query0" with a tab for "HibernateJpaPU". The query text is: `select upper (a.streetName) || ' - ' || upper (a.country) from Address a`. Below the query, there is a "Result SQL" section. A table displays the results of the query, with a single column labeled "Column". The results are: REFORMA - MEXICO, STAR AVENUE - USA, MERSIDE - ENGLAND, BOLIVIA - COLOMBIA, MACARENA - COLOMBIA, PIURA - PERU, HIGH STREET - AUSTRIALIA, and KING STREET - U.S. A speech bubble points to the table with the text: "Concatenates and capitalizes the streetName and Country".

Column
REFORMA - MEXICO
STAR AVENUE - USA
MERSIDE - ENGLAND
BOLIVIA - COLOMBIA
MACARENA - COLOMBIA
PIURA - PERU
HIGH STREET - AUSTRIALIA
KING STREET - U.S

17. EXECUTE THE JPQL QUERY

The query with parameters can not be executed in the Netbeans console, but we will execute it with Java code.

from Student s where s.idStudent = :id

Result: Get the student object with id equal to the given parameter

JPQL Query0 x

Persistence Unit: HibernateJpaPU

from Student s where s.idStudent = 1

Assuming that the parameter provided is id = 1

Result SQL

idStudent	name	address	assignati...	deleted	user	version
1	Juan	Address(idAddress=1, str...	[Assignati...	0	User(idUs...	0

HIBERNATE & JPA COURSE

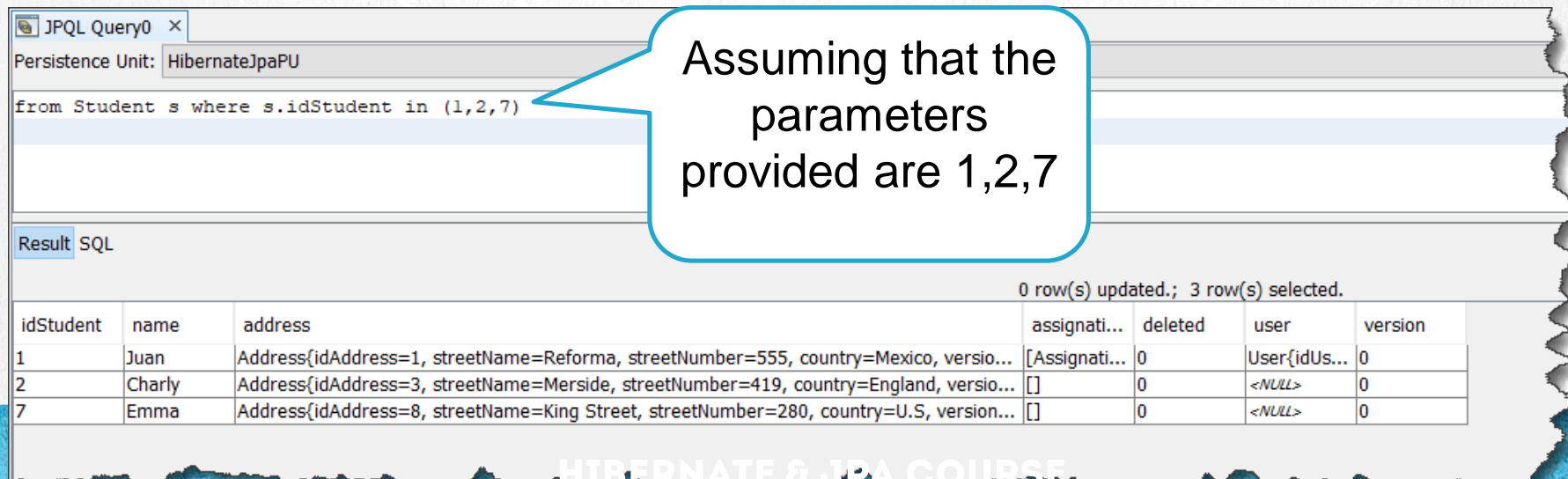
www.globalmentoring.com.mx

18. EXECUTE THE JPQL QUERY

The query with parameters can not be executed in the Netbeans console, since a list of data must be provided. We will do this from Java code.

