CIS 761 SPRING 2022– Group Project
Employee Complaints and Accidents Database

Group Members : Doug Baker, Krish Weragalaarachchi

----

1. Introduction:

The Chicago Employee Complaint and Accident Database can be used to track issues regarding various employees' accidents and complaints. This database can be used by administration officials to keep track of employee history.

All employees in the system get a unique ID and, the system keeps information on employees first and last name, their annual salary, salary category (low, medium, or high), number of promotions achieved, number of projects finished, start year, department that they work for and position they hold. It also keeps records of accidents that they involved and complaints that they received along with its resolution status.

If a new employee joins or resigns, the database users can easily update or delete the employee from the database. Accident types are defined by specific code like A1, A2, A3 ... etc. Similarly, complaint types also have codes and descriptions. If there is any new type of accident or complaint that is not recorded in the existing database happens, the database can be easily updated by adding them to Accident Types table or Compliant Codes table. When an accident happens, each employee involved in the accident get a different accident_id. When an employee receives a complaint, it will be recorded in the database under a certain complaint ID. In case of the complaint is against multiple employees they will be under the same complaint ID but they can have different resolution statuses depending on their conduct.

2. Review the SQL component:

2.1 A technical description of the system, design criteria, and requirements

We built up the Employee Compliant and Accident database in

PostgreSQL. The data is from the City of Chicago Data portal and data has been de-identified and fictionalized to some extent to ensure the anonymity of the real people in the dataset. Most of it is real data. The data set consists of 7 CSV files.

We have created 7 tables in PostgrSQL to insert these files and then used those 7 tables to populate the database later. The tables (relations) in the system are;

1. Employee table
2. Department
3. Positions
4. AccidentTypes
5. Involved_in
6. Received (receivetest)
7. Complaint codes (c_codes)

Following are the functional dependencies of each table and requirements for each attribute.

1. *Employees*

- employee_id determines first_name, last_name, salary_category, promotions, projects, start_year, salary,position_code and department_code.
- Employee_id is the primary key and department_code and position code are forign keys which references department_code from departments table and position_code form positions table respectively.
- Promotions and projects can be null and all others should not be null.

2. *Departments*

- department_code determines department_name, address, phone.
- Department code is the primary key and all attributes need to be not null.

3. *Positions*

- position_code determines description and it is the primary key.
- All attributes need to be not null.

4. *AccidentTypes*

- Accident_type is the only attribute here and this table does not have any functional dependencies.
- Accident_type alone act as both primary key and foreign key. All attributes need to be not null.

5. *Involved_in*

- employee_id, accident_type and accident_id altogether determines accident_date. Therefore it act as the composite primary key.
- Employee_id and accident_id are foreign keys and employee_id references from employee_id of employee table and accident_id references from accident_id of accidents table.
- All attributes need to be not null.

6. *Received (receivetest)*

- employee_id, complain_id and complain_code altogether determines resolution_status, resolution_code and complaint_date and therefore itis the composite primary key.
- Two foreign keys employee_id and complaint_id references from employee_id of employee table and complaint_id from complaint codes (c_code) table.
- All attributes need to be not null.

7. *Complaint codes (c_codes)*

- complain_code determines description and class and it is the primary key.
- All attributes need to be not null. All attributes need to be not null.

2.2 Features implemented, and how they work

The Chicago Employee Complaint and Accident Database can be used to track issues regarding various employees' accidents and complaints. All employees have unique Employee ID (Primary key), Last Name, First Name, Employee Annual Salary, Projects, Salary Category, Promotions. Employee ID determines Employee Last Name, Employee First Name, Employee Annual Salary, Start Year, Projects, Salary Category, and Promotions.

Each employee works for a certain department and holds a certain position. Each of them has a Start Year for that they start working in certain department. Department has a unique Department Code (Primary key), a Department Name, Address, and Phone number. A position has a unique Position Code (Primary key) and Position Description.

AccidentTypes table keeps records of accident types that have happened so far, and the table can be updated with new accident types in case they happen in the future.

An Employee can involve in multiple accident types and an accident can be caused by multiple employees. An accident type is unique (Primary key) in AccidentType table. Each employee gets a unique accident id even if multiple employees happen to involve in the same accident making updating the database easier.

Complaint Codes table keeps track of complaint codes (Primary key) issued so far and if any new complaint type is created the table can be updated with its complaint code, description and complaint class.

An employee can receive multiple complaints and a certain complaint can have multiple employees involved. Different employees included in the same complaint which has a unique Complaint ID can have different Resolution Status and different Resolution Code. This makes it easier to update the resolution status and code for each employee. A complaint is associated with a unique Complaint ID (Primary key) and date. Each complaint has a description that is associated with a unique Complaint Code (Primary key), Class, and Description.

