ITEM 3

Query C/1:

Specification: The average, the minimum, the maximum, and the standard deviation of the number of fix-up tasks per user.

JPQL query:
select avg(1.0*(select count(f.customer) from FixUpTask f where f.customer=c.id)),
min(1*(select count(f.customer) from FixUpTask f where f.customer=c.id)),
max(1*(select count(f.customer) from FixUpTask f where f.customer=c.id)),
stddev(1.0*(select count(f.customer) from FixUpTask f where f.customer=c.id)) from Customer c;

Description:
It counts the Fix up task per customers thanks of the foreign key in the column customer in FixUpTask`s table and the customer id. We finally calculate the average, the maximum, the minimum and the standard desviation. This query is complex due to our navigability.

Result:
1 object selected
[2.0, 1, 3, 1.0]

Query C/2:

Specification: The average, the minimum, the maximum, and the standard deviation of the number of applications per fix-up task.

JPQL query:
select avg(1.0*(select count(a.fixUpTask) from Application a where a.fixUpTask=f.id)),
min(1*(select count(a.fixUpTask) from Application a where a.fixUpTask=f.id)),
max(1*(select count(a.fixUpTask) from Application a where a.fixUpTask=f.id)),
stddev(1.0*(select count(a.fixUpTask) from Application a where a.fixUpTask=f.id)) from FixUpTask f;

Description:
It counts the applications per Fix up task thanks of the foreign key in the column fixUpTask in application`s table and the Fix up task id. We finally calculate the average, the maximum, the minimum and the standard desviation. This query is complex due to our navigability.

Result:
1 object selected
[1.25, 0, 3, 1.29904]

Query C/3:

Specification: The average, the minimum, the maximum, and the standard deviation of the maximum price of the fix-up tasks.

JPQL query: select
avg(f.maxPrice),min(f.maxPrice),max(f.maxPrice),stddev(f.maxPrice) from FixUpTask f;

Description:
This query selects the attribute maxPrice from table FixUpTask and calculates other values such the average, the maximum, the minimum and the standard desviation.

Result:
1 object selected
[327.3425, 150.54, 788.14, 266.62633781895966]

Query C/4:

Specification: The average, the minimum, the maximum, and the standard deviation of the price offered in the applications.

JPQL query: select
avg(a.offeredPrice),min(a.offeredPrice),max(a.offeredPrice),stddev(a.offeredPrice) from Application a;

Description:
This query selects the attribute offeredPrice from table Application and calculates other values such the average, the maximum, the minimum and the standard desviation.

Result:
1 object selected
[1040.708, 15.42, 2008.12, 631.4456436590564]

Query C/5:

Specification: The ratio of pending applications.

JPQL query:
select ((count(a)*1.0)/ (select count(a1)*1.0 from Application a1)) from Application a where a.status='PENDING';

Description:
This query calculates the ratio of pending applications by dividing the number of the pending application between the total number of applications.

Result:
1 object selected
0.2

Query C/6:

Specification: The ratio of accepted applications.

JPQL query:
select ((count(a)*1.0)/ (select count(a1)*1.0 from Application a1)) from Application a where a.status='ACCEPTED';

Description:
This query calculates the ratio of pending applications by dividing the number of the accepted application between the total number of applications.

Result:
1 object selected
0.6

Query C/7:

Specification: The ratio of rejected applications.

JPQL query:
select ((count(a)*1.0)/ (select count(a1)*1.0 from Application a1)) from Application a where a.status='REJECTED';

Description:
This query calculates the ratio of pending applications by dividing the number of the rejected application between the total number of applications.

Result:
1 object selected
0.2

Query C/8:

Specification: The ratio of pending applications that cannot change its status because their time period's elapsed.

JPQL query: select ((count(a)*1.0)/ (select count(a1)*1.0 from Application a1)) from Application a where a.status='PENDING' and a.fixUpTask.startTime<CURRENT_TIMESTAMP();

Description:
This query calculates the ratio of pending applications by dividing the number of applications whose status is pending and their period of time has expired (this period ends when his respective FixUpTask starts) between the total number of applications.

