POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Second EICTA Project

Author(s): **Bamhaoud Younes - 10767795**
Calcara Antonio - 11095781
Foini Lorenzo - 10828129
Martinelli Francesco - 10767793

Group Number: **7**

Academic Year: 2024-2025

Contents

1 | Introduction

The second Enterprise ICT Architectures project is focused on the definition of a graph database and the analysis of a text with the final objective of designing an operation flow for a process, using the BPMN notation.
For the first request, the used tools are Neo4j for the definition of the graph database and Cypher for subsequent querying of it.
For the second request, we used draw.io to define the analyzed business process, which allowed us to represent the process in a clear and detailed way, using the BPMN notation.

2 | Graph database introduction

In this chapter, we report the nodes selected for the creation of the graph database and the relationships between them.

2.1. Chosen nodes from the ER model

As required by the project delivery, five entities of the ER model and their respective relationships were selected, subsequently proceeding with the insertion of nodes and arcs within the graph. The previously defined ER model is shown in Figure 2.1.

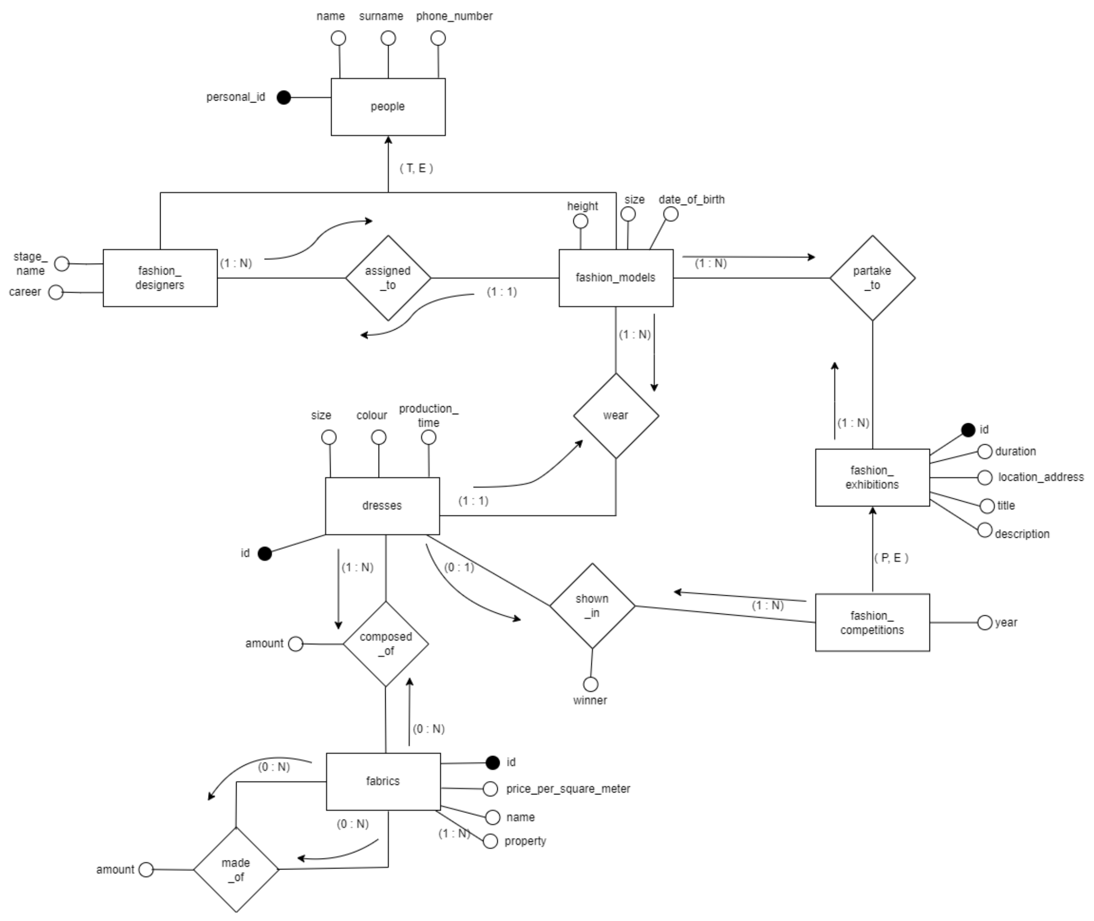


Figure 2.1: ER model from the first project.

Among the entities present, we have selected the following: *fashion_designers, fashion_models, dresses, fabrics, fashion_exhibitions.* These entities constitute the labels that we have used for the various nodes in the graph.
Then the following entity relationships have been translated: *assigned_to, wear, composed_of, partake_to.* These relationships are used as arcs, allowing the graph nodes to be related to each other.

NOTE: The relationship "*assigned_to*" has been transformed into "*manages*", changing the direction of the arrow, this will be analyzed in more detail within Subchapter 3.1.

2.2. Graph database's structure overview

In Figure 2.2 we show how the graph database is structured.
Details regarding the nodes and arcs of the graph are reported in Chapter 3.

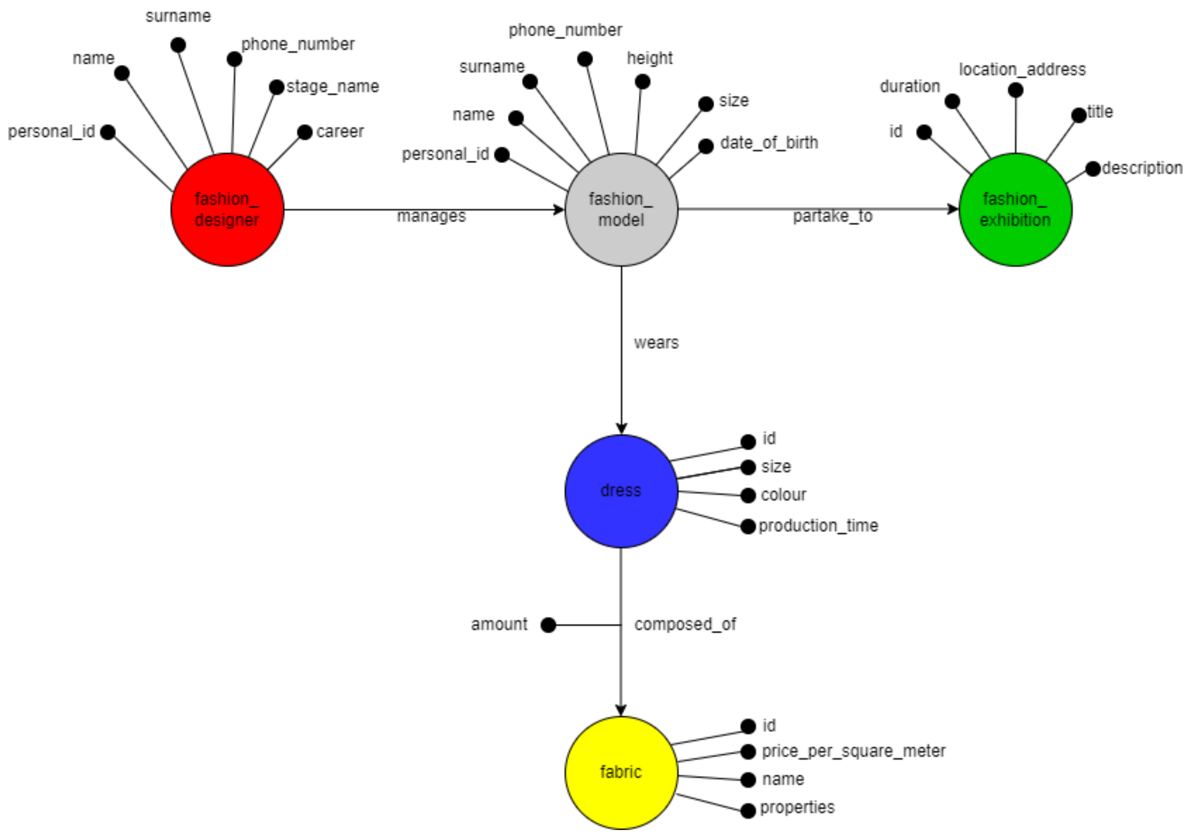


Figure 2.2: Graph nodes and arcs.

3 | Nodes and arcs

This chapter introduces the structure of the nodes and arcs that compose the graph, lastly showing how the data were imported starting from CSV files.

3.1. Nodes and arcs description

The information necessary to better understand all the elements present in the graph are now reported.

Nodes analysis:

- ***fashion_ designer* node:**
  All nodes labeled with *fashion_ designer* represent fashion designers who propose fashion models for fashion exhibitions and design dresses to be shown.
  All nodes that are labeled with *fashion_ designer* have the following attributes:
  - *personal_ id*: unique integer identifier of the designer.
  - *name*: string containing the designer's name.
  - *surname*: string containing the designer's surname.
  - *phone_ number*: string containing the designer's phone number.
  - *stage_ name*: string containing the designer's stage name.
  - *career*: text description of the designer's career.

- ***fashion_ model* node:**
  All nodes labeled with *fashion_ model* represent fashion models who participate in fashion exhibitions.
  All nodes that are labeled with *fashion_ model* have the following attributes:
  - *personal_ id*: unique integer identifier of the model.

  - *name*: string containing the model's name.
  - *surname*: string containing the model's surname.
  - *phone_ number*: string containing the model's phone number.
  - *height*: integer value in centimeters representing the model's height.
  - *size*: string containing the model's size.
  - *date_ of_ birth*: model's date of birth with the format "YYYY-MM-DD".

- ***dress* node:**
  All nodes labeled with *dress* represent the dresses that fashion models can wear during fashion exhibitions.
  All nodes that are labeled with *dress* have the following attributes:
  - *id*: unique integer identifier of the dress.
  - *size*: string containing the size of the dress.
  - *colour*: string containing the colour of the dress.
  - *production_ time*: decimal value of the time required to product the dress, with the format "minutes.seconds".

- ***fabric* node:**
  All nodes labeled with *fabric* contain information about the fabrics used to make the dresses.
  All nodes labeled with *fabric* have the following attributes:
  - *id*: unique integer identifier of the fabric.
  - *price_per_ square_ meter*: decimal value of the price per square meter of the fabric, with the format "euro.cents" and without the currency.
  - *name*: string containing the name of the fabric.
  - *properties*: list containing the properties of the fabric.

- ***fashion_ exhibition* node:**
  All nodes labeled with *fashion_ exhibition* contain information about fashion exhibitions

All nodes labeled with *fashion_ exhibition* have the following attributes:

- *id*: unique integer identifier of the fashion exhibitions.
- *duration*: decimal value of the duration of the exhibition, with the format "minutes.seconds".
- *location_ address*: string containing the address of where the exhibition is located.
- *title*: string containing the title of the exhibition.
- *description*: text containing a description of the exhibition.

Arcs analysis:

- ***manages* arc:**
  All arcs labeled with *manages* connect fashion designers to the respective fashion models they manage.
  These arcs have no attributes and they have a one-way direction:
  fashion_ designer $\rightarrow$ *fashion_ model.*

  NOTE: By creating the arcs in this way, each designer will link to their respective models. Differently, if we had kept the verse of direction as implemented in the ER model, then we would have that each model connects to the designer to which it is assigned, changing the verse of direction of the arcs.

- ***wears* arc:**
  All arcs labeled with *wears* connect dresses to the fashion models who wore them during fashion exhibitions.
  These arcs have no attributes and have a one-way direction:
  fashion_ model $\rightarrow$ *dress.*

- ***composed_ of* arc:**
  All arcs labeled with *composed_ of* connect dresses and the respective fabrics that the dresses are made of.
  Each arc has the attribute *amount*, which represents the percentage of the fabric present in the dress. The arcs have a one-way direction: *dress* $\rightarrow$ *fabric.*

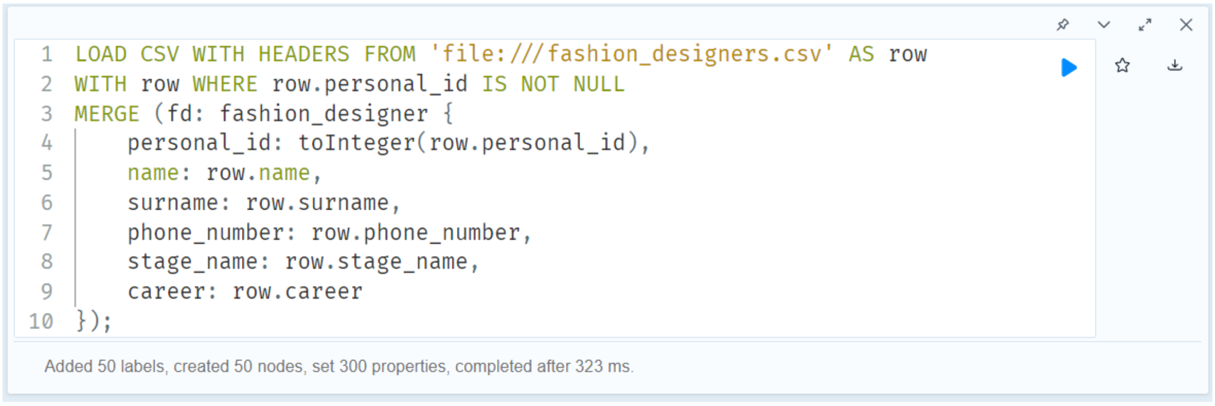
- ***partake_ to* arc:**
  All arcs labeled with *partake_ to* connect the fashion models to the fashion exhibitions they partake in.
  These arcs have no attributes and have a one-way direction:
  fashion_ model → *fashion_ exhibition*.

3.2. Data import

To import the data we extracted a CSV file for each table present in the relational database (defined in the first project), then proceeded to define the nodes and arcs following the guide linked in the project delivery. In particular, we now report the code and some information about it.