2.3 E/R diagram and relational schema implementation

E/R diagram

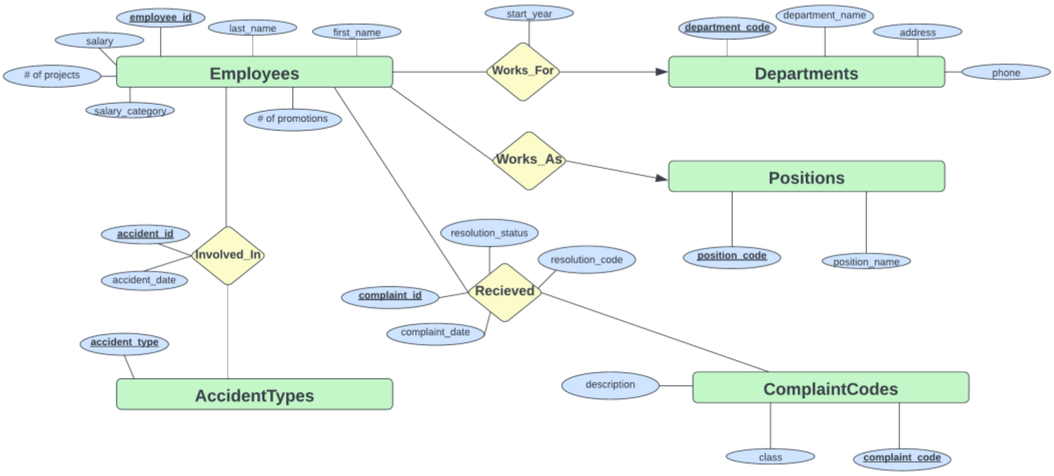


Figure 01

Relational Schema:

Employees (employee_id, first_name, last_name, salary_category, promotions, projects, start_year, salary, department_code,position_code)
department_code references department_code from Departments table
position_code references position_code from Positions table

AccidentTypes (accident_type)

Involved_In (employee_id, accident_type, accident_date, accident_id)
employee_id references employee_id from Employees table
accident_id references accident_id from AccidentTypes table

ComplaintCodes(complain_code,description, class)

Received (resolution_status, resolution_code, employee_id, complaint_id, complaint_date,complaint_code)
employee_id references employee_id from Employees table
complaint_id references complaint_id from ComplaintCodes table

Departments(department_code, department_name, address, phone)

Positions(position_code, description)

2.4 SQL queries

1. accident count by department for year 2020

```
select d.department_name, t2.accident_count
from department d
join (select e.department_code as d_code, sum(t1.count1) as accident_count
        from employees e join
                        (select i.employee_id as e_id, count(i.accident_id) as count1
                        from involved_in i
                        where extract(year from i.accident_date) = 2020
                        group by i.employee_id) as t1
        on e.employee_id = t1.e_id
        group by e.department_code) t2
on d.department_code = t2.d_code
order by t2.accident_count desc;
```

2. complaint count by department for year 2020

```
select d.department_name, t2.complaint_count
from department d
join (select e.department_code as d_code, sum(t1.count1) as complaint_count
        from employees e join
                        (select i.employee_id as e_id, count(i.complaint_id) as count1
                        from receivedtest i
                        where extract(year from i.c_date) = 2020
                        group by e_id) as t1
        on e.employee_id = t1.e_id
        group by e.department_code) as t2
on d.department_code = t2.d_code
order by t2.complaint_count desc;
```

3. list of complaint description of compliants from each departmnet in 2020 where compliant class is 'COVID'

```
select dep.department_name, string_agg(des.complaint_description, ',')
from (select  d.department_name, r.complaint_id as ff
    from department d
    join employees e ON e.department_code = d.department_code
    join receivedtest r on e.employee_id = r.employee_id ) dep
join (select cc.complaint_description , r.complaint_id as ff
    from receivedtest r
    join c_codes cc on r.complaint_code = cc.complaint_code
    where extract(year from r.c_date) = 2020 AND  cc.complaint_class = 'COVID') des
on dep.ff = des.ff
group by dep.department_name;
```