from Student s where s.idStudent in (:list)

Result: Get students in a range of ids (can not run in the console)



The screenshot shows the NetBeans IDE with a JPQL Query window. The query is: `from Student s where s.idStudent in (1,2,7)`. A speech bubble points to the parameters `(1,2,7)` with the text: "Assuming that the parameters provided are 1,2,7". Below the query, the results are displayed in a table.

idStudent	name	address	assignati...	deleted	user	version
1	Juan	Address{idAddress=1, streetName=Reforma, streetNumber=555, country=Mexico, versio...	[Assignati...	0	User{idUs...	0
2	Charly	Address{idAddress=3, streetName=Merside, streetNumber=419, country=England, versio...	[]	0	<NULL>	0
7	Emma	Address{idAddress=8, streetName=King Street, streetNumber=280, country=U.S, version...	[]	0	<NULL>	0

HIBERNATE & JPA COURSE

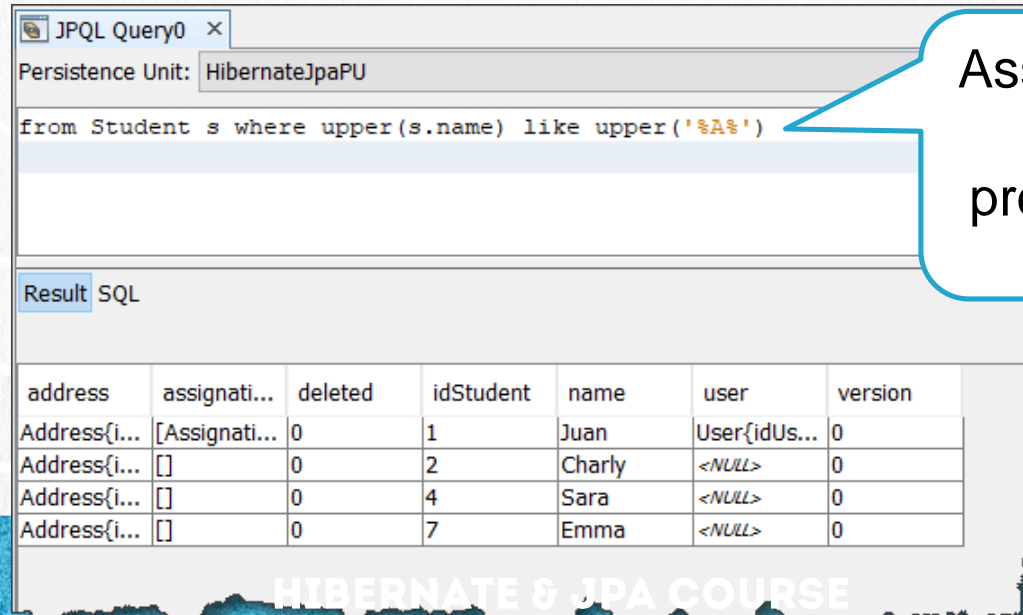
www.globalmentoring.com.mx

19. EXECUTE THE JPQL QUERY

The query with parameters can not be executed in the Netbeans console, but we will execute it with Java code.

```
from Student s where upper(s.name) like upper(:param1)
```

Result: Get students that contain a letter A, regardless of uppercase / lowercase.



The screenshot shows the NetBeans IDE with a window titled "JPQL Query0". The "Persistence Unit" is set to "HibernateJpaPU". The query text is "from Student s where upper(s.name) like upper('%A%')". Below the query, there is a "Result" tab showing a table of results. The table has columns: address, assignati..., deleted, idStudent, name, user, and version. The results are as follows:

address	assignati...	deleted	idStudent	name	user	version
Address[i...]	[Assignati...	0	1	Juan	User{idUs...	0
Address[i...]	[]	0	2	Charly	<NULL>	0
Address[i...]	[]	0	4	Sara	<NULL>	0
Address[i...]	[]	0	7	Emma	<NULL>	0