Nodes creation:

- Creation of ***fashion_ designer* nodes**:



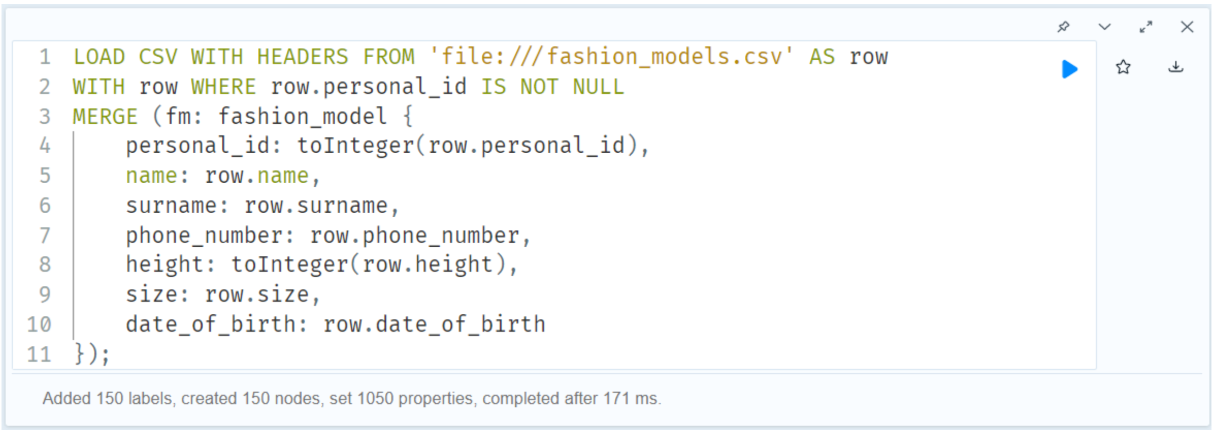
```
LOAD CSV WITH HEADERS FROM 'file:///fashion_designers.csv' AS row
WITH row WHERE row.personal_id IS NOT NULL
MERGE (fd: fashion_designer {
    personal_id: toInteger(row.personal_id),
    name: row.name,
    surname: row.surname,
    phone_number: row.phone_number,
    stage_name: row.stage_name,
    career: row.career
});
```

```
Added 50 labels, created 50 nodes, set 300 properties, completed after 323 ms.
```

Figure 3.1: Cypher code for creating *fashion_ designer* nodes.

As you can see in Figure 3.1, 50 nodes have been created with the *fashion_ designer* label. Each node has 6 attributes.

- Creation of ***fashion_ model* nodes**:



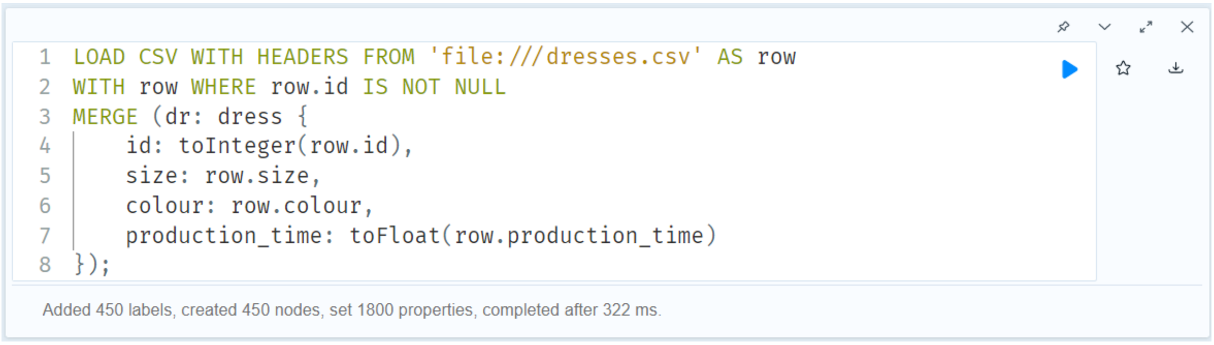
```
LOAD CSV WITH HEADERS FROM 'file:///fashion_models.csv' AS row
WITH row WHERE row.personal_id IS NOT NULL
MERGE (fm: fashion_model {
    personal_id: toInteger(row.personal_id),
    name: row.name,
    surname: row.surname,
    phone_number: row.phone_number,
    height: toInteger(row.height),
    size: row.size,
    date_of_birth: row.date_of_birth
});
```

```
Added 150 labels, created 150 nodes, set 1050 properties, completed after 171 ms.
```

Figure 3.2: Cypher code for creating *fashion_ model* nodes.

As you can see in Figure 3.2, 150 nodes have been created with the *fashion_ model* label. Each node has 7 attributes.

- Creation of ***dress* nodes**:



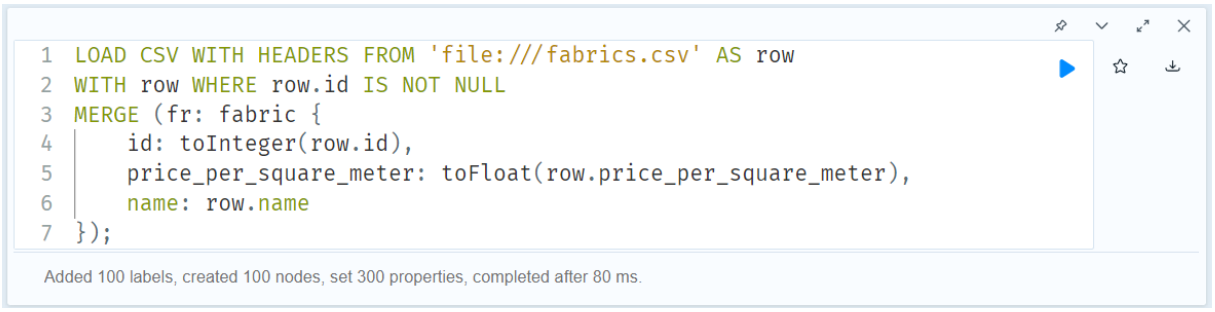
```
LOAD CSV WITH HEADERS FROM 'file:///dresses.csv' AS row
WITH row WHERE row.id IS NOT NULL
MERGE (dr: dress {
    id: toInteger(row.id),
    size: row.size,
    colour: row.colour,
    production_time: toFloat(row.production_time)
});
```

```
Added 450 labels, created 450 nodes, set 1800 properties, completed after 322 ms.
```

Figure 3.3: Cypher code for creating *dress* nodes.

As you can see in Figure 3.3, 450 nodes have been created with the *dress* label. Each node has 4 attributes.

- Creation of ***fabric* nodes**:



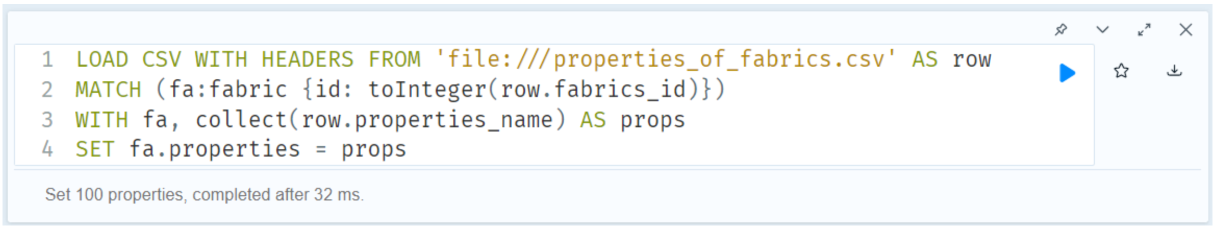
```
LOAD CSV WITH HEADERS FROM 'file:///fabrics.csv' AS row
WITH row WHERE row.id IS NOT NULL
MERGE (fr: fabric {
    id: toInteger(row.id),
    price_per_square_meter: toFloat(row.price_per_square_meter),
    name: row.name
});
```

Added 100 labels, created 100 nodes, set 300 properties, completed after 80 ms.

Figure 3.4: Cypher code for creating *fabric* nodes.

As you can see in Figure 3.4, 100 nodes have been created with the *fabric* label. Each node has 3 attributes.

- Creation of ***fabric*'s attribute *properties***:



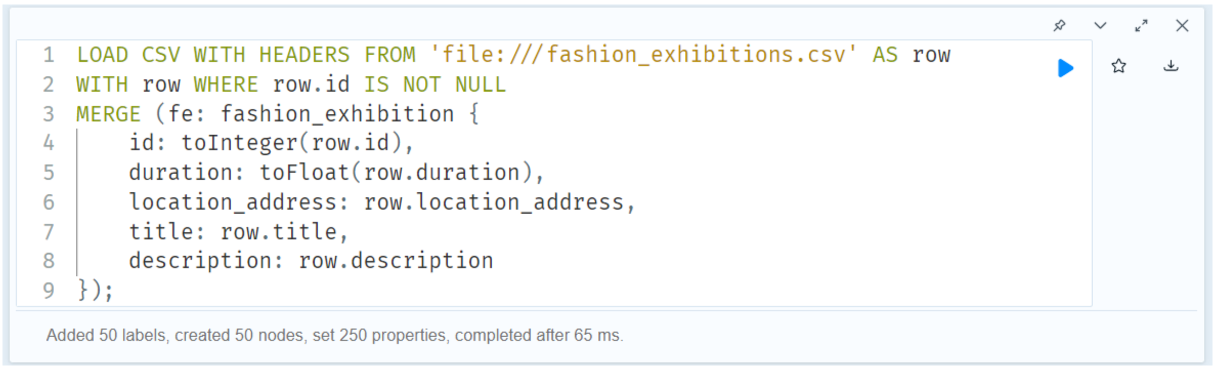
```
LOAD CSV WITH HEADERS FROM 'file:///properties_of_fabrics.csv' AS row
MATCH (fa:fabric {id: toInteger(row.fabrics_id)})
WITH fa, collect(row.properties_name) AS props
SET fa.properties = props
```

Set 100 properties, completed after 32 ms.

Figure 3.5: Cypher code for creating *fabric*'s attribute *"properties"*.

As you can see in Figure 3.5, 100 attributes have been added to the fabrics' nodes (1 list of properties for each node).

- Creation of ***fashion_ exhibition* nodes**:



```
LOAD CSV WITH HEADERS FROM 'file:///fashion_exhibitions.csv' AS row
WITH row WHERE row.id IS NOT NULL
MERGE (fe: fashion_exhibition {
    id: toInteger(row.id),
    duration: toFloat(row.duration),
    location_address: row.location_address,
    title: row.title,
    description: row.description
});
```

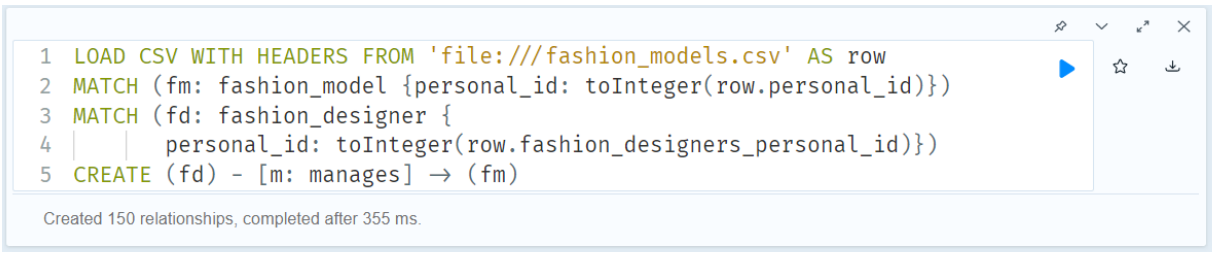
Added 50 labels, created 50 nodes, set 250 properties, completed after 65 ms.

Figure 3.6: Cypher code for creating *fashion_ exhibition* nodes.

As you can see in Figure 3.6, 50 nodes have been created with the *fashion_ exhibition* label. Each node has 5 attributes.

Arcs creation:

- Creation of ***manages* arcs**, connecting the designers to their respective models:



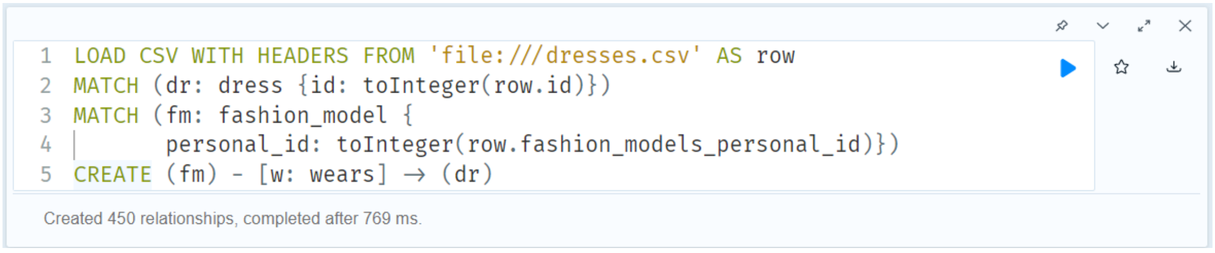
```
LOAD CSV WITH HEADERS FROM 'file:///fashion_models.csv' AS row
MATCH (fm: fashion_model {personal_id: toInteger(row.personal_id)})
MATCH (fd: fashion_designer {
        personal_id: toInteger(row.fashion_designers_personal_id)})
CREATE (fd) - [m: manages] → (fm)
```

```
Created 150 relationships, completed after 355 ms.
```

Figure 3.7: Cypher code for creating *manages* arcs.

As you can see in Figure 3.7, 150 arcs have been created with the *manages* label. These arcs have no attributes and they have a one-way direction: *fashion_ designer* → *fashion_ model*.

- Creation of ***wears* arcs**, connecting the models to the dresses they wear in exhibitions:



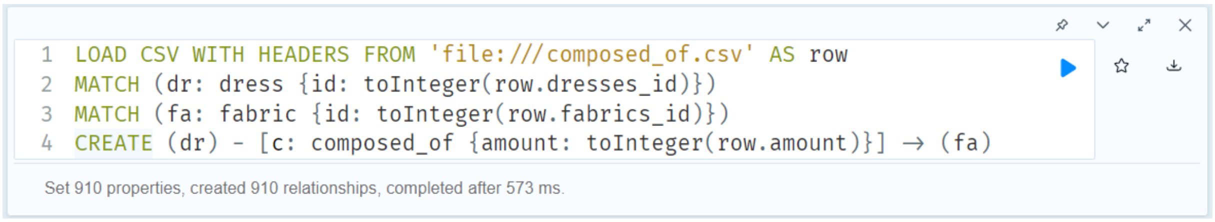
```
LOAD CSV WITH HEADERS FROM 'file:///dresses.csv' AS row
MATCH (dr: dress {id: toInteger(row.id)})
MATCH (fm: fashion_model {
        personal_id: toInteger(row.fashion_models_personal_id)})
CREATE (fm) - [w: wears] → (dr)
```

```
Created 450 relationships, completed after 769 ms.
```

Figure 3.8: Cypher code for creating *wears* arcs.

As you can see in Figure 3.8, 450 arcs have been created with the *wears* label. These arcs have no attributes and they have a one-way direction: *fashion_ model* → *dress*.