4. Concatenated(first name and last name) of employees who works for fire department who has received a complaint class as 'SexOffense' in 2020 with the resolved status and complaint description, order alphabetically.

```
select t2.employee_name, t2.complaint_description, t2.resolution_status, t2.department_code
from (select CONCAT(e.first_name, ' ', e.last_name) employee_name ,t1.complaint_description,
t1.resolution_status, e.department_code
        from (select r.employee_id, r.resolution_status, cc.complaint_description
            from receivedtest r
            join c_codes cc on r.complaint_code = cc.complaint_code
            where cc.complaint_class = 'SexOffense' and (extract(year from r.c_date)) = 2020)
t1
        join employees e on t1.employee_id = e.employee_id
        order by employee_name) t2
join departments d on t2.department_code = d.department_code
where department_name ='FIRE';
```

5. Number of complaint_ids issued for each employee , ordered by number of complaints. output full name, employee_id, department name along with complaint_id count.

```
select t2.employee_id, t2.employee_name, d.department_name, t2.complaint_count
from (select t1.employee_id, CONCAT(e.first_name, ' ', e.last_name) as employee_name,
e.department_code ,t1.complaint_count
        from (select r.employee_id, count(*) as complaint_count
                from receivedtest r
                group by r.employee_id
                order by count(*) desc) t1
        join employees e on t1.employee_id = e.employee_id) t2
join departments d on t2.department_code = d.department_code;
```

6. Accident count by month ordered by month with highest count to lowest count.

```
select t1.month, t1.count
from (select extract(month from accident_date) as month, count(*) as count
        from involved_in
        group by month
        order by count desc) t1
```

7. Employees who get salary more than 70000 and recieved more than 5 complaint-ids. Output employee_id, their position, department that they are working for and complaint count.

```
select t3.employee_id, p.position_name ,t3.department_name, t3.complaint_count
from (select t2.employee_id, t2.complaint_count, t2.position_code, d.department_name
        from (select t1.employee_id, t1.complaint_count, e.position_code, e.department_code
                from (select r.employee_id, count(*) as complaint_count
                        from receivedtest r
                        group by r.employee_id
                        having count(*)>5) t1
                join employees e on t1.employee_id = e.employee_id
                where e.salary >70000) t2
        join departments d on t2.department_code = d.department_code) t3
join positions p on t3.position_code = p.position_code;
```

8. Department with highest number of accidents

```
select departments.department_name, witness.ct from departments
join (select d.department_name ,count(i.accident_id) as ct
      from Involved_in  i
      join employees e on e.employee_id = i.employee_id
      join departments d on e.department_code = d.department_code
      group by d.department_code
      having count(i.accident_id) >= ALL(select count(i.accident_id) as count from Involved_in
i
                                         join employees e on e.employee_id = i.employee_id
                                         join departments d on e.department_code =
d.department_code
                                         group by d.department_code)) witness
on witness.department_name = departments.department_name;
```

9. Department with highest number of complaints

```
select departments.department_name, witness.complaint_count from departments
join (select d.department_name ,count(r.complaint_id) as complaint_count
      from receivedtest  r
      join employees e on e.employee_id = r.employee_id
      join departments d on e.department_code = d.department_code
      group by d.department_code
      having count(r.complaint_id) >= ALL(select count(r.complaint_id) as count from
receivedtest  r
                                         join employees e on e.employee_id = r.employee_id
                                         join departments d on e.department_code =
d.department_code
                                         group by d.department_code)) witness
on witness.department_name = departments.department_name;
```

10. Number of employees in each department in descending order

```
select d.department_name , count(e.employee_id) as Employee_Count
from employee1 e
inner join department d
ON e.department_code = d.department_code
group by d.department_name
Order by employee_Count desc;
```

11. Promotion count for employees (with their full name) in descending order with their accident count and complaint count.

```
SELECT CONCAT(e.First_Name, ' ' ,e.Last_Name) as Full_Name , e.promotions as promotions_count
, count(i.accident_Type) accident_Count, count(r.complaint_code) complaint_Count
from employees e
LEFT JOIN Involved_in i
ON e.employee_id = i.employee_id
LEFT JOIN receivedtest r
ON e.employee_id = r.employee_id
where e.promotions > 0
Group by Full_Name , promotions_count
order by Full_Name desc;
```

12. Average salary, number of employees for each salary category

```
SELECT salary_category ,
AVG (salary) as Avg_Salary ,
count(employee_id) as Employee_Count
```

```
From employees
Group By salary_category;
```

13. Number of pending, resolved and non-resolved complaint count for each department.

```
select o.department_name,d.resolution_status,count(e.*)
from (select distinct d.department_name
        from departments d
        join employees e on d.department_code = e.department_code
        join receivedtest r on r.employee_id = e.employee_id) o
        cross join
        (select distinct r.resolution_status
        from departments d
        join employees e on d.department_code = e.department_code
        join receivedtest r on r.employee_id = e.employee_id) d
        left join
        (select *
        from departments d
        join employees e on d.department_code = e.department_code
        join receivedtest r on r.employee_id = e.employee_id) e
on e.department_name = o.department_name and e.resolution_status = d.resolution_status
group by o.department_name,d.resolution_status
order by o.department_name,d.resolution_status;
```