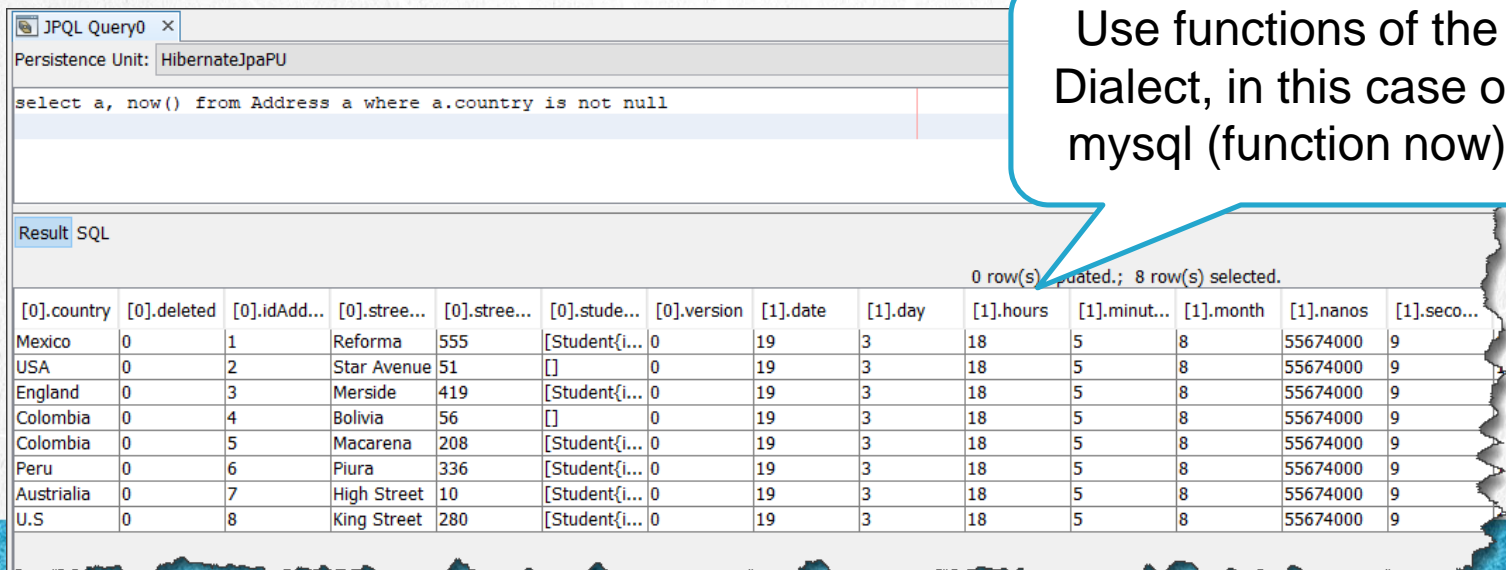
Assuming that the parameter provided is %A%

20. EXECUTE THE JPQL QUERY

We execute the following query:

```
select a, now() from Address a where a.country is not null
```

Result: Return the now date of mysql, similar to Date in Java.



JPQL Query0 x

Persistence Unit: HibernateJpaPU

```
select a, now() from Address a where a.country is not null
```

Result SQL

0 row(s) updated.; 8 row(s) selected.

[0].country	[0].deleted	[0].idAdd...	[0].stree...	[0].stree...	[0].stude...	[0].version	[1].date	[1].day	[1].hours	[1].minut...	[1].month	[1].nanos	[1].seco...
Mexico	0	1	Reforma	555	[Student{i...	0	19	3	18	5	8	55674000	9
USA	0	2	Star Avenue	51	[]	0	19	3	18	5	8	55674000	9
England	0	3	Merside	419	[Student{i...	0	19	3	18	5	8	55674000	9
Colombia	0	4	Bolivia	56	[]	0	19	3	18	5	8	55674000	9
Colombia	0	5	Macarena	208	[Student{i...	0	19	3	18	5	8	55674000	9
Peru	0	6	Piura	336	[Student{i...	0	19	3	18	5	8	55674000	9
Australia	0	7	High Street	10	[Student{i...	0	19	3	18	5	8	55674000	9
U.S	0	8	King Street	280	[Student{i...	0	19	3	18	5	8	55674000	9