- Creation of ***composed_ of* arcs**, connecting the dresses with their respective fabrics:



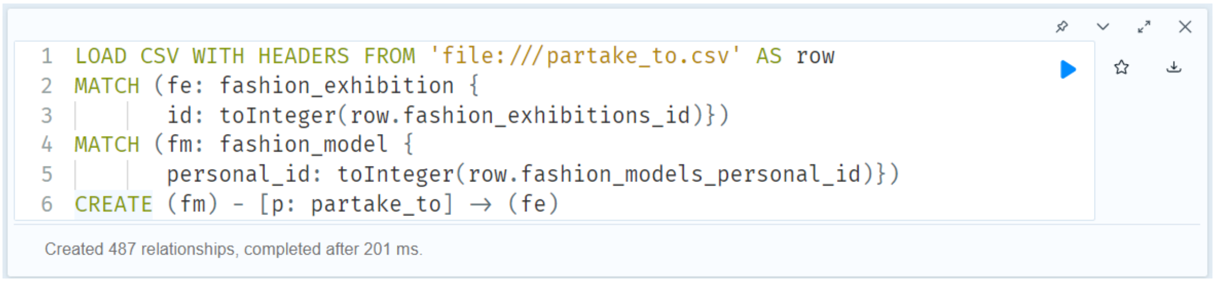
```
LOAD CSV WITH HEADERS FROM 'file:///composed_of.csv' AS row
MATCH (dr: dress {id: toInteger(row.dresses_id)})
MATCH (fa: fabric {id: toInteger(row.fabrics_id)})
CREATE (dr) - [c: composed_of {amount: toInteger(row.amount)}] → (fa)
```

```
Set 910 properties, created 910 relationships, completed after 573 ms.
```

Figure 3.9: Cypher code for creating *composed_ of* arcs.

As you can see in Figure 3.9, 910 arcs have been created with the *composed_ of* label.
Each arc has the attribute *amount*, which represents the percentage of the fabric present in the dress. The arcs have a one-way direction: *dress* → *fabric*.

- Creation of ***partake_ to* arcs**, connecting the models to the respective exhibitions in which they partook:



```
LOAD CSV WITH HEADERS FROM 'file:///partake_to.csv' AS row
MATCH (fe: fashion_exhibition {
        id: toInteger(row.fashion_exhibitions_id)})
MATCH (fm: fashion_model {
        personal_id: toInteger(row.fashion_models_personal_id)})
CREATE (fm) - [p: partake_to] → (fe)
```

```
Created 487 relationships, completed after 201 ms.
```

Figure 3.10: Cypher code for creating *partake_ to* arcs.

As you can see in Figure 3.10, 487 arcs have been created with the *partake_ to* label.
These arcs have no attributes and they have a one-way direction:
fashion_ model → *fashion_ exhibition*.

4 | Queries

In this chapter we present the queries used to interrogate the graph, dividing them by the type of request.

4.1. Create queries

Below we report the two queries used to create nodes and arcs.

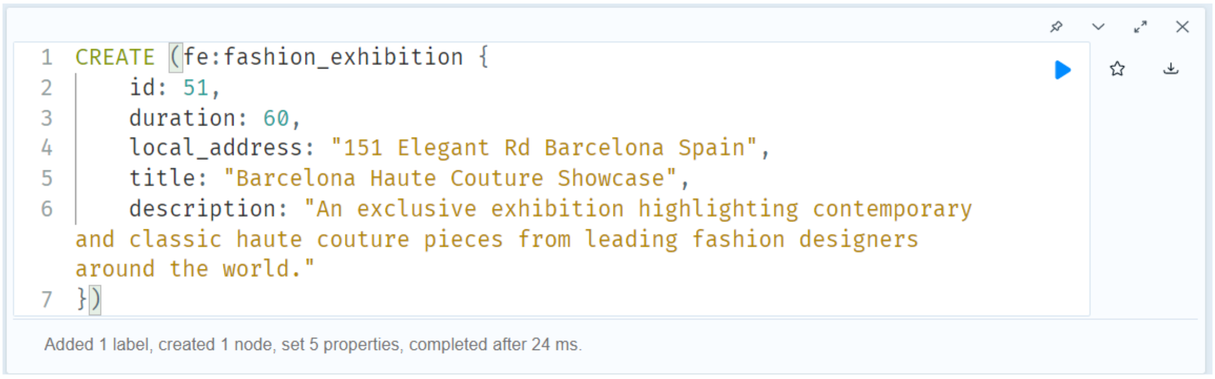
CREATE QUERY 1:

Request:

Create a new *fashion_ exhibition* node with the following attributes:

- *id*: 51
- *duration*: 60
- *description*: "An exclusive exhibition highlighting contemporary and classic haute couture pieces from leading fashion designers around the world."
- *location_ address*: "151 Elegant Rd Barcelona Spain."
- *title*: "Barcelona Haute Couture Showcase."

Cypher code:



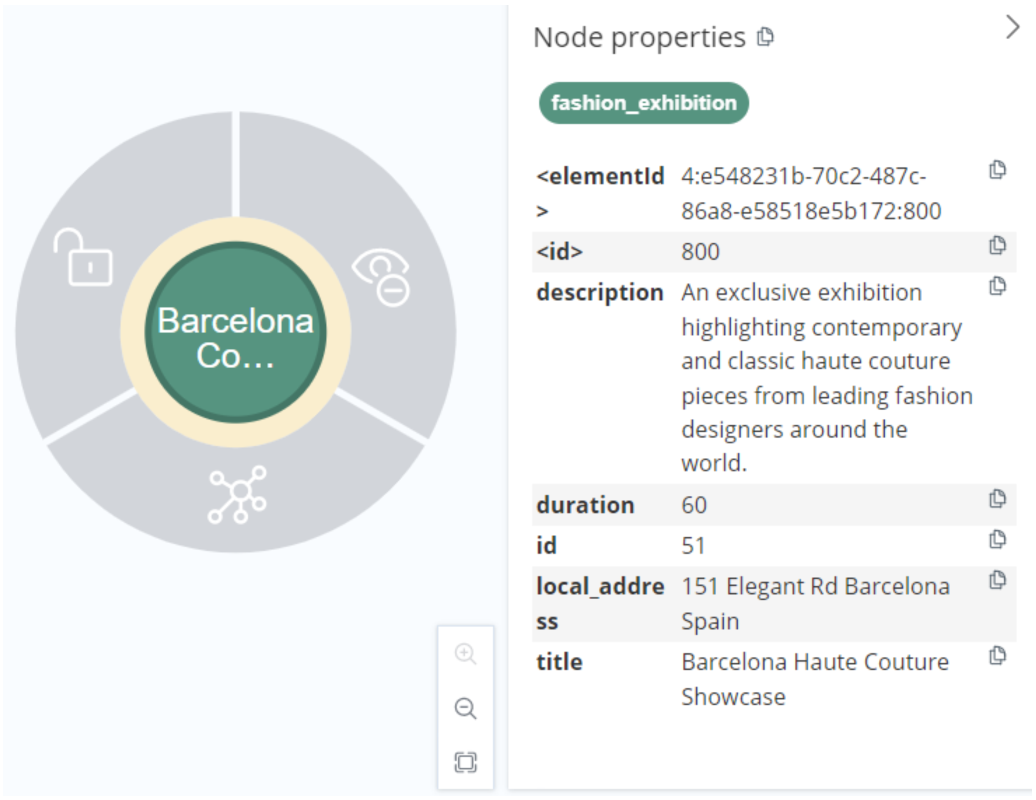
```
CREATE (fe:fashion_exhibition {
    id: 51,
    duration: 60,
    local_address: "151 Elegant Rd Barcelona Spain",
    title: "Barcelona Haute Couture Showcase",
    description: "An exclusive exhibition highlighting contemporary
and classic haute couture pieces from leading fashion designers
around the world."
})
```

```
Added 1 label, created 1 node, set 5 properties, completed after 24 ms.
```

Figure 4.1: Cypher code of "CREATE QUERY 1".

Output:

The *fashion_exhibition* node with such attributes is correctly added in the graph, as shown in Figure 4.2.



Node properties

fashion_exhibition

<elementId>	4:e548231b-70c2-487c-86a8-e58518e5b172:800
<id>	800
description	An exclusive exhibition highlighting contemporary and classic haute couture pieces from leading fashion designers around the world.
duration	60
id	51
local_address	151 Elegant Rd Barcelona Spain
title	Barcelona Haute Couture Showcase

Figure 4.2: "CREATE QUERY 1" output.

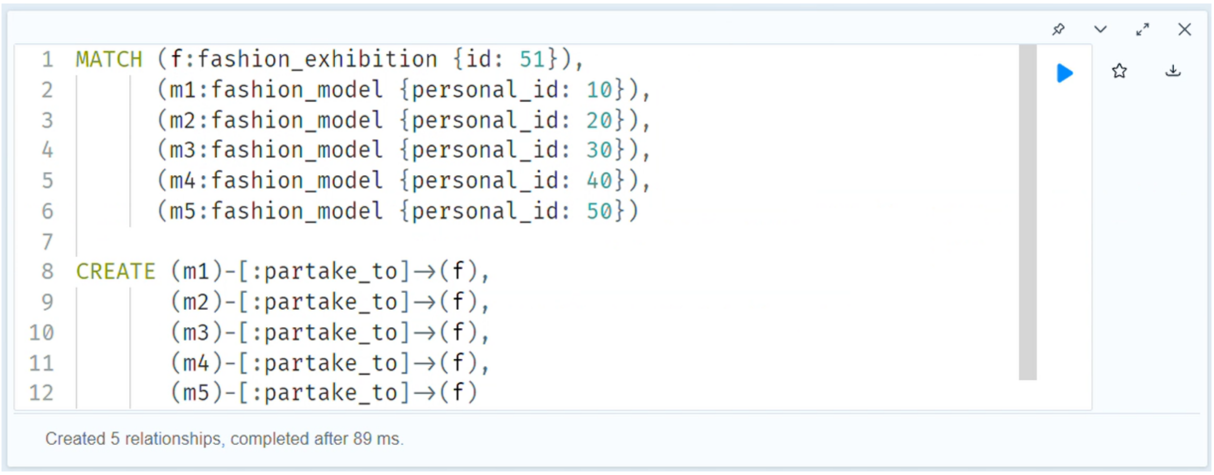
CREATE QUERY 2:

Request:

Given the previous added *fashion_ exhibition* node, create the arcs between such exhibition and the following models:

- Model with $personal\_id = 10$;
- Model with $personal\_id = 20$;
- Model with $personal\_id = 30$;
- Model with $personal\_id = 40$;
- Model with $personal\_id = 50$;

Cypher code:



```
MATCH (f:fashion_exhibition {id: 51}),
      (m1:fashion_model {personal_id: 10}),
      (m2:fashion_model {personal_id: 20}),
      (m3:fashion_model {personal_id: 30}),
      (m4:fashion_model {personal_id: 40}),
      (m5:fashion_model {personal_id: 50})

CREATE (m1)-[:partake_to]→(f),
       (m2)-[:partake_to]→(f),
       (m3)-[:partake_to]→(f),
       (m4)-[:partake_to]→(f),
       (m5)-[:partake_to]→(f)
```

```
Created 5 relationships, completed after 89 ms.
```

Figure 4.3: Cypher code of "CREATE QUERY 2".

Output:

As we can see in Figure 4.4, the arcs between the *fashion_ model* nodes and the previously created *fashion_ exhibition* node are correctly present in the graph.

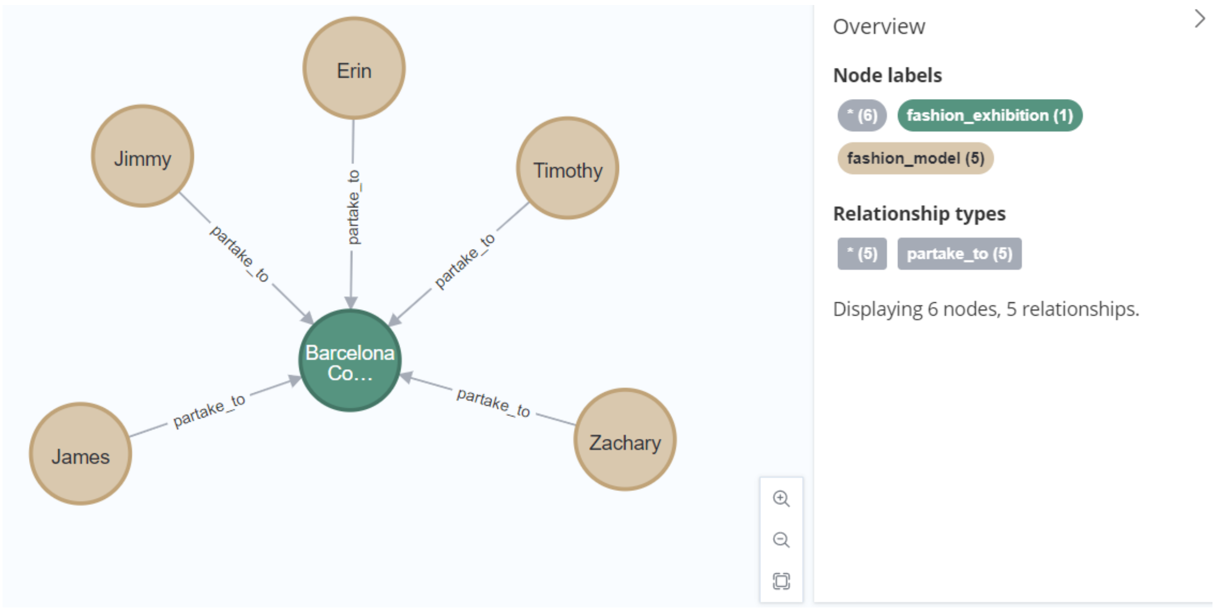


Figure 4.4: "CREATE QUERY 2" output.

4.2. Update queries

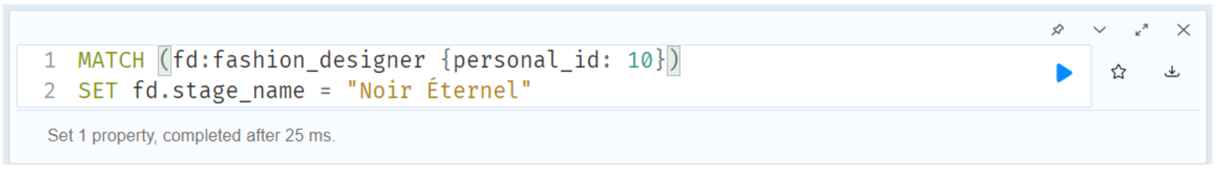
Below we report the two queries used to update nodes' attributes.

<u>UPDATE QUERY 1:</u>

Request:

> Update the *stage_ name* of the *fashion_ designer* node with *personal_ id* equal to 10 to "Noir Éternel".