14. Top 3 complainees in each department

```
select t3.department_name,string_agg(t3.employee_name, ',') as top_3_complainees
from (select t2.department_name,
            t2.employee_name,
            t2.sum,
            row_number() over (partition by t2.department_name order by t2.sum desc) as rank
        from (select t1.employee_name,t1.department_name, sum(count) as sum
                from (select CONCAT(e.first_name, ' ', e.last_name) employee_name ,
d.department_name, count(*)
                        from receivedtest r
                        join  employees e on r.employee_id = e.employee_id
                        join departments d on d.department_code = e.department_code
                        group by employee_name,d.department_name) as t1
                group by t1.employee_name,t1.department_name
           ) as t2
    ) as t3
where rank <= 3
group by t3.department_name
```

15. Employee who has received highest number of distinct complaint codes

```
select CONCAT(e.first_name, ' ', e.last_name) as employee_name, witness.ct from employees e
join (select e.employee_id ,count(distinct c.complaint_code) as ct
      from employees  e
      join receivedtest r on e.employee_id = r.employee_id
      join c_codes c on r.complaint_code = c.complaint_code
      group by e.employee_id
      having count(c.complaint_code) >= ALL(select count(distinct c.complaint_code) as count
from employees  e
                                            join receivedtest r on e.employee_id =
r.employee_id
                                            join c_codes c on r.complaint_code =
c.complaint_code
                                            group by e.employee_id)) witness
on witness.employee_id = e.employee_id;
```

16. Employees working for 'FAMILY & SUPPORT' department for more than 30 years, with their promotion count, duration

```
select t1.employee_name,t1.promotions,t1.duration
from (
    select CONCAT(e.first_name, ' ', e.last_name) as employee_name, e.promotions,
d.department_name, date_part('year',current_date)-e.start_year as duration
    from employees e join departments d
    on e.department_code = d.department_code ) as t1
where t1.duration >30 and t1.department_name = 'FAMILY & SUPPORT'
order by t1.promotions desc;
```

17. All the positions offered from Department of IT (DoIT) with head count for each position

```
select p.position_name,  count(*) as number_of_employees
from employees e
join departments d on e.department_code =d.department_code
join positions p on p.position_code = e.position_code
where d.department_name ='DoIT'
group by p.position_name
order by number_of_employees desc;
```

3. Usage of NoSQL

3.1 Specific technology used and justification

As the NoSQL component we used a Graph Database : Neo4j. Like other graph databases, neo4j is built for use with transactional (OLTP) systems and are engineered with transactional integrity and operational availability in mind. Neo4j Graph Database stores all of its data in Nodes and Relationships. We don’t need any additional RRBMS Database or any SQL database to store Neo4j database data. It stores its data in terms of Graphs in its native format.

All Create, Read, Update and Delete (CRUD) operations work on a neo4j graph data model. It is significantly simpler and more expressive than those of relational or other NoSQL databases. Relationships take first priority in graph databases. This means your application doesn’t have to infer data connections using things like foreign keys or out-of-band processing, such as MapReduce.

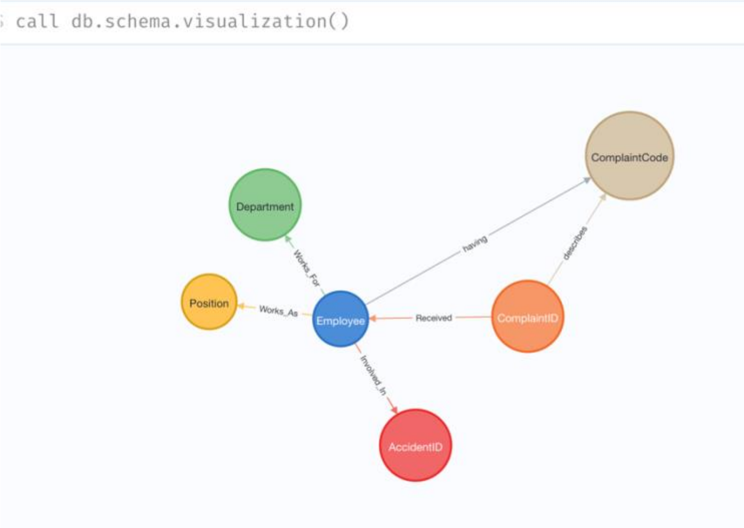
The main reason for selecting neo4j as our NoSQL component is its simplicity. Once the model is made, we can see how the employees are connected to different entities. Also, the CRUD operations are easy to implement in Neo4j using its own query language Cypher. Cypher queries are much simpler than SQL as well.