HIBERNATE & JPA COURSE

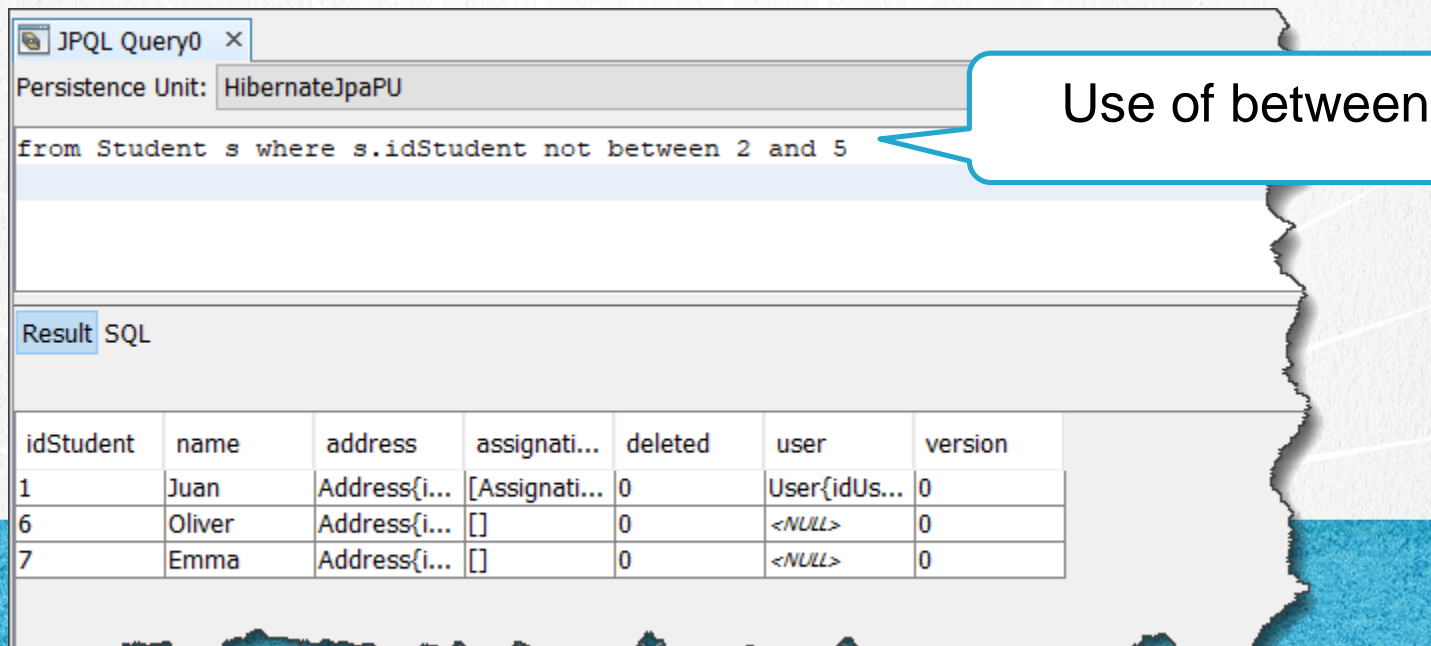
www.globalmentoring.com.mx

21. EXECUTE THE JPQL QUERY

We execute the following query:

```
from Student s where s.idStudent not between 2 and 5
```

Result: Return students whose id is NOT between 2 and 7



The screenshot shows a window titled "JPQL Query0" with a persistence unit of "HibernateJpaPU". The query entered is "from Student s where s.idStudent not between 2 and 5". A callout bubble points to the "between" keyword with the text "Use of between". Below the query, the "Result SQL" section is visible, showing a table with 7 columns: idStudent, name, address, assignati..., deleted, user, and version. The table contains 3 rows of data.