Cypher code:

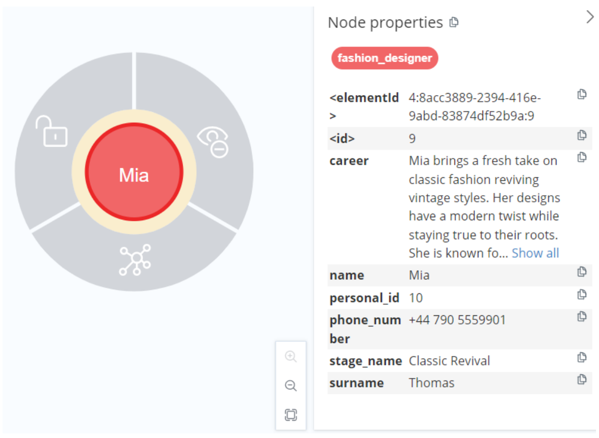


```
MATCH (fd:fashion_designer {personal_id: 10})
SET fd.stage_name = "Noir Éternel"
```

```
Set 1 property, completed after 25 ms.
```

Figure 4.5: Cypher code of "UPDATE QUERY 1".

Output:

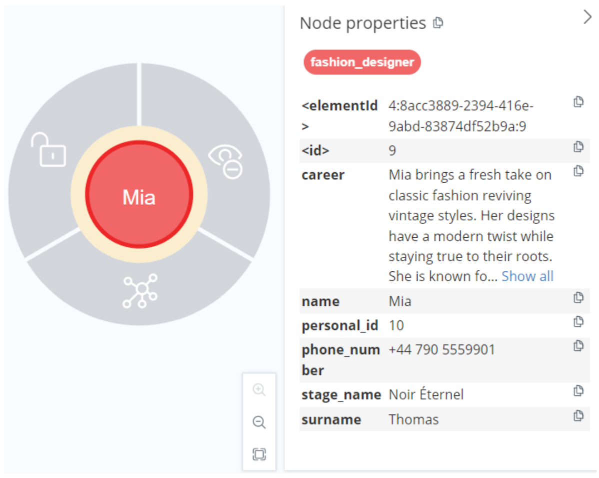


Node properties

fashion_designer

<elementId>	4:8acc3889-2394-416e-9abd-83874df52b9a:9
<id>	9
career	Mia brings a fresh take on classic fashion reviving vintage styles. Her designs have a modern twist while staying true to their roots. She is known fo... Show all
name	Mia
personal_id	10
phone_number	+44 790 5559901
stage_name	Classic Revival
surname	Thomas

Figure 4.6: Node before the query.



Node properties

fashion_designer

<elementId>	4:8acc3889-2394-416e-9abd-83874df52b9a:9
<id>	9
career	Mia brings a fresh take on classic fashion reviving vintage styles. Her designs have a modern twist while staying true to their roots. She is known fo... Show all
name	Mia
personal_id	10
phone_number	+44 790 5559901
stage_name	Noir Éternel
surname	Thomas

Figure 4.7: Node after the query.

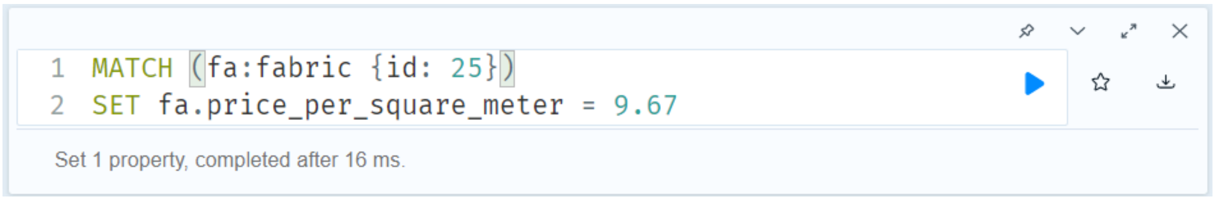
The two figures above show the *stage_name* of the designer searched before executing the query (Figure 4.6, old *stage_name*: "Classic Revivial") and after executing the query (Figure 4.7, new *stage_name*: "Noir Éternel").

UPDATE QUERY 2:

Request:

Update the *price_per_square_meter* of the *fabric* node with *id* equal to 25 to 9.67.

Cypher code:

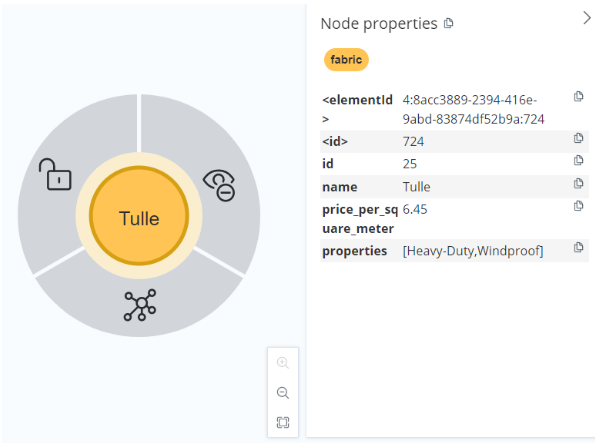


```
MATCH (fa:fabric {id: 25})
SET fa.price_per_square_meter = 9.67
```

```
Set 1 property, completed after 16 ms.
```

Figure 4.8: Cypher code of "UPDATE QUERY 2".

Output:

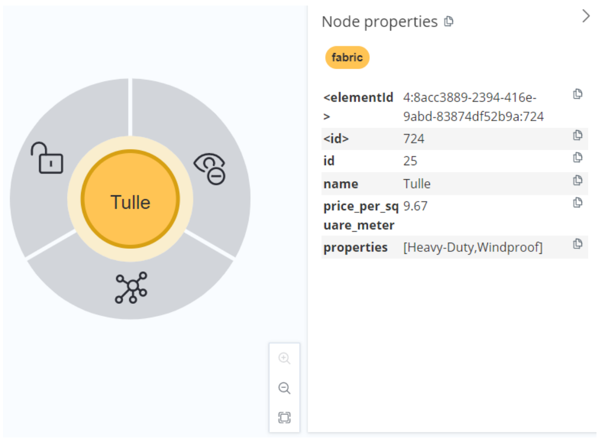


Node properties

fabric

<elementId>	4:8acc3889-2394-416e-9abd-83874df52b9a:724
<id>	724
id	25
name	Tulle
price_per_square_meter	6.45
properties	[Heavy-Duty,Windproof]

Figure 4.9: Node before the query.



Node properties

fabric

<elementId>	4:8acc3889-2394-416e-9abd-83874df52b9a:724
<id>	724
id	25
name	Tulle
price_per_square_meter	9.67
properties	[Heavy-Duty,Windproof]

Figure 4.10: Node after the query.

The two figures above show the *price_per_square_meter* of the *fabric* node before executing the query (Figure 4.9, old *price_per_square_meter*: 6.45) and after executing the query (Figure 4.10, new *price_per_square_meter*: 9.67).

4.3. Delete queries

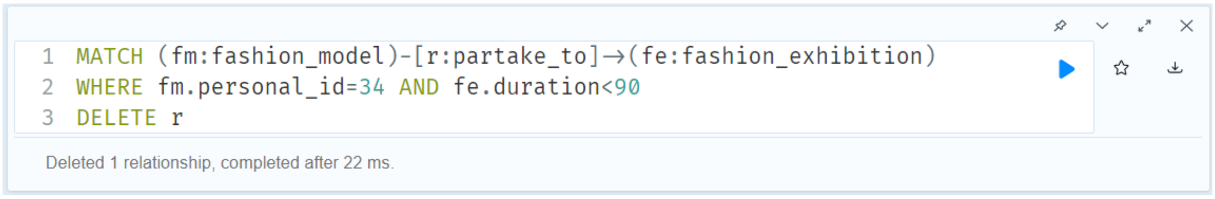
Below we report the two queries used to delete nodes and arcs.

DELETE QUERY 1:

Request:

Delete the arc between the *fashion_model* node with *personal_id* equal to 34 and exhibitions that he partakes in, but only the ones whose *duration* is less than 90 minutes.

Cypher code:



```
MATCH (fm:fashion_model)-[r:partake_to]→(fe:fashion_exhibition)
WHERE fm.personal_id=34 AND fe.duration<90
DELETE r
```

```
Deleted 1 relationship, completed after 22 ms.
```

Figure 4.11: Cypher code of "DELETE QUERY 1".

Output:

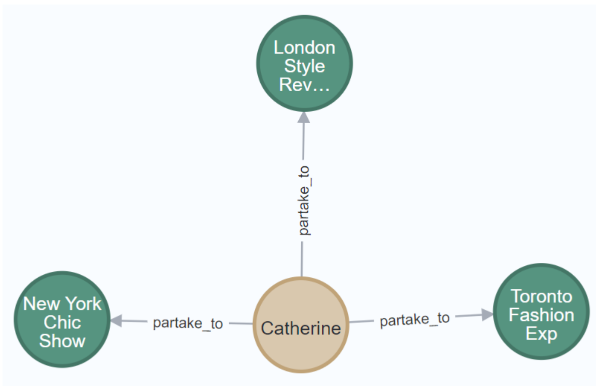


Figure 4.12: Nodes before the query.

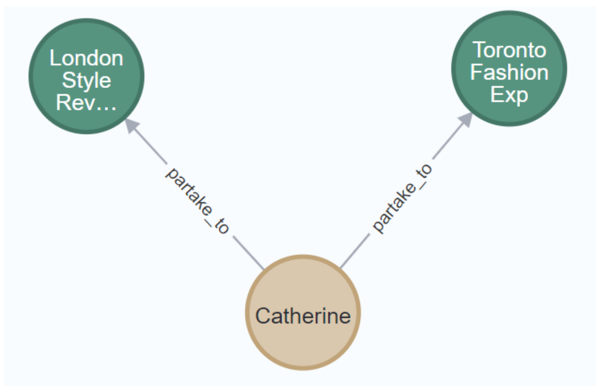


Figure 4.13: Nodes after the query.

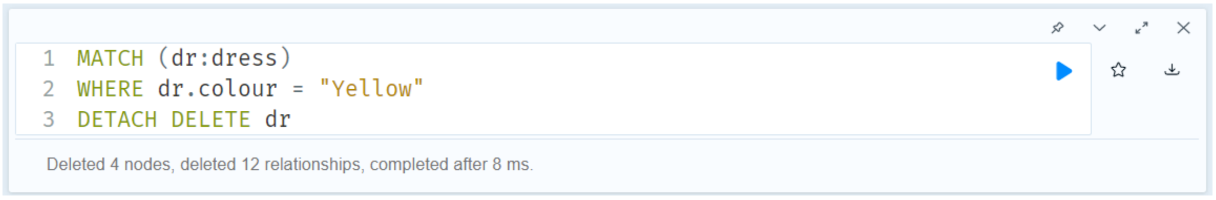
The two figures above show the sub-portion of the graph of interest, highlighting the node of the searched model and the nodes of the exhibitions in which he/she participates, also showing the arcs of interest.
It can be seen that before the execution of the query (Figure 4.12) the searched model participates in 3 exhibitions, however after the execution of the query (Figure 4.13) the searched model participates in only 2 exhibitions, thus removing the exhibition with *duration* less than 90 minutes.

DELETE QUERY 2:

Request:

Delete all the *dress* nodes that have yellow *colour* and the respective arcs, both with the *fabric* and the *fashion_ model* nodes.

Cypher code:



```
MATCH (dr:dress)
WHERE dr.colour = "Yellow"
DETACH DELETE dr
```

```
Deleted 4 nodes, deleted 12 relationships, completed after 8 ms.
```

Figure 4.14: Cypher code of "DELETE QUERY 2".

Output:

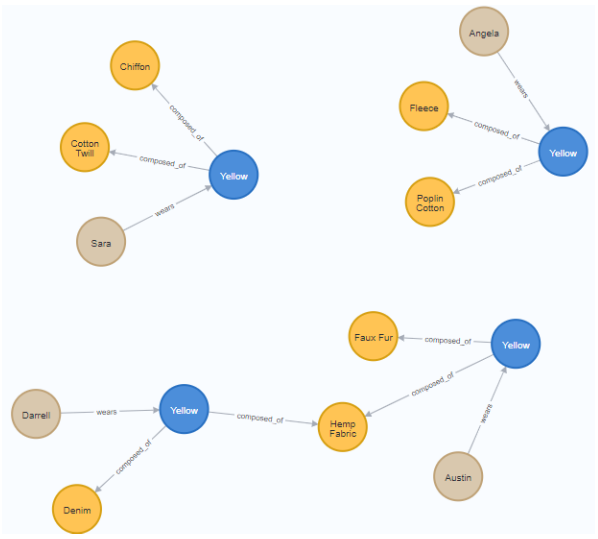
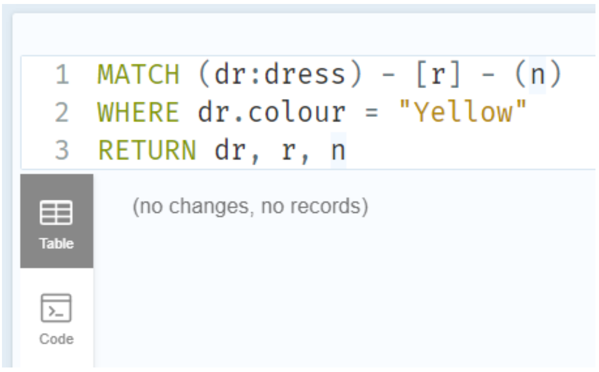


Figure 4.15: Before the query.



```
MATCH (dr:dress) - [r] - (n)
WHERE dr.colour = "Yellow"
RETURN dr, r, n
```

```
(no changes, no records)
```

Figure 4.16: After the query.

From the two figures above it can be seen that before the execution of the query (Figure 4.15) the *dress* nodes are present in the graph and also the arcs entering and exiting them, while after the execution of the query (Figure 4.16) such nodes are no longer present in the graph.
We can state that the arcs that connected these nodes with others are no longer present since the term DETACH has been used.

4.4. Queries with given complexities

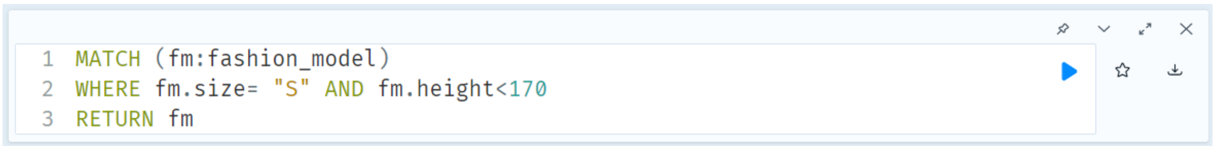
Below we report ten queries, with different types of complexity, used to interrogate the graph.

QUERY 1: At least 2 nodes in the MATCH statement and conditions.

Request:

Show all models who wear *size* S and are less than 170 centimeters tall.