Result:
1 object selected
0.2

Query C/9:

Specification: The listing of customers who have published at least 10% more fix-up tasks than the average, ordered by number of applications.

JPQL query:
select c from Customer c where (select (count(f1)*1.0) / (select count(c2) from Customer c2) from FixUpTask f1)* 1.1 <= (select (count(f3)*1.0) from FixUpTask f3 where f3.customer.id= c.id) ;

Description: In this query we couldn´t complete it because the sorting criteria was so complex because of our navigability.

Result:
1 object selected
domain.Customer{id=3891, version=0}
domain.DomainEntity::id: int = 3891
domain.DomainEntity::version: int = 0
domain.Actor::name: java.lang.String = "Lola"
domain.Actor::middleName: java.lang.String = "Hugarte"
domain.Actor::surname: java.lang.String = "Hernandez"
domain.Actor::photo: java.lang.String = "http://Lola.com"
domain.Actor::email: java.lang.String = "lola@gmail.com"
domain.Actor::phoneNumber: java.lang.String = "+34 (123) 698893476"
domain.Actor::address: java.lang.String = "Avenida Alola"
domain.Actor::numSocialProfile: int = 8
domain.Actor::boxes: java.util.Collection = [domain.Box{id=3892, version=0}, domain.Box{id=3893, version=0}, domain.Box
{id=3894, version=0}, domain.Box{id=3895, version=0}]
domain.Actor::userAccount: security.UserAccount = security.UserAccount{id=3817, version=0}
domain.Endorser::score: double = 0.0

Query C/10:

Specification: The listing of handy workers who have got accepted at least 10% more applications than the average, ordered by number of applications.

JPQL query:
select hw from HandyWorker hw where (select (count(a1)*1.0) / (select count(a2) from Application a2) from Application a1 where a1.status='ACCEPTED')* 1.1 <= (select (count(a3)*1.0) / (select count(a4) from Application a4 where hw.id= a4.handyWorker.id)from Application a3 where a3.status='ACCEPTED' and hw.id=a3.handyWorker.id) order by(1.0*(select count(a.handyWorker) from Application a where a.handyWorker=hw.id));

Description:
In this query we selects the Handy workers who have 110% more accepted applications than the mean. In order to do that, we have calculated the mean of accepted applications, then, we multiply the mean with 1.1 to obtain the 110%. Below we calculate the mean of accepted applications per Handy worker (then we multiply it with 1.0 to transform it to double). Then we compare the result of the obtained before, to get if this Handy worker fulfill the condition. Finally we order the results by the number of applications per Handy worker.

Result:
domain.HandyWorker{id=3929, version=0}
domain.DomainEntity::id: int = 3929
domain.DomainEntity::version: int = 0
domain.Actor::name: java.lang.String = "Antonio"
domain.Actor::middleName: java.lang.String = null
domain.Actor::surname: java.lang.String = "Pérez Alvarez"
domain.Actor::photo: java.lang.String = "http://www.us.es"
domain.Actor::email: java.lang.String = "Antonio2001@alum.us.es"
domain.Actor::phoneNumber: java.lang.String = "+34 (655) 55559343"
domain.Actor::address: java.lang.String = null
domain.Actor::numSocialProfile: int = 1
domain.Actor::boxes: java.util.Collection = [domain.Box{id=3930, version=0}, domain.Box{id=3931, version=0}, domain.Box{id=3932, version=0}, domain.Box{id=3933, version=0}]
domain.Actor::userAccount: security.UserAccount = security.UserAccount{id=3825, version=0}
domain.Endorser::score: double = 0.0
domain.HandyWorker::make: java.lang.String = "Gamusinos S.A"
domain.HandyWorker::curriculum: domain.Curriculum = domain.Curriculum{id=3912, version=0}

Query B/1:

Specification: The minimum, the maximum, the average, and the standard deviation of the number of complaints per fix-up task.