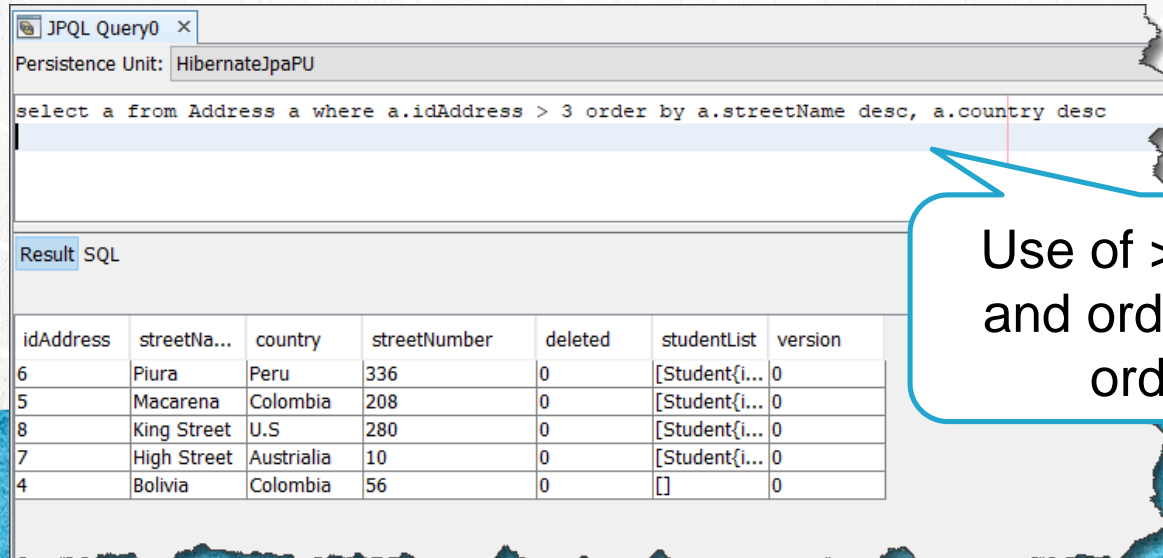
idStudent	name	address	assignati...	deleted	user	version
1	Juan	Address{i...	[Assignati...	0	User{idUs...	0
6	Oliver	Address{i...	[]	0	<NULL>	0
7	Emma	Address{i...	[]	0	<NULL>	0

22. EXECUTE THE JPQL QUERY

We execute the following query:

```
select a from Address a where a.idAddress > 3 order by a.streetName desc,  
a.country desc
```

Result: Return the addresses whose idAddress is greater than 3 and sorted by streetName and country in descending order



The screenshot shows a software interface for executing JPQL queries. At the top, a tab labeled 'JPQL Query0' is active. Below it, the 'Persistence Unit' is set to 'HibernateJpaPU'. The query text area contains: `select a from Address a where a.idAddress > 3 order by a.streetName desc, a.country desc`. Below the query area, there are two tabs: 'Result' (selected) and 'SQL'. The 'Result' tab displays a table with 7 columns: idAddress, streetNa..., country, streetNumber, deleted, studentList, and version. The table contains 5 rows of data, sorted by streetName and country in descending order. A blue callout bubble points to the query text, containing the text: 'Use of > operator and ordering with order by'.

idAddress	streetNa...	country	streetNumber	deleted	studentList	version
6	Piura	Peru	336	0	[Student(i... 0	
5	Macarena	Colombia	208	0	[Student(i... 0	
8	King Street	U.S	280	0	[Student(i... 0	
7	High Street	Australia	10	0	[Student(i... 0	
4	Bolivia	Colombia	56	0	[]	0