Cypher code:



```
MATCH (fm:fashion_model)
WHERE fm.size= "S" AND fm.height<170
RETURN fm
```

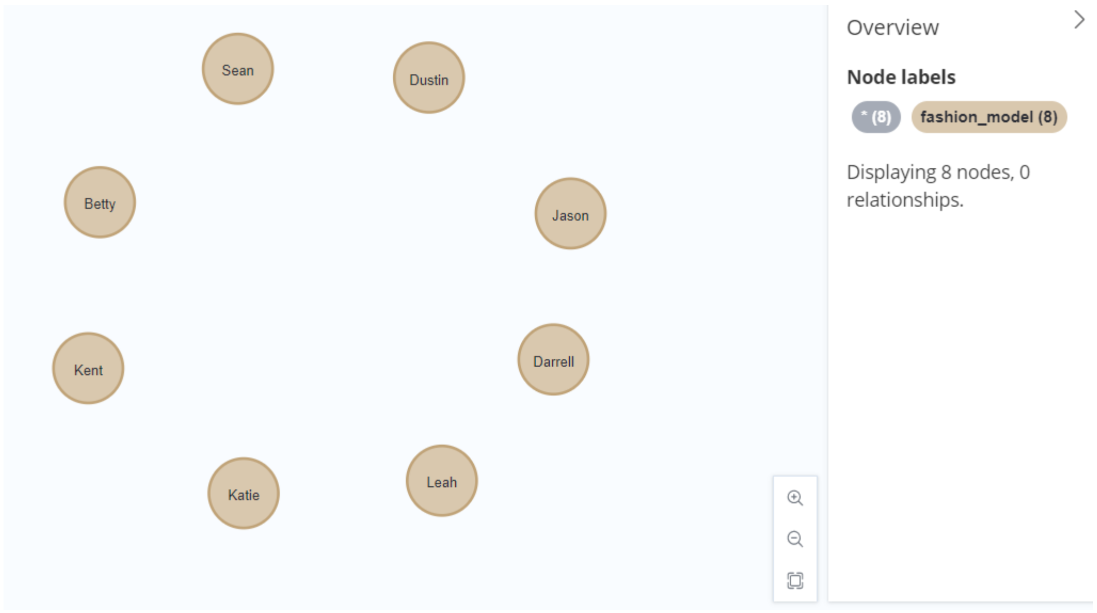
Figure 4.17: Cypher code of "QUERY 1".

Description:

The query performs a MATCH between the nodes present in the graph and only keeps those that have *fashion_model* as label.
Subsequently, these nodes are filtered using the WHERE clause, keeping only the nodes with the searched attributes.
Finally, these nodes are returned.

Output:

As you can see in Figure 4.18, 8 nodes with label *fashion_model* are returned as a result.



Displaying 8 nodes, 0 relationships.

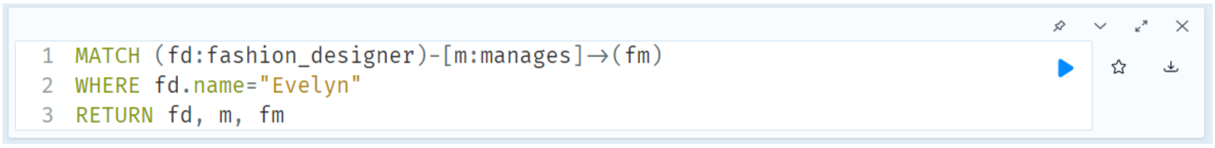
Figure 4.18: "QUERY 1" output.

QUERY 2: At least 2 nodes in the MATCH statement and conditions.

Request:

Find the models managed by a designer named Evelyn.
Show as a result the *designer* node, the nodes representing the models it manages and the arcs of interest.

Cypher code:



```
MATCH (fd:fashion_designer)-[m:manages]→(fm)
WHERE fd.name="Evelyn"
RETURN fd, m, fm
```

Figure 4.19: Cypher code of "QUERY 2".

Description:

The query identifies nodes in the graph that have the label *fashion_ designer* and are connected through a relationship labeled *manages* to other nodes. Using the WHERE clause, it filters these *fashion_ designer* nodes to only include those with the attribute *name* set to "Evelyn".
Finally, the query outputs the *fashion_ designer* node, the *manages* relationships, and the associated nodes that it manage.

Output:

As you can see in Figure 4.20, the designer named Evelyn, the 4 models she manages and the 4 arcs are returned as a result.

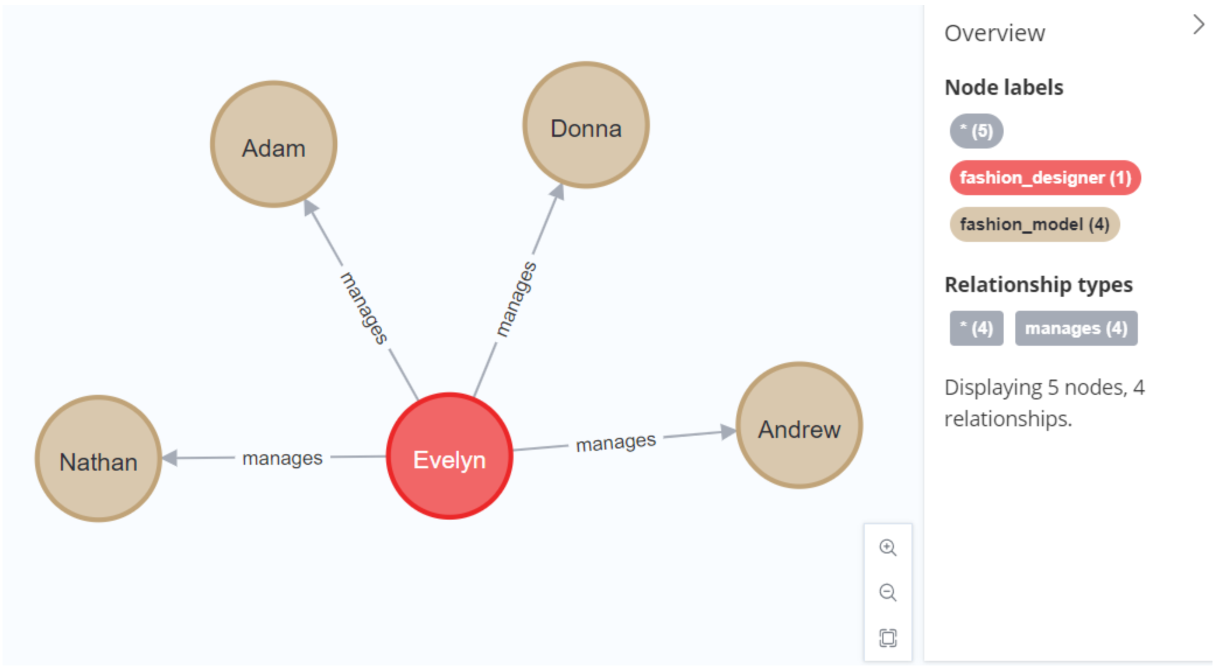


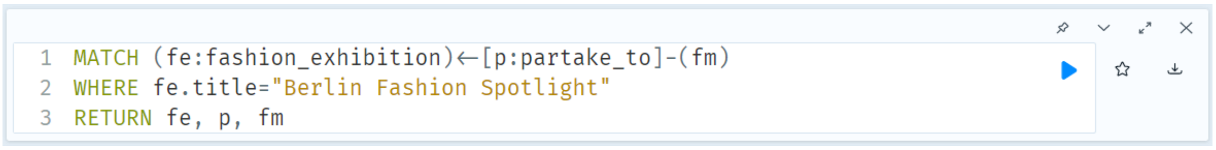
Figure 4.20: "QUERY 2" output.

QUERY 3: At least 2 nodes in the MATCH statement and conditions.

Request:

Find the models that partake in the exhibition with *title* “Berlin Fashion Spotlight”.
Return, as a result, the nodes of the searched models, the node of that exhibition, and the arcs that connect the models to the exhibition.

Cypher code:



```
MATCH (fe:fashion_exhibition)←[p:partake_to]-(fm)
WHERE fe.title="Berlin Fashion Spotlight"
RETURN fe, p, fm
```

Figure 4.21: Cypher code of "QUERY 3".

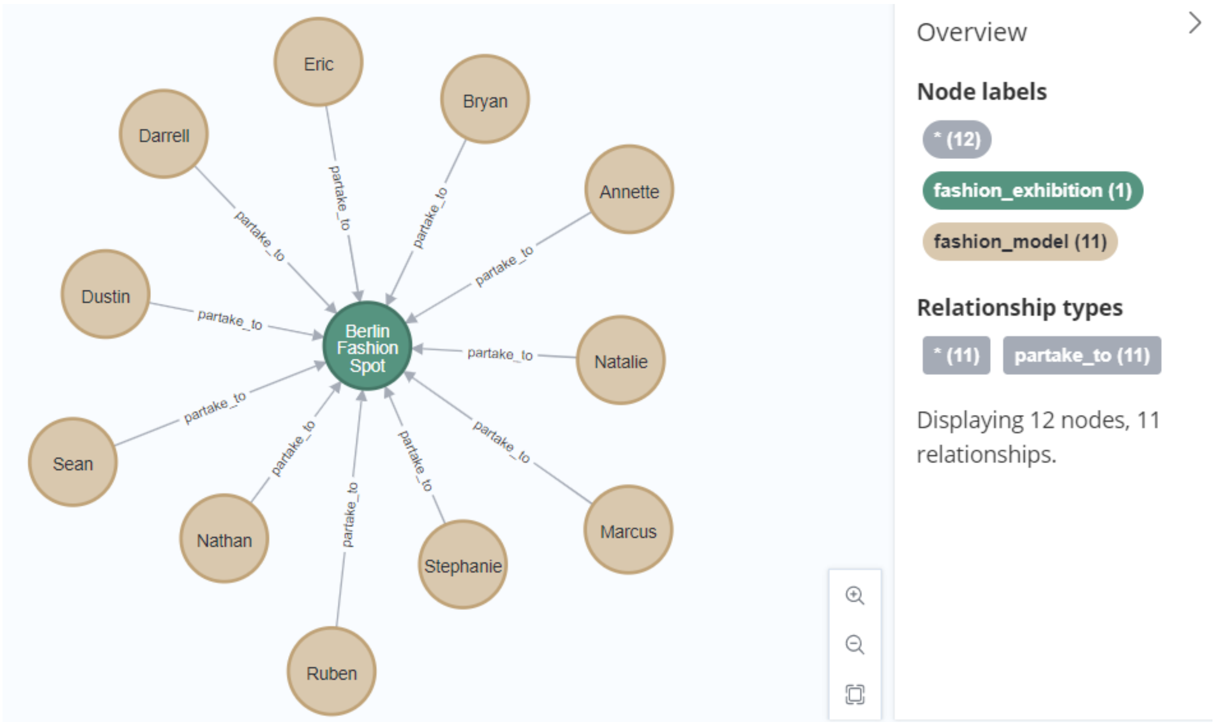
Description:

The query searches for nodes labeled *fashion_exhibition* and identifies incoming relationships of type *partake_to* that connect these exhibitions to other nodes.
It then filters the results to include only exhibitions with the *title* "Berlin

Fashion Spotlight".
Finally, the query returns the *fashion_exhibition* node, the *partake_to* relationships, and the connected *fashion_model* nodes.

Output:

As you can see in Figure 4.22, the exhibition entitled "Berlin Fashion Spotlight", the 11 models who participated in it and the 11 arcs are returned as a result.



Overview

Node labels

* (12)

fashion_exhibition (1)

fashion_model (11)

Relationship types

* (11) partake_to (11)

Displaying 12 nodes, 11 relationships.

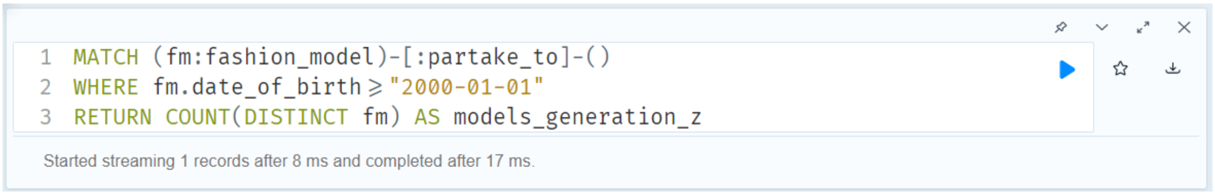
Figure 4.22: "QUERY 3" output.

QUERY 4: At least 2 nodes in the MATCH statement, conditions and aggregation without a WITH statement.

Request:

Find the models that can be considered within the generation Z (born after year 2000).
Return the counter of the nodes that matches such condition.

Cypher code:



```
MATCH (fm:fashion_model)-[:partake_to]-()
WHERE fm.date_of_birth >= "2000-01-01"
RETURN COUNT(DISTINCT fm) AS models_generation_z
```

Started streaming 1 records after 8 ms and completed after 17 ms.

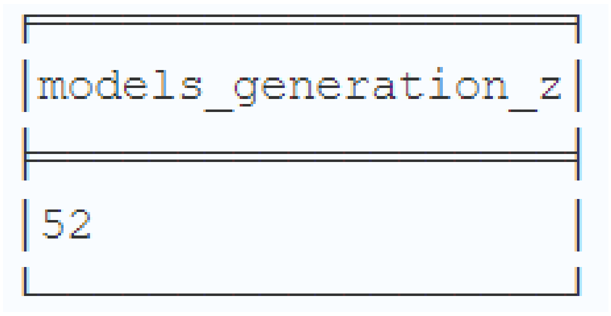
Figure 4.23: Cypher code of "QUERY 4".

Description:

The query looks for nodes labeled *fashion_model* and finds the *partake_to* relationships connecting them to other nodes.
It then filters the models based on the condition that their *date_of_birth* is equals or after January 1st 2000.
The query returns the count of distinct *fashion_model* nodes that satisfy this condition, which represents the number of models that can be considered within Generation Z.

Output:

As you can see in Figure 4.24, 52 models belong to generation Z.



```
╒═══════════════════╕
│models_generation_z│
╞═══════════════════╡
│52                 │
└───────────────────┘
```

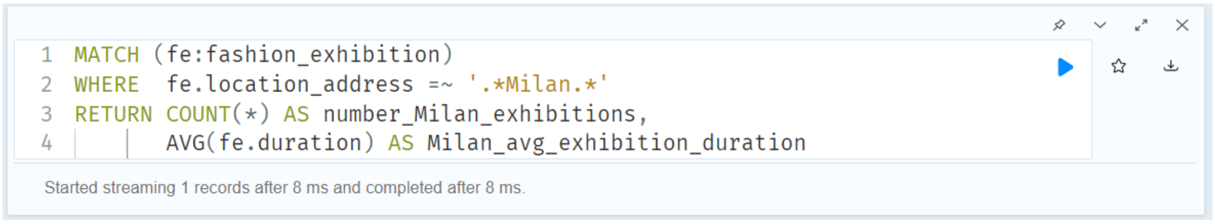
Figure 4.24: "QUERY 4" output.