JPQL query: select avg(1.0*(select count(co.fixUpTask) from Complaint co where co.fixUpTask=f.id)), min(1*(select count(co.fixUpTask) from Complaint co where co.fixUpTask=f.id)), max(1*(select count(co.fixUpTask) from Complaint co where co.fixUpTask=f.id)), stddev(1.0*(select count(co.fixUpTask) from Complaint co where co.fixUpTask=f.id)) from FixUpTask f;

Description:
It counts the complaints per Fix up task thanks of the foreign key in the column fixUpTask in complain`s table and the Fix up task id. We finally calculate the average, the maximum, the minimum and the standard desviation. This query is complex due to our navigability.

Result:
1 object selected
[0.75, 0, 2, 0.82916]

Query B/2:

Specification: The minimum, the maximum, the average, and the standard deviation of the number of notes per referee report.

JPQL query: select avg(1.0*(select count(n.report) from Note n where n.report=r.id)), min(1*(select count(n.report) from Note n where n.report=r.id)), max(1*(select count(n.report) from Note n where n.report=r.id)), stddev(1.0*(select count(n.report) from Note n where n.report=r.id)) from Report r;

Description:
It counts the notes per Report thanks of the foreign key in the column report in note`s table and the Report id. We finally calculate the average, the maximum, the minimum and the standard desviation. This query is complex due to our navigability.

Result: 1 object selected
[1.25, 0, 3, 1.29904]

Query B/3:

Specification: The ratio of fix-up tasks with a complaint.

JPQL query:
select count(*)/(select count(f)*1.0 from FixUpTask f) from Complaint c group by c.fixUpTask having count(*)=1;

Description:
We have counted the Fix up taks with only have one complaint. To do that we count how many times each fixUpTask´s id appears in the column fixUpTask in the table complaint. Then we divide this number between the total of fix up taks.

Result:
1 object selected
0.25

Query B/4:

Specification: The top-three customers in terms of complaints.

JPQL query:
select c from Customer c order by (1.0*(select count(co.customer) from Complaint co where co.customer=c.id)) desc;

Description:
In this query we obtain the customer order by the numbers of complaints. To obtain the number of caomplaints per customer we have counted how many times appears the id of each customer in the customer´s column in the complaint table.

Result: 2 objects selected
domain.Customer{id=3896, version=0}
domain.DomainEntity::id: int = 3896
domain.DomainEntity::version: int = 0
domain.Actor::name: java.lang.String = "Ana"
domain.Actor::middleName: java.lang.String = "Hugarte"
domain.Actor::surname: java.lang.String = "Hernandez"
domain.Actor::photo: java.lang.String = "http://Loa.com"
domain.Actor::email: java.lang.String = "loa@gmail.com"
domain.Actor::phoneNumber: java.lang.String = "+34 (123) 690893476"
domain.Actor::address: java.lang.String = null
domain.Actor::numSocialProfile: int = 8
domain.Actor::boxes: java.util.Collection = [domain.Box{id=3897, version=0}, domain.Box{id=3898, version=0}, domain.Box{id=3899, version=0}, domain.Box{id=3900, version=0}]
domain.Actor::userAccount: security.UserAccount = security.UserAccount{id=3818, version=0}
domain.Endorser::score: double = 0.4
domain.Customer{id=3891, version=0}
domain.DomainEntity::id: int = 3891
domain.DomainEntity::version: int = 0
domain.Actor::name: java.lang.String = "Lola"
domain.Actor::middleName: java.lang.String = "Hugarte"
domain.Actor::surname: java.lang.String = "Hernandez"
domain.Actor::photo: java.lang.String = "http://Lola.com"
domain.Actor::email: java.lang.String = "lola@gmail.com"
domain.Actor::phoneNumber: java.lang.String = "+34 (123) 698893476"
domain.Actor::address: java.lang.String = "Avenida Alola"
domain.Actor::numSocialProfile: int = 8
domain.Actor::boxes: java.util.Collection = [domain.Box{id=3892, version=0}, domain.Box{id=3893, version=0}, domain.Box{id=3894, version=0}, domain.Box{id=3895, version=0}]
domain.Actor::userAccount: security.UserAccount = security.UserAccount{id=3817, version=0}
domain.Endorser::score: double = 0.0

Query B/5:

Specification: The top-three handy workers in terms of complaints.

JPQL query:

Description:

Result: We dind´t manage to do it, we found it very hard because of our navegability.