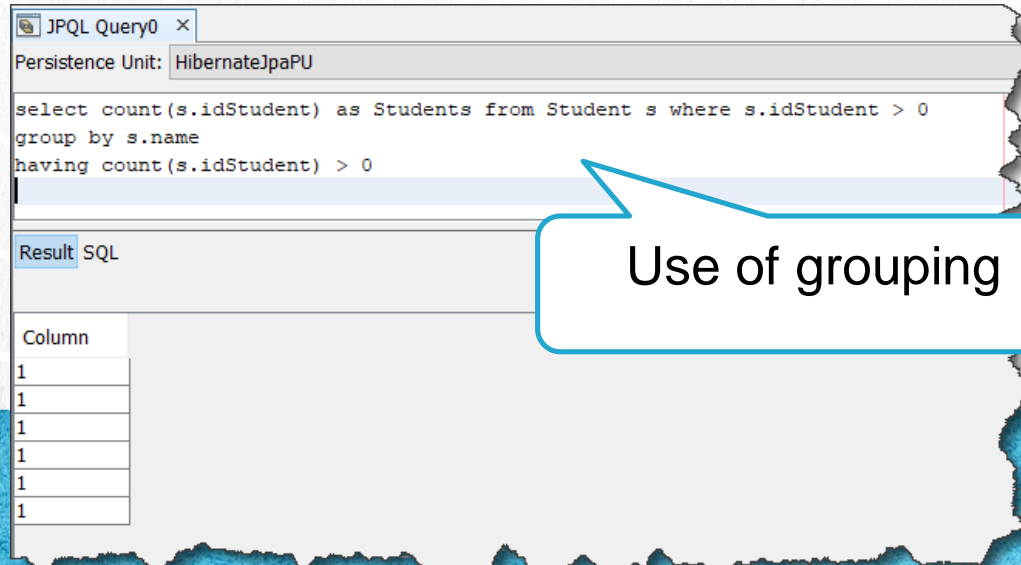
Use of > operator
and ordering with
order by

23. EXECUTE THE JPQL QUERY

We execute the following query:

```
select count(s.idStudent) as Students from Student s where s.idStudent > 0
group by s.name
having count(s.idStudent) > 0
```

Result: Return the student count, grouping by name, where the student ID is greater than zero and the group count contains at least 1 item.