QUERY 5: At least 2 nodes in the MATCH statement, conditions and aggregation without a WITH statement.

Request:

Find the exhibitions located in "Milan".
Return the number of exhibitions and their average *duration*.

Cypher code:



```
MATCH (fe:fashion_exhibition)
WHERE  fe.location_address =~ '.*Milan.*'
RETURN COUNT(*) AS number_Milan_exhibitions,
          AVG(fe.duration) AS Milan_avg_exhibition_duration
```

Started streaming 1 records after 8 ms and completed after 8 ms.

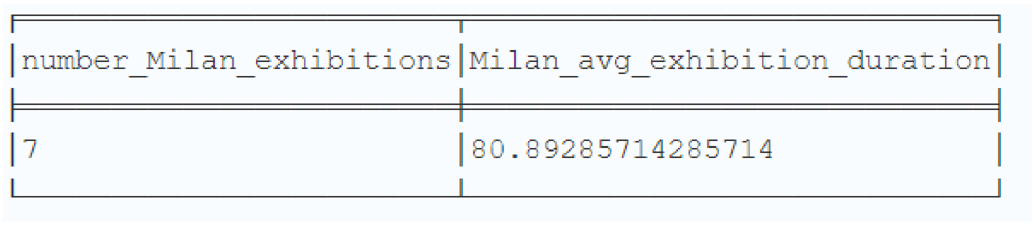
Figure 4.25: Cypher code of "QUERY 5".

Description:

The query searches for nodes labeled *fashion_ exhibition* and filters them based on their *location_ address* matching the pattern ".Milan.".
It then returns two metrics: the count of exhibitions that are located in "Milan" and the average *duration* of these exhibitions.

Output:

As you can see in Figure 4.26, 7 exhibitions are located in Milan, with an average *duration* of approximately 81 minutes.



number_Milan_exhibitions	Milan_avg_exhibition_duration
7	80.89285714285714

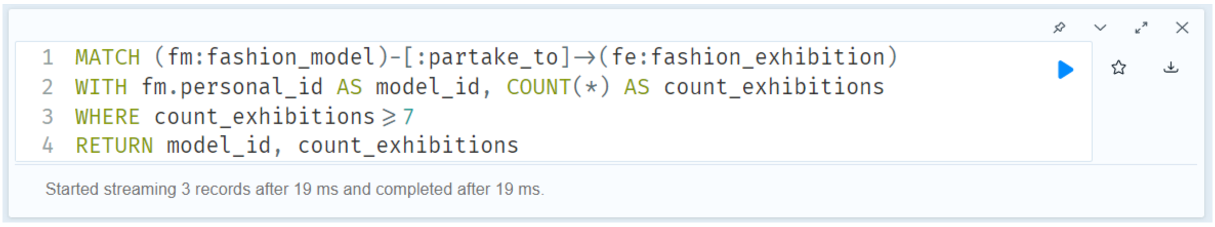
Figure 4.26: "QUERY 5" output.

QUERY 6: At least 2 nodes in the MATCH statement, conditions, and a WITH statement.

Request:

Find the models who have partaken in at least 7 exhibitions.
Return the couple (fashion model *personal_ id*, number of participation).

Cypher code:



```
MATCH (fm:fashion_model)-[:partake_to]→(fe:fashion_exhibition)
WITH fm.personal_id AS model_id, COUNT(*) AS count_exhibitions
WHERE count_exhibitions ≥ 7
RETURN model_id, count_exhibitions
```

Started streaming 3 records after 19 ms and completed after 19 ms.

Figure 4.27: Cypher code of "QUERY 6".

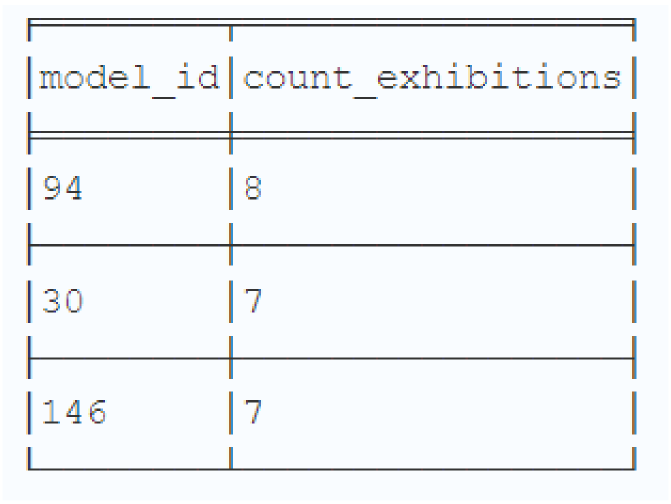
Description:

The query first looks for nodes labeled *fashion_ model* and identifies relationships of type *partake_ to* connecting them to *fashion_ exhibition* nodes.
It then counts the number of exhibitions each model participated in and filters the results to include only models who have participated in 7 or more exhibitions.
Finally, the query returns the *personal_ id* of those models along with the count of exhibitions they took part in.

Output:

As you can see in Figure 4.28:

- The model with *personal_ id* equal to 94 has partaken in 8 exhibitions.
- The model with *personal_ id* equal to 30 has partaken in 7 exhibitions.
- The model with *personal_ id* equal to 146 has partaken in 7 exhibitions.



model_id	count_exhibitions
94	8
30	7
146	7

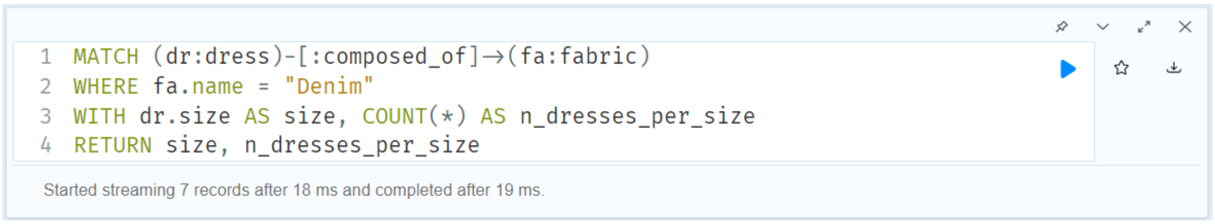
Figure 4.28: "QUERY 6" output.

QUERY 7: At least 2 nodes in the MATCH statement, conditions and a WITH statement.

Request:

Find the number of dresses, grouped by *size*, made up of at least "Denim" fabric.
Return the number of nodes per *size* that satisfy the condition.

Cypher code:



```
MATCH (dr:dress)-[:composed_of]→(fa:fabric)
WHERE fa.name = "Denim"
WITH dr.size AS size, COUNT(*) AS n_dresses_per_size
RETURN size, n_dresses_per_size
```

```
Started streaming 7 records after 18 ms and completed after 19 ms.
```

Figure 4.29: Cypher code of "QUERY 7".

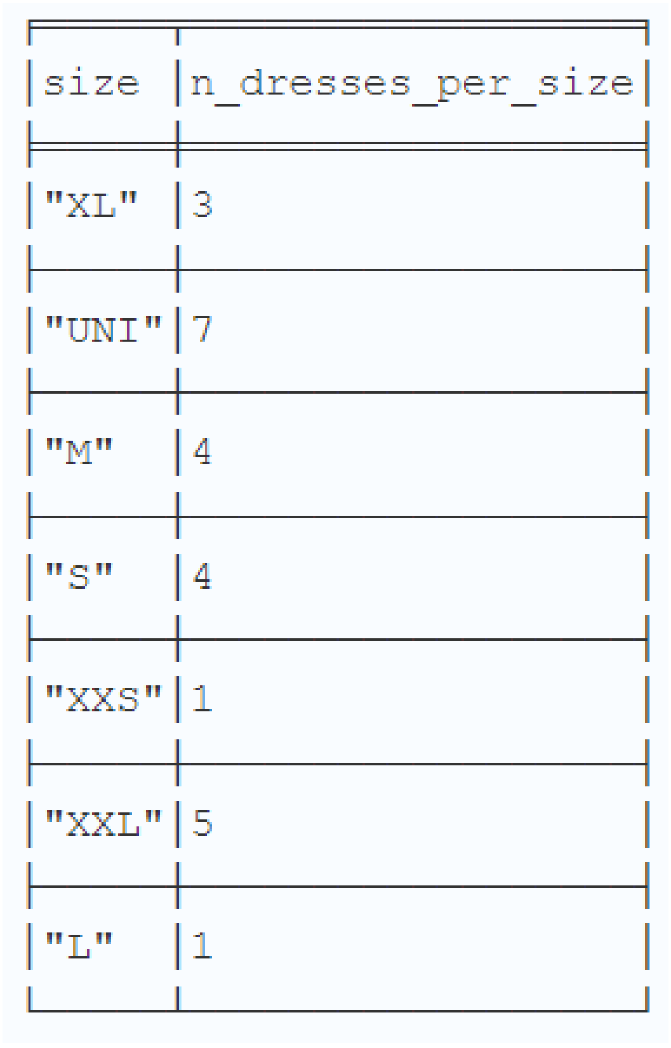
Description:

The query searches for nodes labeled *dress* and identifies relationships of type *composed_ of* connecting them to nodes labeled *fabric*.
It filters the results to include only those dresses that are composed of fabric with *name* "Denim".
The query then groups the *dress* nodes by their *size* and counts how many of them there are per *size*, returning the *size* and the corresponding count.

Output:

As you can see in Figure 4.30:

- There are 3 *dress* nodes with *size* XL made of Denim.
- There are 7 *dress* nodes with *size* UNI made of Denim.
- There are 4 *dress* nodes with *size* M made of Denim.
- There are 4 *dress* nodes with *size* S made of Denim.
- There is 1 *dress* nodes with *size* XXS made of Denim.
- There are 5 *dress* nodes with *size* XXL made of Denim.
- There is 1 *dress* nodes with *size* L made of Denim.



size	n_dresses_per_size
"XL"	3
"UNI"	7
"M"	4
"S"	4
"XXS"	1
"XXL"	5
"L"	1

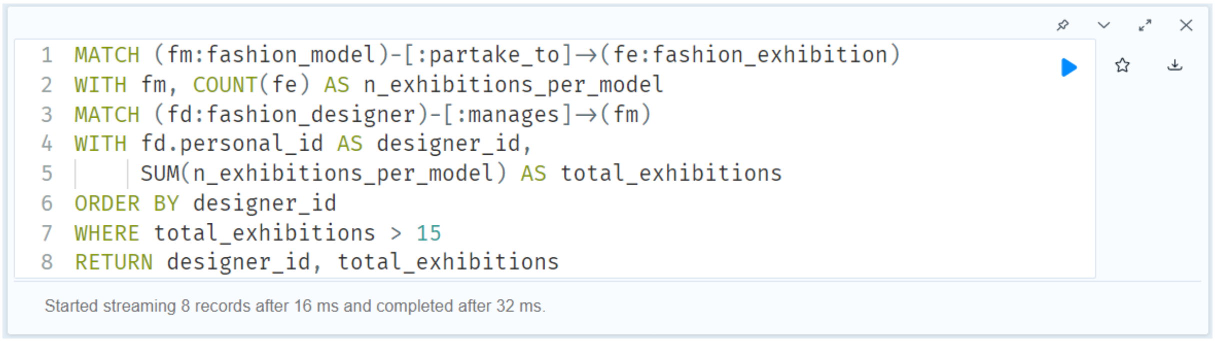
Figure 4.30: "QUERY 7" output.

QUERY 8: At least 3 nodes in the MATCH statement, conditions and multiple WITH statements.

Request:

> For each designer, find the total number of participation of his/her models in exhibitions. Then select only the designers whose models have more than 15 total participation.
> Return the couple (designer's *personal_id*, total number of participation of his/her models).

Cypher code:



```
MATCH (fm:fashion_model)-[:partake_to]→(fe:fashion_exhibition)
WITH fm, COUNT(fe) AS n_exhibitions_per_model
MATCH (fd:fashion_designer)-[:manages]→(fm)
WITH fd.personal_id AS designer_id,
     SUM(n_exhibitions_per_model) AS total_exhibitions
ORDER BY designer_id
WHERE total_exhibitions > 15
RETURN designer_id, total_exhibitions
```

Started streaming 8 records after 16 ms and completed after 32 ms.

Figure 4.31: Cypher code of "QUERY 8".

Description:

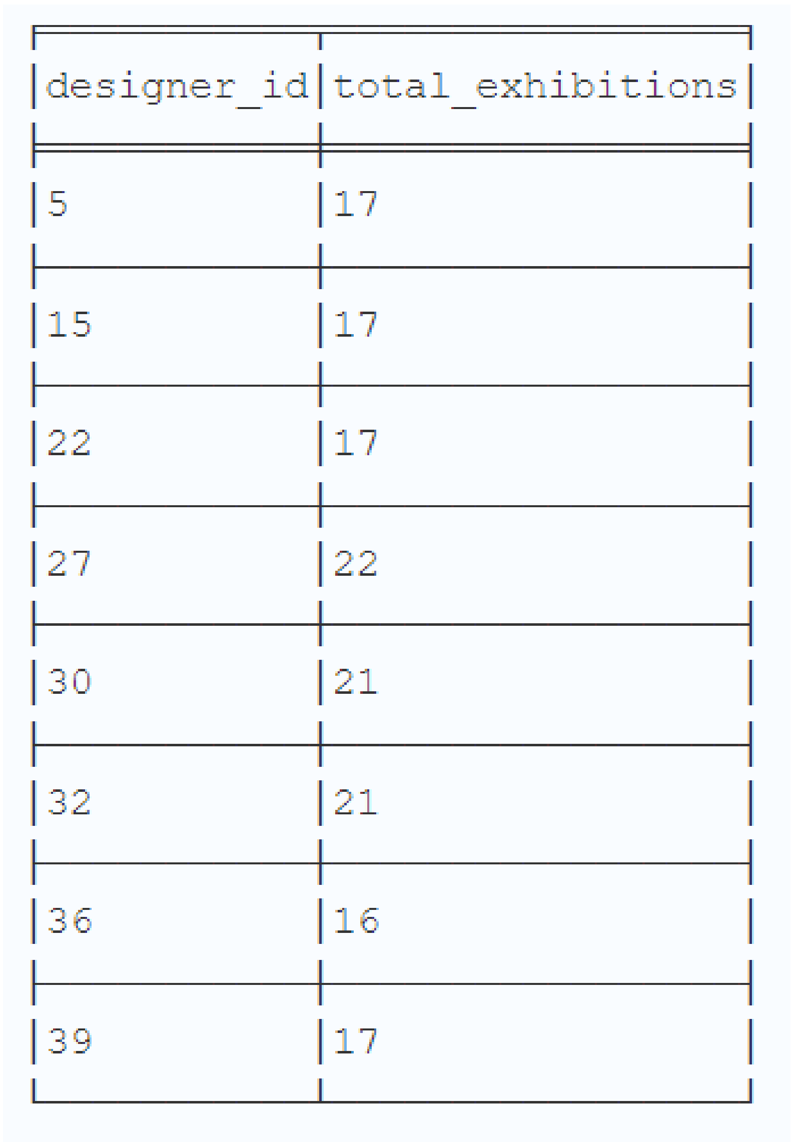
The query first looks for nodes labeled *fashion_model* and identifies the relationships of type *partake_to* connecting them to *fashion_exhibition* nodes. It then counts the number of exhibitions each model participated in. The query proceeds to match *fashion_designer* nodes managing the models and calculates the total number of exhibitions for each designer.
It filters the results to include only designers whose models have participated in more than 15 exhibitions, then returns the *personal_id* of these designers along with their total exhibition count."