3.2 Data model

The data model in Neo4j, data is represented as nodes and relationship between them are known as edges. Both nodes and relationships can have properties, which store the data items associated with nodes and relationships. A node can have zero or multiple labels. The nodes that have the same label are grouped into a collection that identifies a subset of the nodes in the database graph for querying purposes.

Relationships are directed, each relationship has a start node and end node as well as a relationship type, which serves a similar role to a node label by identifying similar relationships that have the same relationship type Properties can be specified via a map pattern, which is made of one or more “name : value” pairs enclosed in curly brackets.

Figure 02 shows the schema of our graph model. Our model has Employee nodes, Department nodes, Position nodes, ComplaintID nodes, AccidentID nides and ComplaintCode nodes. Employee nodes are connected to Department nodes by ‘Works_For’ edge. Employees and Position nodes are connected with ‘Works_As’ relationship. Every employee who cause an accident gets unique accident id and once that happens Employee nodes connect with AccidentID nodes via ‘Involved_In’ edge. When an employee get a complaint against them, ComplaintID nodes connects with Employee nodes via ‘Received’ edge. ComplaintCode nodes and ‘ComplaintID’ nodes are connected by ‘describes’ edge and this represent a description of complaint codes. Finally, Employee nodes and ComplaintCode nodes are connected via ‘having’ relationship.



```
call db.schema.visualization()
```

Figure 02

3.3 Features implemented and how they work

Following are the nodes in our graph model and their CONSTRAINTS and properties.

Nodes:

Employee

Constraint on: employee_id
Properties: employee_id last_name, first_name, department_code, position_code, salary, start_year,salary_category, projects,promotions,

Department

Constraint on: department_code
Properties: department_code ,department_name, address , phone

Position

Constraint on: position_code
Properties: position_code , position_name

AccidentID

Constraint on: accident_id
Properties: accident_id

ComplaintID

Constraint on: complaint_id
Properties: complaint_id, complaint_date

ComplaintCode

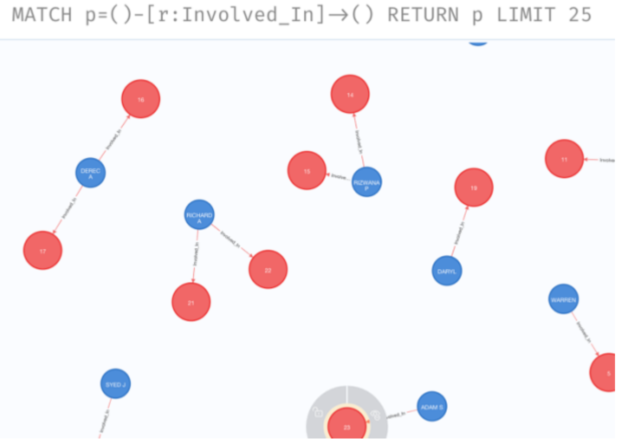
Constraint on: complaint_code
Properties: complaint_code, description, class

Following are the relationships in our graph model and their properties.

Edges (Relationships)

[Involved_In]

Properties : accident_type, date_happen

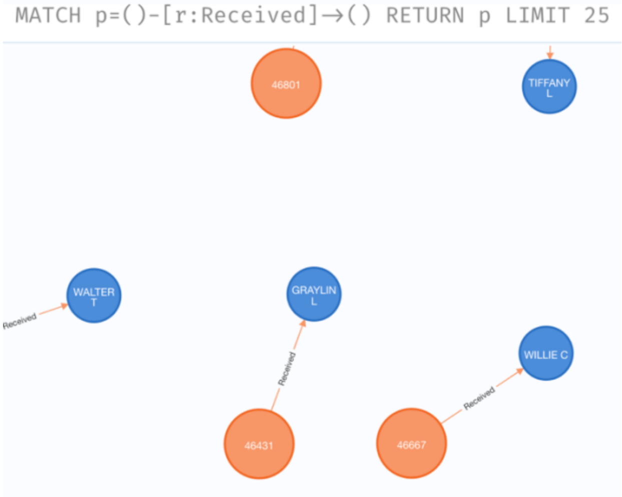


```
MATCH p=()-[r:Involved_In]→() RETURN p LIMIT 25
```