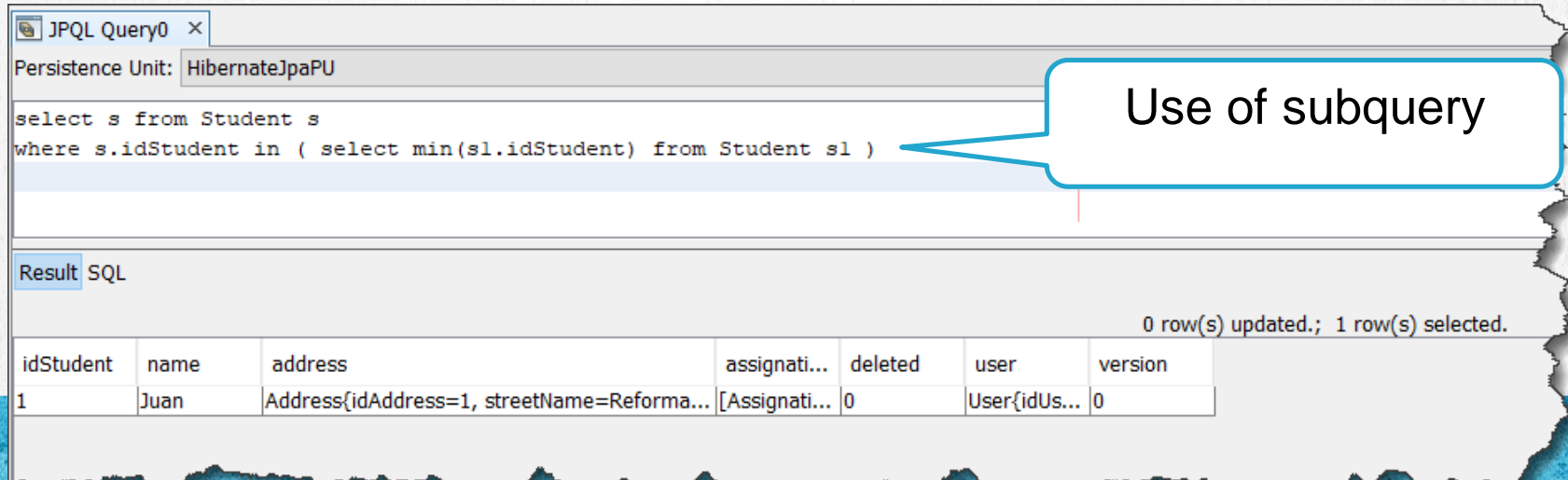


24. EXECUTE THE JPQL QUERY

We execute the following query:

```
select s from Student s
where s.idStudent in ( select min(s1.idStudent) from Student s1 )
```

Result: Return the student whose id is the smallest of the idAlumno. the support of this functionality depends on the database used.



The screenshot shows a software interface for executing JPQL queries. At the top, a tab labeled 'JPQL Query0' is open. Below it, the 'Persistence Unit' is set to 'HibernateJpaPU'. The query text area contains the JPQL statement: `select s from Student s where s.idStudent in (select min(s1.idStudent) from Student s1)`. A blue callout bubble with the text 'Use of subquery' points to the subquery part of the code. Below the query area, the 'Result' tab is selected, displaying 'SQL'. A status message indicates '0 row(s) updated.; 1 row(s) selected.' Below this, a table shows the result of the query.

idStudent	name	address	assignati...	deleted	user	version
1	Juan	Address{idAddress=1, streetName=Reforma...	[Assignati...	0	User{idUs...	0

25. EXECUTE THE JPQL QUERY

We execute the following query :

```
from Student s join s.address a
```

Result: Return the students without returning their relationships as is their address or contact.

JPQL Query0 x

Persistence Unit: HibernateJpaPU

from Student s
join s.address a

Using join with lazy loading

Result SQL

0 row(s) updated.; 6 row(s) selected.

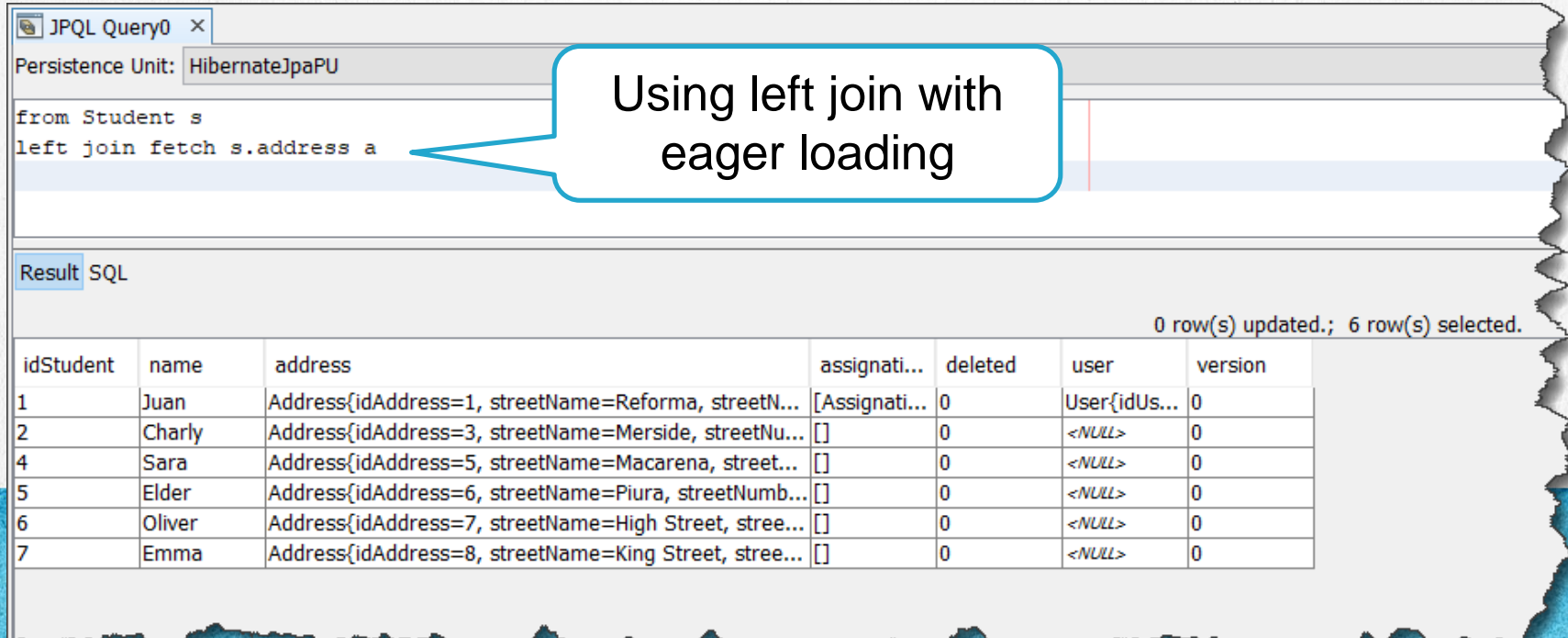
[0].idStu...	[0].name	[0].addr...	[0].assign...	[0].deleted	[0].user	[0].version	[1].idAdd...	[1].stree...	[1].country	[1].stree...	[1].studentList	[1].version	[1].deleted
1	Juan	Address{i...	[Assignati...	0	User{idUs...	0	1	Reforma	Mexico	555	[Student{idStud...	0	0
2	Charly	Address{i...	[]	0	<NULL>	0	3	Merside	England	419	[Student{idStud...	0	0
4	Sara	Address{i...	[]	0	<NULL>	0	5	Macarena	Colombia	208	[Student{idStud...	0	0
5	Elder	Address{i...	[]	0	<NULL>	0	6	Piura	Peru	336	[Student{idStud...	0	0
6	Oliver	Address{i...	[]	0	<NULL>	0	7	High Street	Austrialia	10	[Student{idStud...	0	0
7	Emma	Address{i...	[]	0	<NULL>	0	8	King Street	U.S	280	[Student{idStud...	0	0