Output:

As you can see in Figure 4.32:

- The models managed by the designer with *personal_id* set to 5 have collected a total of 17 participation.
- The models managed by the designer with *personal_id* set to 15 have collected a total of 17 participation.
- The models managed by the designer with *personal_id* set to 22 have collected a total of 17 participation.
- The models managed by the designer with *personal_id* set to 27 have collected a total of 22 participation.
- The models managed by the designer with *personal_id* set to 30 have collected a total of 21 participation.

- The models managed by the designer with *personal_id* set to 32 have collected a total of 21 participation.
- The models managed by the designer with *personal_id* set to 36 have collected a total of 16 participation.
- The models managed by the designer with *personal_id* set to 39 have collected a total of 17 participation.



```
╒═══════════╤═════════════════╕
│designer_id│total_exhibitions│
╞═══════════╪═════════════════╡
│5          │17               │
├───────────┼─────────────────┤
│15         │17               │
├───────────┼─────────────────┤
│22         │17               │
├───────────┼─────────────────┤
│27         │22               │
├───────────┼─────────────────┤
│30         │21               │
├───────────┼─────────────────┤
│32         │21               │
├───────────┼─────────────────┤
│36         │16               │
├───────────┼─────────────────┤
│39         │17               │
└───────────┴─────────────────┘
```

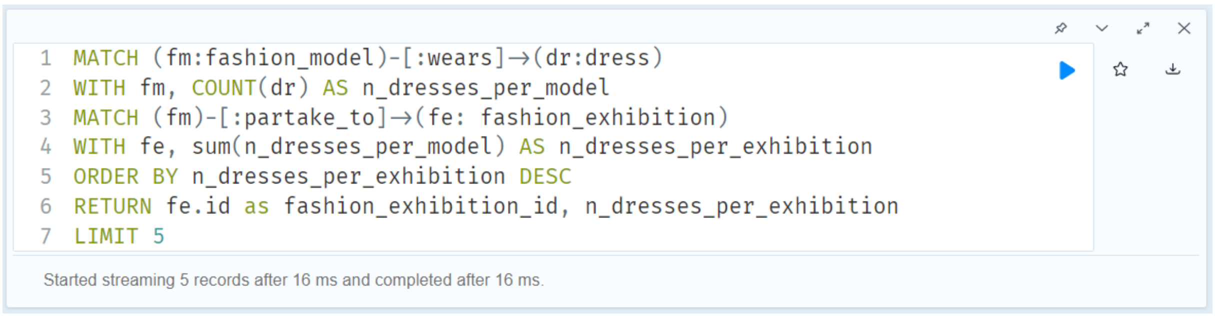
Figure 4.32: "QUERY 8" output.

QUERY 9: At least 3 nodes in the MATCH statement, conditions and multiple WITH statements.

Request:

Find the 5 exhibitions with the most number of dresses shown in them. Return the couple (fashion exhibition's *id*, total number of dresses shown in it).

Cypher code:



```
MATCH (fm:fashion_model)-[:wears]→(dr:dress)
WITH fm, COUNT(dr) AS n_dresses_per_model
MATCH (fm)-[:partake_to]→(fe: fashion_exhibition)
WITH fe, sum(n_dresses_per_model) AS n_dresses_per_exhibition
ORDER BY n_dresses_per_exhibition DESC
RETURN fe.id as fashion_exhibition_id, n_dresses_per_exhibition
LIMIT 5
```

```
Started streaming 5 records after 16 ms and completed after 16 ms.
```

Figure 4.33: Cypher code of "QUERY 9".

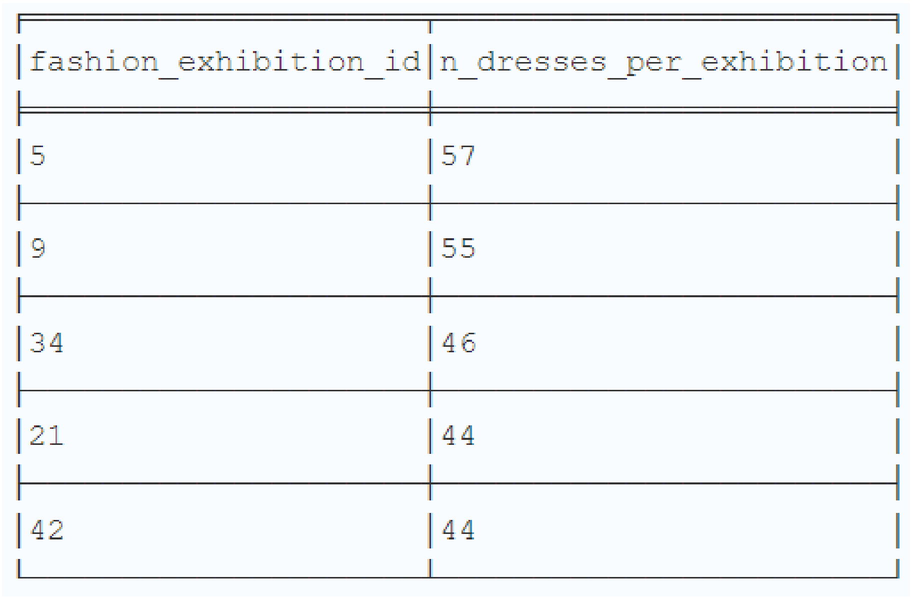
Description:

The query searches for nodes labeled *fashion_ model* and identifies relationships of type *wears* connecting them to *dress* nodes. It then counts how many dresses each model wears.
The query proceeds to find the *fashion_ exhibition* nodes where each model has participated. It sums the number of dresses worn by the models in each exhibition and orders the exhibitions by the total number of dresses worn in descending order.
The query returns the *id* of top 5 exhibitions based on this count.

Output:

As you can see in Figure 4.34:

- 57 dresses were shown at the exhibition with *id* equal to 5.
- 55 dresses were shown at the exhibition with *id* equal to 9.
- 46 dresses were shown at the exhibition with *id* equal to 34.
- 44 dresses were shown at the exhibition with *id* equal to 21.
- 44 dresses were shown at the exhibition with *id* equal to 42.



fashion_exhibition_id	n_dresses_per_exhibition
5	57
9	55
34	46
21	44
42	44

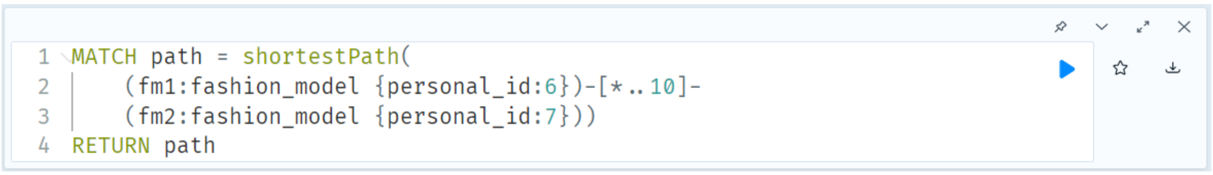
Figure 4.34: "QUERY 9" output.

QUERY 10: Use the shortestPath(...) function.

Request:

Find the shortest path (set an upper bound of 10 steps) between the model whose *personal_id* is 6 and the one that has *personal_id* equal to 7.
Return the path.

Cypher code:



```
MATCH path = shortestPath(
    (fm1:fashion_model {personal_id:6})-[*..10]-
    (fm2:fashion_model {personal_id:7}))
RETURN path
```

Figure 4.35: Cypher code of "QUERY 10".

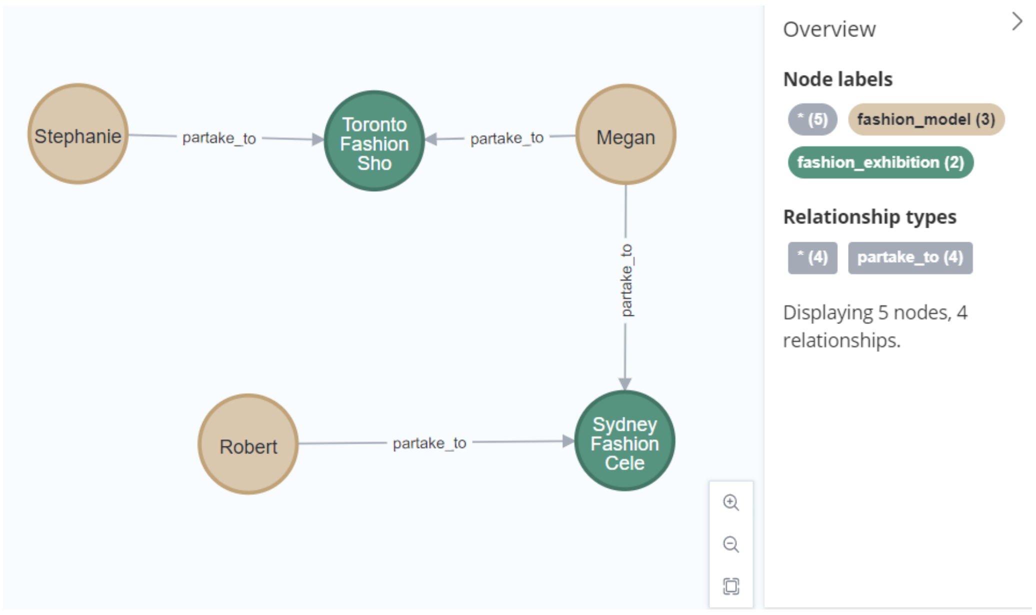
Description:

The query looks for the shortest path between two *fashion_model* nodes, where one has *personal_id* equals to 7 and the other has *personal_id* equals to 6. It searches for any possible paths of relationships up to a length of 10 between these two models.
The query then returns the found path.

Output:

As you can see in Figure 4.36, the shortest path to get to the model with *personal_id* set to 6 (called Stephanie), starting from the model with *personal_id* set to 7 (called Robert) is the following:

- **Robert** partook in the "Sydney Fashion Cele" exhibition.
- Megan also participated in the "Sydney Fashion Cele exhibition.
- Megan partook in the "Toronto Fashion Sho" exhibition.
- **Stephanie** also participated in the "Toronto Fashion Sho" exhibition.



Displaying 5 nodes, 4 relationships.

Figure 4.36: "QUERY 10" output.

5 | Business Process Management

5.1. Text delivery

We now report the business case to be analysed in the second part of this project.
“Models submit their CVs through the company’s front desk. As soon as a CV is received, it is forwarded to the HR department, which checks it. If they deem the candidate suitable for a position, they forward the CV to the head of the company. Otherwise, the candidate is noticed, and the process terminates. The head of the company contacts the fashion designers (internals) who work for the company and submits the model’s CV to them. Designers have three days to reply to the head (a response is mandatory, even if negative). If a designer is interested in working with the model, the head asks the front desk to contact the model to set up a meeting with them. If no designer is interested, the candidate is notified, and the process terminates. If the model answers affirmatively, the designer is contacted through the front desk, and they are asked to prepare a dress for the model. The dress is sent to the model, and then a meeting with the head is held. After the meeting, the head of the company provides their final decision through the front desk. They have at most 5 days to do so. Otherwise, the candidate is notified, and the process terminates.”

5.2. Text analysis

5.2.1. Actors Identification

Six actors are involved: the company, the front desk, the HR department, the head of the company, the fashion designers, and the model. These actors can be grouped into:

- The company and its internal entities:
  - The front desk;
  - The HR department;
  - The head of the company;

  - The fashion designers.

- The models.

Therefore, we have:

- Two pools: the company and the model.
- The first pool has four lanes: the front desk, the HR department, the head of the company, and the fashion designers.
- The second pool has no lanes.

5.2.2. Task Identification

CV Submission and Evaluation

Models submit their CVs through the company's front desk. As soon as a CV is received, it is forwarded to the HR department, which checks it. If the candidate is deemed suitable for a position, the CV is forwarded to the head of the company. Otherwise, the candidate is notified, and the process terminates.

Assumption: The candidate is informed of the rejection by the front desk (later in the process, it is explicitly mentioned that "the company provides their final decision through the front desk").

Tasks for each actor involved during this phase:

- **Front Desk:**
  - Receive CV (event);
  - Forward CV to HR department;
  - Receive rejection notice (optional event);
  - Send rejection notice to model (optional).
- **HR Department:**
  - Receive CV (event);
  - Evaluate CV;
  - Forward CV to head of company (optional);
  - Send rejection notice to front desk (optional).

- **Model:**
  - Submit CV to front desk;
  - Receive rejection notice (optional event).

Interaction with Fashion Designers

The head of the company contacts the fashion designers (internals) who work for the company and submits the model's CV to them. Designers have three days to reply to the head (a response is mandatory, even if negative). If a designer is interested in working with the model, the head asks the front desk to contact the model to set up a meeting with them. If no designer is interested, the candidate is notified, and the process terminates.

Assumptions:

- Contacting and submitting the model's CV to the fashion designers is the same task.
- If more than one fashion designer is interested, the head of the company chooses one of them.

Tasks for each actor involved during this phase:

- **Head of the Company:**
  - Receive CV (event);
  - Submit CV to the fashion designers (multi-instance: three designers are contacted);
  - Receive responses from fashion designers (multi-instance: three answers received from the designers);
  - Evaluate responses of fashion designers.
  - Send meeting request to front desk (optional);
  - Send rejection notice to front desk (optional).
- **Fashion Designers:**
  - Receive CV (event);
  - Evaluate CV;
  - Send response to head of the company (maximum three days to reply).

