TASK 2 Mistakes:

In general, tasks were done well… Just to avoid unnecessary information overloading I provide possible right answers (some other solutions might also exist):

Task 2-1:

Select all persons who belong to the class Father.

SPARQL query

```
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?name
WHERE {
       ?name a g:Father
      }
```

Select all pairs **x**-**y** where **x** is a father and **y** is his child. At the same time their weights should be different. Show the results in descending order of an age of the children.

SPARQL query

```
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?father ?child
WHERE {
   ?child g:hasParent ?father .
   ?father g:hasChild ?child .
   ?child g:weight ?c_weight .
   ?father g:weight ?f_weight .
   ?child g:hasAge ?Age .
   FILTER (!SAMETERM(?c_weight, ?f_weight))
}
ORDER BY DESC(?Age)

OR

PREFIX g: <http://www.owl-ontologies.com/generations.owl#>
SELECT ?father  ?child
WHERE {
  ?father a g:Father; g:hasChild ?Son; g:weight ?x.
  ?child g:hasAge ?Age; g:weight ?y.
  FILTER NOT EXISTS {
    ?child g:weight ?weight.
    ?father  g:weight ?weight }
}
ORDER BY DESC(?Age)

OR

PREFIX g: <http://www.owl-ontologies.com/generations.owl#>
SELECT ?father  ?child
WHERE {
  ?father a g:Father .
  ?father g:hasChild ?child .
  ?child g:weight ?weight .
  ?child g:hasAge ?Age .
  MINUS {?father g:weight ?weight}
}
ORDER BY DESC(?Age)
```

Select all Men, and optionally their children and their parents if this information is available.

SPARQL query

```
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?menName ?children ?parent
WHERE {
       ?menName a g:Man.
       OPTIONAL{?menName g:hasChild ?children}.
       OPTIONAL{?menName g:hasParent ?parent}
      }
OR
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?menName ?children ?parent
WHERE {
       ?menName a g:Male.
       OPTIONAL{?menName g:hasChild ?children}.
       OPTIONAL{?menName g:hasParent ?parent}
      }
OR
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?menName ?children ?parent
WHERE {
       ?menName g:hasSex g:MaleSex .
       OPTIONAL{?menName g:hasChild ?children} .
       OPTIONAL{?menName g:hasParent ?parent}
      }
```

Select all persons who belong to either of Brother or Sister class.

SPARQL query

```
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?people
WHERE {
       {?people a g:Sister}
       UNION
       {?people a g:Brother}
      }
```

Select all parents older than 30 who do not have any information whether he/she has parent.

SPARQL query

```
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?parent
WHERE {?parent a g:Parent; g:hasAge ?age .
FILTER (?age > 30).
FILTER NOT EXISTS{?parent g:hasParent ?x }
}

OR

PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?parent
WHERE {?parent a g:Parent; g:hasAge ?age .
FILTER (?age > 30).
MINUS { ?parent g:hasParent ?x. }
}
```

```
OR

PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

SELECT ?parent
WHERE {?parent a g:Parent; g:hasAge ?age .
FILTER (?age > 30).
OPTIONAL { ?parent g:hasParent ?x. }.
FILTER (!bound (?x)).
}
```

Return an amount of people that have child (or children).

SPARQL query

```
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT (COUNT(?parent) as ?amount)
WHERE {
         ?parent a g:Parent
         FILTER EXISTS {?parent g:hasChild ?child}
      }
OR

SELECT (COUNT(DISTINCT ?person) AS ?count)
WHERE {
         ?person g:hasChild ?child
      }
```

Try CONSTRUCT query: For all persons **x** who belong to either of Brother or Sister class, make a statement that **x** is of type **g:Sibling**.

SPARQL query

```
PREFIX g: <http://www.owl-ontologies.com/generations.owl#>

CONSTRUCT {?people a g:Sibling}
WHERE {
       {?people a g:Sister}
       UNION
       {?people a g:Brother}
      }
```

Task 2-2:

Show me full names (first name + surname) of all the people below 30 years of age who have a research job in ascending order of their ages.

SPARQL query

```
PREFIX e: <http://jyu.fi/employment#>
PREFIX f: <http://example.org#>
PREFIX j: <http://jyu.fi/jobs#>

SELECT ?fullName
WHERE {
       SERVICE <sparql endpoint of corresponding repository> {
          ?people e:worksAs ?job.
       }
       SERVICE <sparql endpoint of corresponding repository> {
          ?people f:age ?age.
          ?people f:firstName ?firstName.
          ?people f:surName ?surName.
       }
       SERVICE <sparql endpoint of corresponding repository> {
          ?job a j:ResearchJob.
       }
       FILTER (?age < 30)
       BIND(CONCAT(?firstName," ",?surName) AS ?fullName)
      }
ORDER BY(?age)
```

Show me all people who love a senior researcher.