26. EXECUTE THE JPQL QUERY

We execute the following query :

```
from Student s left join fetch s.address a
```

Result: Return the students also loading the relationships as address.



The screenshot shows a JPQL query execution window. The query is `from Student s left join fetch s.address a`. A callout bubble points to the `left join fetch` part of the query, stating "Using left join with eager loading". The results section shows the SQL query and a table of 6 rows. The table columns are `idStudent`, `name`, `address`, `assignati...`, `deleted`, `user`, and `version`. The first row shows a student named Juan with an address. The other five rows show students with null addresses.

JPQL Query0 x

Persistence Unit: HibernateJpaPU

```
from Student s
left join fetch s.address a
```

Using left join with eager loading

Result SQL

0 row(s) updated.; 6 row(s) selected.

idStudent	name	address	assignati...	deleted	user	version
1	Juan	Address{idAddress=1, streetName=Reforma, streetN...	[Assignati...	0	User{idUs...	0
2	Charly	Address{idAddress=3, streetName=Merside, streetNu...	[]	0	<NULL>	0
4	Sara	Address{idAddress=5, streetName=Macarena, street...	[]	0	<NULL>	0
5	Elder	Address{idAddress=6, streetName=Piura, streetNumb...	[]	0	<NULL>	0
6	Oliver	Address{idAddress=7, streetName=High Street, stree...	[]	0	<NULL>	0
7	Emma	Address{idAddress=8, streetName=King Street, stree...	[]	0	<NULL>	0

EXERCISE CONCLUSION

With this exercise we have reviewed several of the queries that we can execute with HQL / JPQL.

From simple queries, with parameters, ordering, groupings, lazy loading and eager loading, among several other examples.

Then we will execute these queries but from Java code.

ONLINE COURSE

HIBERNATE & JPA

By: Eng. Ubaldo Acosta



HIBERNATE & JPA COURSE
www.globalmentoring.com.mx