Figure 03

[Received]

Properties: resolution_code, resolution_status

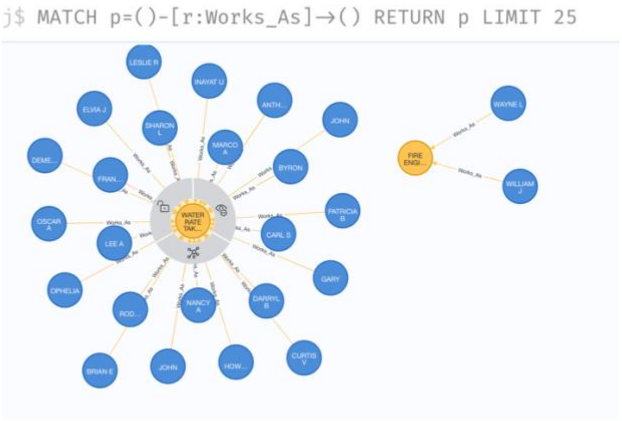


```
MATCH p=()-[r:Received]→() RETURN p LIMIT 25
```

Figure 04

[Works_As]

Properties: none

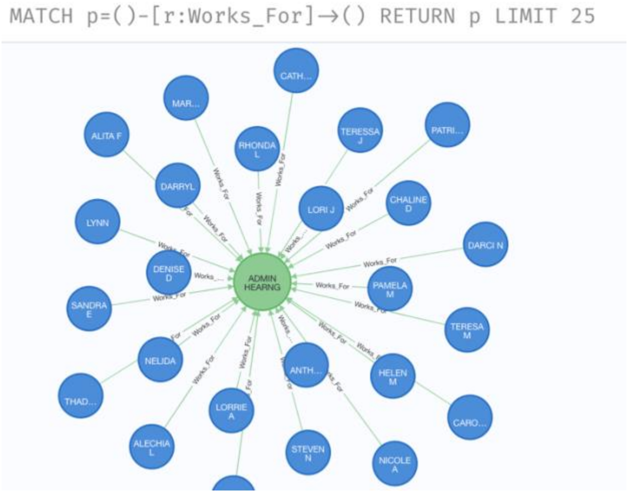


```
j$ MATCH p=()-[r:Works_As]→() RETURN p LIMIT 25
```

Figure 04

[Works_For]

Properties: none

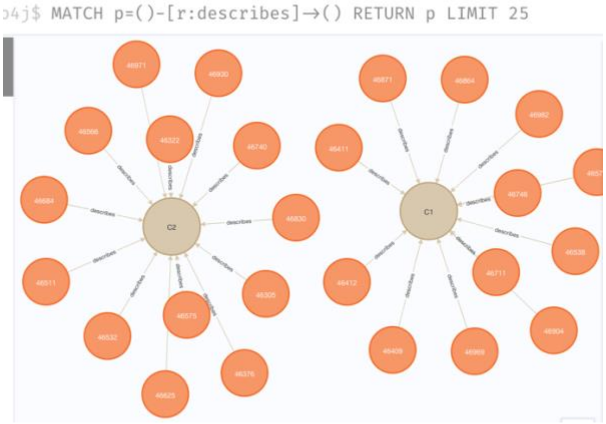


```
MATCH p=()-[r:Works_For]→() RETURN p LIMIT 25
```

Figure 05

[describes]

Properties: none

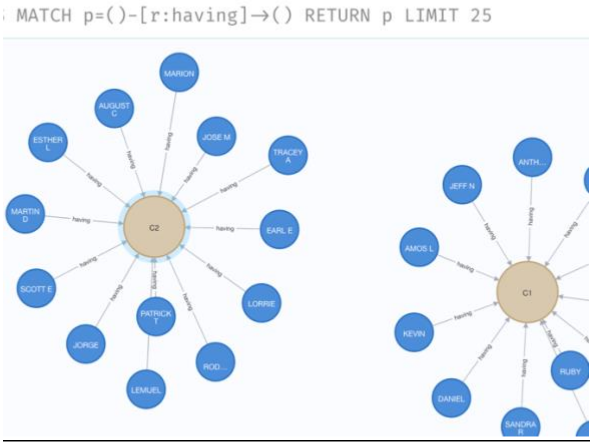


```
o4j$ MATCH p=()-[r:describes]→() RETURN p LIMIT 25
```

Figure 06

[having]