- **Front Desk:**
  - Receive meeting request (optional event);
    - Send meeting request to model.
  - Receive rejection notice (optional event);
    - Send rejection notice to model.
- **Model:**
  - Receive rejection (optional event);
  - Receive meeting request (optional event).

Meeting and Final Decision

If the model answers affirmatively, the designer is contacted through the front desk, and they are asked to prepare a dress for the model. The dress is sent to the model, and then a meeting with the head is held. After the meeting, the head of the company provides their final decision through the front desk. They have at most five days to do so. Otherwise, the candidate is notified, and the process terminates.

Assumptions:

- The dress is sent to the model directly from the designer.
- If the model refuses the meeting request, the head of the company is informed through the front desk.
- The automatic rejection after five days is sent as a negative final decision. It is not necessary to separate the automatic negative decision from the head of the company's negative decision, as there is no difference for the model (both are rejections).

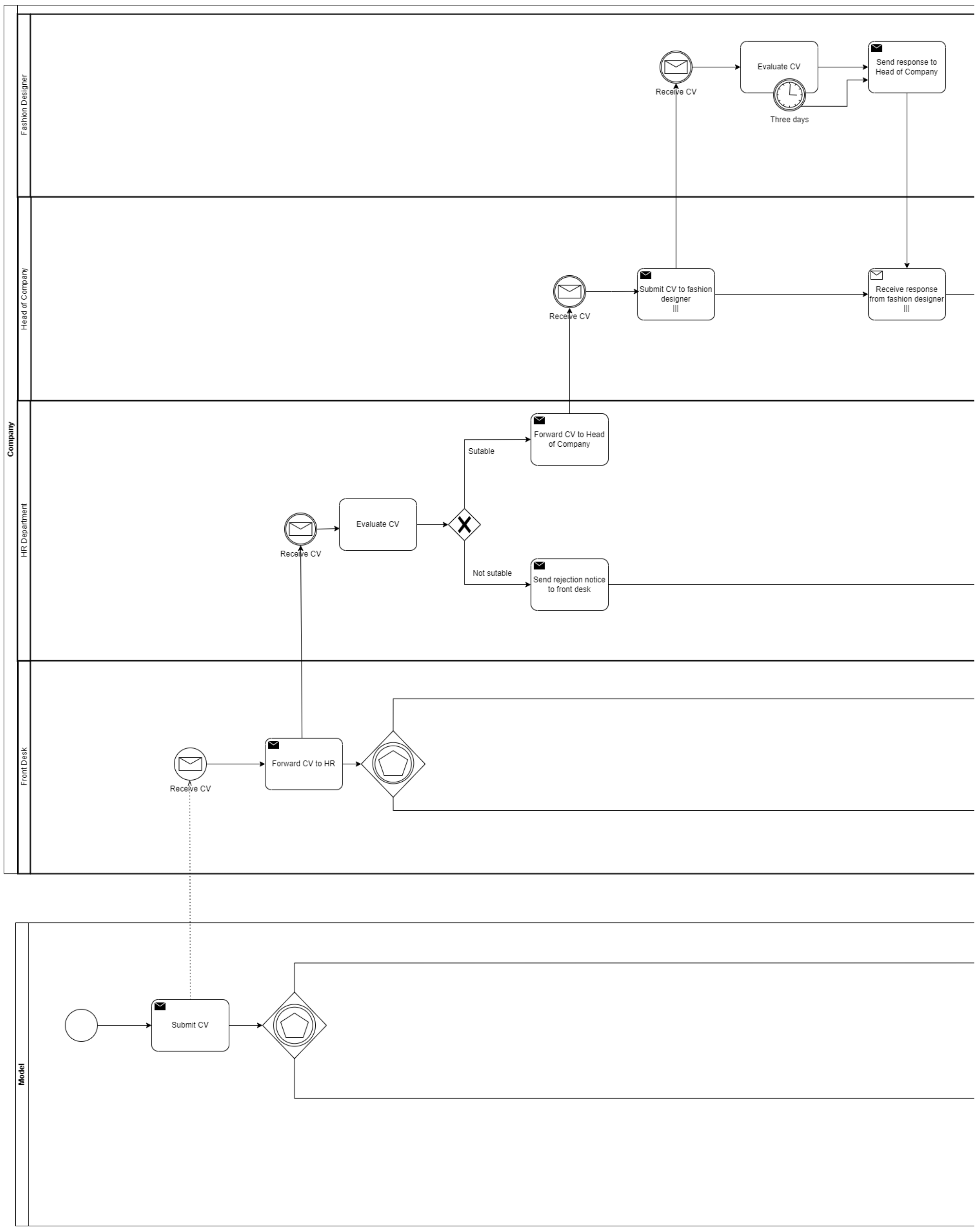
Tasks for each actor involved during this phase:

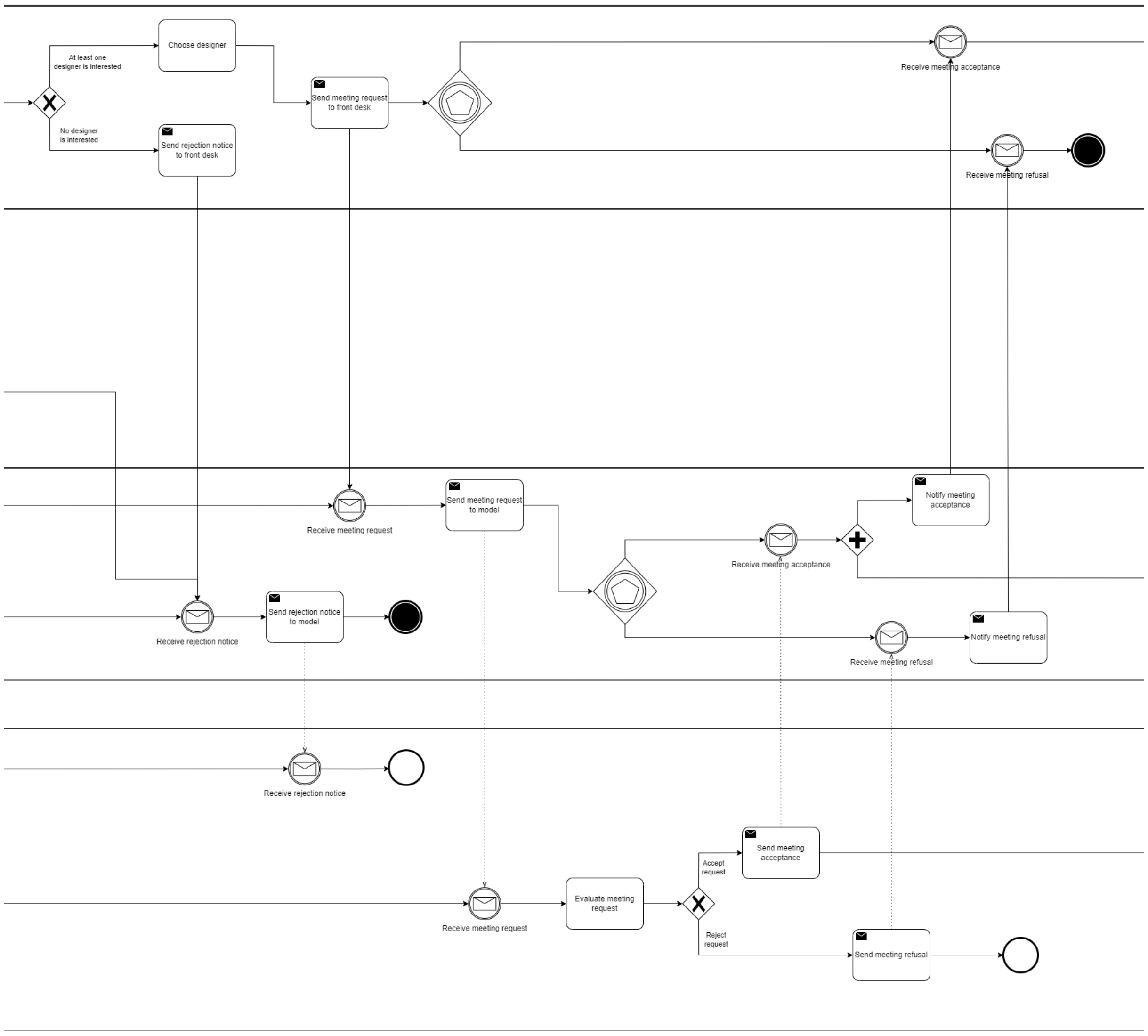
- **Model:**
  - Evaluate meeting request;
  - Send meeting acceptance (optional);
  - Send meeting refusal (optional);
  - Receive dress (event);

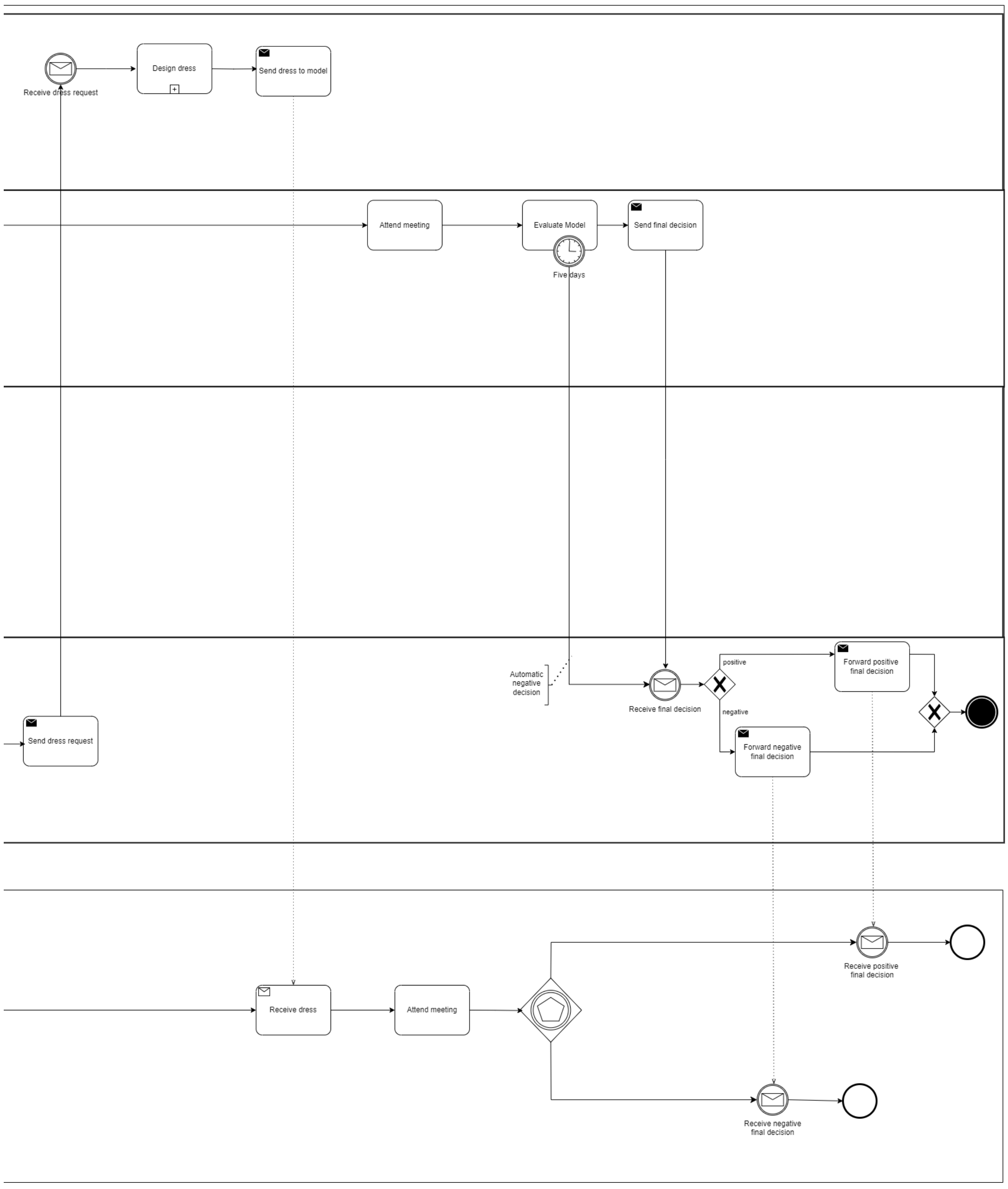
  - Attend meeting;
  - Receive positive final decision (optional event);
  - Receive negative final decision (optional event).
- **Front Desk:**
  - Receive meeting acceptance (optional event);
  - Receive meeting refusal (optional event);
  - Notify meeting acceptance (optional);
  - Notify meeting refusal (optional);
  - Send dress request;
  - Receive final decision (event);
  - Forward negative final decision (optional);
  - Forward positive final decision (optional).
- **Head of the Company:**
  - Receive meeting acceptance (optional event);
  - Receive meeting refusal (optional event);
  - Attend meeting;
  - Evaluate model;
  - Send final decision (maximum five days to send it).
- **Fashion Designer:**
  - Receive dress request (event);
  - Design dress (sub-process);
  - Send dress to model.

5.3. BPMN chart of the process

We now report a series of figures of the process for better reading.
Among the files delivered there are also the ".drawio" file and a ".png" image, for a complete view of the process.


Company
Fashion Designer
Head of Company
HR Department
Front Desk
Model
Receive CV
Evaluate CV
Three days
Send response to Head of Company
Submit CV to fashion designer
Receive response from fashion designer
Forward CV to Head of Company
Sutable
Not sutable
Send rejection notice to front desk
Forward CV to HR
Submit CV
]


Choose designer
At least one designer is interested
No designer is interested
Send rejection notice to front desk
Send meeting request to front desk
Receive meeting acceptance
Receive meeting refusal
Receive meeting request
Send meeting request to model
Receive meeting acceptance
Notify meeting acceptance
Receive rejection notice
Send rejection notice to model
Receive meeting refusal
Notify meeting refusal
Receive rejection notice
Send meeting acceptance
Accept request
Receive meeting request
Evaluate meeting request
Reject request
Send meeting refusal


Receive dress request
Design dress
Send dress to model
Attend meeting
Evaluate Model
Send final decision
Five days
Automatic negative decision
Receive final decision
positive
negative
Forward positive final decision
Forward negative final decision
Send dress request
Receive dress
Attend meeting
Receive positive final decision
Receive negative final decision

6 | Conclusion

The second Enterprise ICT Architectures project involved defining a graph database and analyzing a business process for designing an operational flow.
We utilized Neo4j for the creation of the graph database and Cypher for querying it, ensuring the accuracy and functionality of the designed queries.
For the BPMN modeling, draw.io was employed to represent the business process with clarity and detail.
This project was very stimulating and enhanced our understanding of both graph databases and process modeling, providing us with key skills for designing and analyzing complex business processes.