Question I: SQL queries (6 pts.) See the code below:

```
select h.document_name, h.author, h.genre
from hypertext h, link l, webpage w
where
h.document_name = l.document_name and
h.author = l.author and
l.url = w.url and
w.blackliste = true;

select h.genre,sum(w.visit_count) as total_visits
from hypertext h, link l, webpage w
where
h.document_name = l.document_name and
h.author = l.author and
l.url = w.url and
w.blackliste = true
group by h.genre;

select h.document_name, h.author, h.genre
from hypertext h, link l, webpage w
where
h.document_name = l.document_name and
h.author = l.author and
l.url = w.url and
l.url not in(
select l.url
from link l, webpage w
where
l.url = w.url and
w.blackliste = true)
```

Question II: Transactions (6 pts.) There is a lost update on C. See the table below for the strict 2PL scheduling:

T1	T2
S(A)	
R(A)	
S(C)	
R(C)	
X(C)	
W(C)	
	S(A)
	R(A)
	WaitLock(C)
Commit	
Unlock(C)	
	X(C)
	W(C)
	Commit
	Unlock(A)
	Unlock(C)

There is a loop in the wait graph between T2 and T3, thus a deadlock. In order to break it, either T2 or T3 must be aborted.

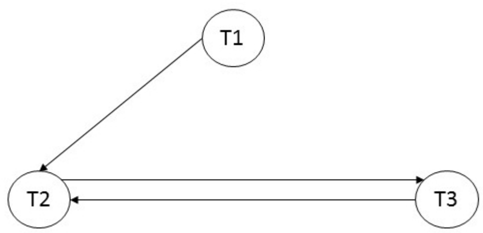


Figure 1: Wait graph for Question II

Question III: Graph databases (6 pts.) See code below:

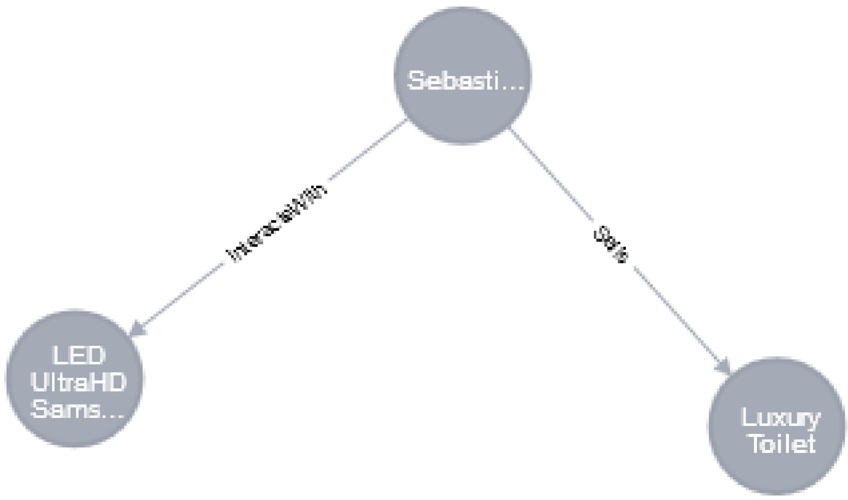
```
create(s:Sim{name:'Sebastian Castellanos',age:30,bladder:100,energy
    :100,happiness:20,position:[3,75],money:5000})
return s;

create(o:Object{name:'Luxury Toilet',type:'toilet',position
    :[3,75]})
return o;

match (s:Sim{name:'Sebastian Castellanos'}),(o:Object{name:'LED
    UltraHD Samsung'})
create s-[i:InteractsWith]->o
return s,i,o;

match (s:Sim{name:'Sebastian Castellanos'}),(o:Object{name:'Luxury
    Toilet'})
create s-[se:Sells{Price:250}]->o
return s,se,o;
```

The graph is the following:



The queries are the following:

```
match (s:Sim)-[i:InteractsWith]->(o:Object)
return s.name,o.type;

match (s:Sim)-[se:Sells]->(o:Object)
return sum(se.Price) as total;
```

Question IV - Database normalization (6 pts.) The table is in 1NF because all attributes are atomic. Decompose the first dependency using the decomposition rule in
`ssn` $\rightarrow$ `pnumber`
`ssn` $\rightarrow$ `hours`
The first dependency above does not violate 2NF because the left argument is part of the primary key but the right argument is a key attribute, but all the others do because the left argument is part of the primary key and the right argument a non-key attribute (see the definition of 2NF).

emp1	
<u>ssn</u>	hours

emp2	
<u>ssn</u>	ename

proj1		
<u>pnumber</u>	pname	plocation

employee_proj	
<u>ssn</u>	<u>pnumber</u>

In `employee_proj` the attribute `ssn` is a foreign key to `emp1` and `emp2`, while `pnumber` is a foreign key to `proj1`. Note that you can combine tables `emp1` and `emp2` because the dependencies
`ssn` $\rightarrow$ `hours`
`ssn` $\rightarrow$ `ename`
are the same of
`ssn` $\rightarrow$ `hours,ename`
because of the decomposition rule

Question V - Map-reduce (6pts.) The solution of the exercise is correct even without writing the commands to populate the database. This is the equivalent map-reduce code:

```
db.owners.insert({actor_id:"a1", name:"Morris",age:35});
db.owners.insert({actor_id:"a2", name:"Johnson", age: 46});
db.owners.insert({actor_id:"a3", name:"Louis", age: 46});
db.cars.insert({actor_id:"a1", plate:"AAB",model:"Mercedes SLK"});
db.cars.insert({actor_id:"a1", plate:"XXB",model:"Mercedes SLK"});
db.cars.insert({actor_id:"a2", plate:"NZY",model:"Mercedes SLK"});
db.cars.insert({actor_id:"a3", plate:"AAZ",model:"Porche GT"});

cars_map = function () {
    emit(this.owner_id, {plate: this.plate, model: this.model})
};

r = function(key, values) {
                var result = {
                model : "",
                plates : []
        };

        values.forEach(function(value) {
                        if(value.plate != null) {
                        result.plates.push(value);
                }
        });

        return result;

}

res = db.cars.mapReduce(cars_map, r, {query:{model : "Mercedes SLK"},out: {
    reduce : "joined"}});

-------------------------------------

cars_map = function() {
        emit(this.owner_id, {bsn : this.owner_id, plate : this.plate});
};


r = function(key, values) {
                var result = {
                bsn : "",
                plates : []
        };

        values.forEach(function(value) {
                if(value.bsn != null && value.plate != null) {
                        result.bsn = value.bsn;
                        result.plates.push(value.plate);
                }
        });

return result;

}

f = function(key, reduceValue) {
        if(reduceValue.plates != null && reduceValue.plates.length >= 2) {
                return reduceValue.bsn;
        }
        else {
                return {};
        }
}

res = db.cars.mapReduce(cars_map, r, {out: {reduce : "joined"},finalize : f})
    ;
```