Properties: none



```
MATCH p=()-[r:having]→() RETURN p LIMIT 25
```

Figure 07

3.4 Queries answered in NoSQL

1. Accident count by department for year 2020 (SQL Query 1)

```
MATCH p=(d:Department)-[:Works_For]-(e:Employee)-[i:Involved_In]-(a:AccidentID)
WHERE i.date_happen.year=2020
RETURN d.department_name AS dep,
count(i) AS no_of_accidents
ORDER BY no_of_accidents DESC;
```

2. Complaint count by department for year 2020 (SQL Query 2)

```
MATCH p=(d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID)
WHERE c.complaint_date.year=2020
RETURN d.department_name AS dep,
count(r) AS no_of_complaints
ORDER BY no_of_complaints DESC;
```

3. list of complaint description of complaints from each department in 2020 where compliant class is 'COVID' (SQL Query 3)

```
MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID)-
[f:describes]-(cc:ComplaintCode)
WHERE c.complaint_date.year=2020 AND cc.class='COVID'
RETURN d.department_name,[x IN COLLECT(cc) | x.description],[y IN COLLECT(e) |
y.employee_id];
```

4. Concatenated (first name and last name) of employees who works for fire department who has received a complaint class as 'SexOffense' in 2020 with the resolved status and complaint description, order alphabetically. (SQL Query 4)

```
MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID)-
[f:describes]-(cc:ComplaintCode)
WHERE c.complaint_date.year=2020 AND  d.department_name ='FIRE' AND cc.class
='SexOffense'
MATCH  (d:Department)-[:Works_For]-(e:Employee)-[h:having]-(cc:ComplaintCode)
RETURN c.complaint_id,COALESCE(e.first_name ,"") + ' ' + COALESCE(e.last_name ,"")
AS employee_name,
cc.description,r.resolution_status
ORDER BY employee_name;
```

5. Number of complaint_ids issued for each employee , ordered by number of complaints. output full name, employee_id, department name along with complaint_id count. (SQL Query 5)

```
MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID)
RETURN e.employee_id,COALESCE(e.first_name ,"") + ' ' + COALESCE(e.last_name ,"")
AS employee_name,d.department_name, count (c:ComplaintID) as number_of_complaints
ORDER BY number_of_complaints DESC;
```

6. Accident count by month ordered by month with highest count to lowest count. (SQL Query 6)

```
MATCH p=(d:Department)-[:Works_For]-(e:Employee)-[i:Involved_In]-(a:AccidentID)
RETURN i.date_happen.month AS month ,COUNT(i.date_happen.month) AS count
ORDER BY (count) desc;
```

7. Employees who get salary more than 70000 and recieved more than 5 complaint-ids. Output employee_id, their position, department that they are working for and complaint count. (SQL Query 7)

```
MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID),
(e:Employee)-[:Works_As]-(p:Position)
With count(c:ComplaintID) as x,d,e,p
where e.salary >70000 and x>5
RETURN e.employee_id,p.position_name,d.department_name,x;
```

8. Department with highest number of accidents(SQL Query 8)

```
MATCH (d:Department)-[:Works_For]-(e:Employee)-[i:Involved_In]-(a:AccidentID)
With d,COUNT(a:AccidentID) as accident_count
ORDER BY accident_count DESC
RETURN collect(d.department_name)[0] as dep,max(accident_count) as
accident_prone_department;
```

9. Department with highest number of complaints(SQL Query 9)

```
MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID)
WITH d,COUNT(c:ComplaintID) as complaint_count
ORDER BY complaint_count DESC
RETURN collect(d.department_name)[0] as dep,max(complaint_count) as
complaint_prone_department;
```

10. Number of employees in each department in descending order. (SQL Query 10)

```
MATCH (d:Department)-[:Works_For]-(e:Employee)
WITH d,COUNT(e:Employee) as employee_count
ORDER BY employee_count desc
RETURN d.department_name,employee_count;
```