SPARQL query

```
PREFIX e: <http://jyu.fi/employment#>
PREFIX f: <http://example.org#>
PREFIX j: <http://jyu.fi/jobs#>

SELECT ?people
WHERE {
       SERVICE <sparql endpoint of corresponding repository> {
          ?senior e:worksAs j:seniorResearcher.
       }
       SERVICE <sparql endpoint of corresponding repository> {
          ?people f:loves ?senior.
       }
      }
OR

PREFIX e: <http://jyu.fi/employment#>
PREFIX f: <http://example.org#>
PREFIX j: <http://jyu.fi/jobs#>

SELECT ?firstName ?surName
WHERE {
       SERVICE <sparql endpoint of corresponding repository> {
          ?senior e:worksAs j:seniorResearcher.
       }
       SERVICE <sparql endpoint of corresponding repository> {
          ?people f:loves ?senior.
          ?people f:firstName ?firstName.
          ?people f:surName ?surName.
       }
      }
```

Task 2-3:

You might notice some problems during uploading the content caused by wrong encoding of quotation character (“). Try to manually re-type it.
Since Fuseki v3 stars to support TriG format as well, you can upload data in TriG format directly from the file. But, there is a possibility to define named graphs in the structure of repository in other way. If you noticed, there is a possibility to provide Graph Id while uploading a file.
So… you may create 2 Turtle files. First one should contain triples from default graph of initial TriG document, second file should contain triples from the named graph. Specify corresponding graph ID while uploading the files (“default” for the first file, *<id of the named graph>* for the second). Unfortunately, some problems occur with IDs that contain “#”.Fuseki does not support “#” character in the graph URI (drops out the rest starting form #). Use “/” character instead of “#”.

Describe the way you did it and create/perform query that shows all triples with corresponding Graph ID. Perform the task using both: Sesame and Fuseki.
SPARQL query

`SELECT ?graphID ?subject ?predicate ?object`
`WHERE {`
`?subject ?predicate ?object.`
`OPTIONAL{GRAPH ?graphID { ?subject ?predicate ?object}}`
`}`

SPARQL query

`SELECT ?graphID ?subject ?predicate ?object`
`WHERE {`
`{?subject ?predicate ?object}`
`UNION`
`{GRAPH ?graphID { ?subject ?predicate ?object}}`
`}`

There were no requirement to perform all the sub-tasks in Task3 using Fuseki. Therefore, you are free to use Sesame, Fuseki, or both. Both of the repositories use different SPARQL Endpoints for normal queries and for Update. In Sesame you just chose different links for queries and Updates. In Fuseki you should specify corresponding SPARQL Endpoint: “http://localhost:3030/ds/query" or "http://localhost:3030/ds/update" (in case if the repository name is “ds”).
Check the notes regarding the graph handling approaches in different data stores as well as handling of empty graphs from the material of the Lecture 3.

Using SPARQL Update query create new Graph with ID: `http://www.example.com/ont#scBooks` in the repository. Create/perform query that returns “Graph exists” in case a graph with id (`http://www.example.com/ont#scBooks`) exists in repository.
SPARQL Update query
`CREATE GRAPH <http://www.example.com/ont#scBooks>`
SPARQL query

`SELECT ?result`
`WHERE{`
`GRAPH <http://www.example.com/ont#scBooks>{} .`
`BIND ("Graph exists" AS ?result)`
`}`
OR

```
SELECT ?result
WHERE{
      BIND (IF(EXISTS{GRAPH <http://www.example.com/ont#scBooks>{}},
      "Graph exists", "Graph doesn't exist") AS ?result)
     }
```

Using SPARQL Update query move all the scientific books (books that belongs to the ex:Science class) from graph (`http://www.example.com/ont#books`) to the newly created graph (`http://www.example.com/ont#scBooks`) . Shows content of repository (all triples with corresponding Graph ID.)

SPARQL Update query

```
PREFIX ex: <http://www.example.com/ont#>

INSERT {GRAPH ex:scBooks {?book ?x ?y}}
WHERE {GRAPH ex:books {?book a ex:Science .
                       ?book ?x ?y .  }
      };
WITH ex:books
DELETE {?book ?x ?y }
WHERE {?book a ex:Science .
       ?book ?x ?y .
      }
```

SPARQL query

```
SELECT ?graphID ?subject ?predicate ?object
WHERE {{?subject ?predicate ?object}
       UNION
       {OPTIONAL{GRAPH ?graphID { ?subject ?predicate ?object}}}
      }
```