11. Employee who has received highest number of distinct complaint codes (SQL Query 15)

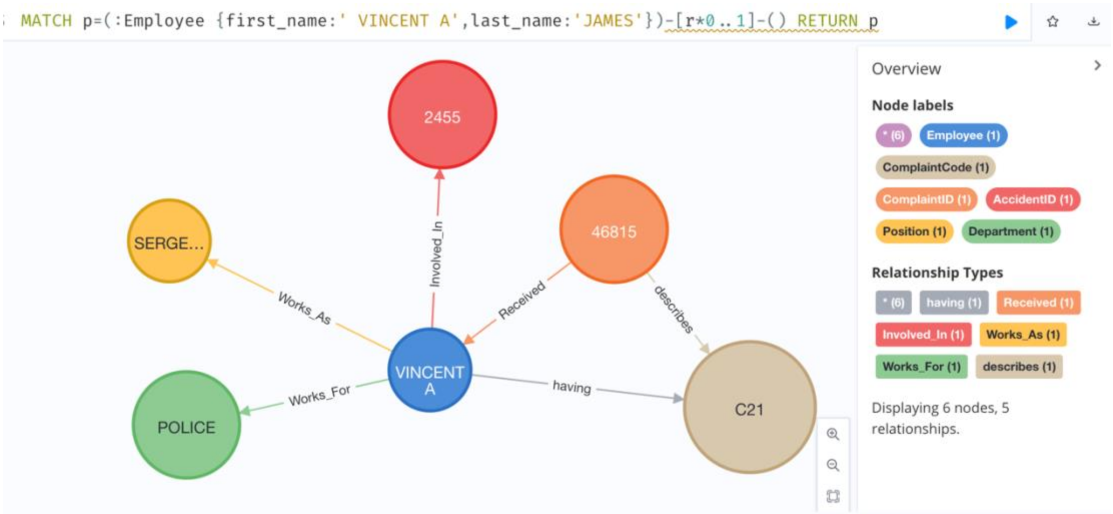
```
MATCH (d:Department)-[:Works_For]-(e:Employee)-[h:having]-(cc:ComplaintCode)
WITH COALESCE(e.first_name ,"") + ' ' + COALESCE(e.last_name ,"") AS employee_name,
COUNT(DISTINCT(cc.complaint_code)) AS distinct_complaint_codes
ORDER BY distinct_complaint_codes DESC
```

```
RETURN collect(employee_name)[0] as employee,max(distinct_complaint_codes) as
complaint_prone_department;
```

4. SQL and NoSQL to complement each other

We choose Neo4j graph database as our NoSQL component. Being a graph database, Neo4j can visualize how the entities are connected with each other which is not an option in traditional database management systems.

As an example, we can get all the details visualize for a certain employee with a single line of code as below. (Figure 08)This helps user to easily understand and spot the relations between entities. This option is not available in RDBMS systems and if we need to get that information, we have to write long lines of code but, visualization is not an option.

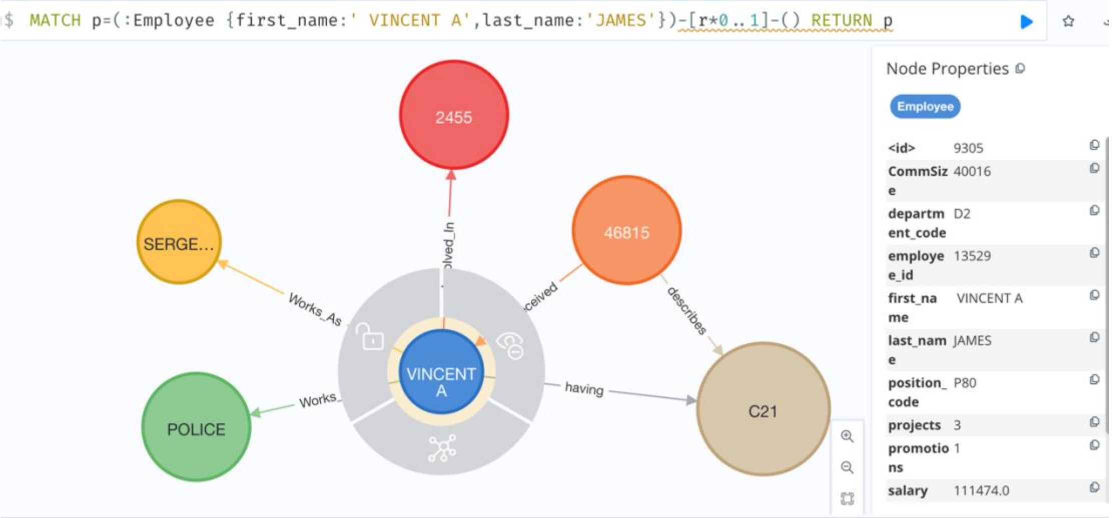


```
MATCH p=(:Employee {first_name:' VINCENT A',last_name:'JAMES'})-[r*0..1]-() RETURN p
```

Displaying 6 nodes, 5 relationships.

Figure 08

Taking the curser to an entity lets you see all the properties for that entity as well. (Figure 09)



```
$ MATCH p=(:Employee {first_name:' VINCENT A',last_name:'JAMES'})-[r*0..1]-() RETURN p
```

Node Properties

Employee

<id>	9305
CommSize	40016
department_code	D2
employee_id	13529
first_name	VINCENT A
last_name	JAMES
position_code	P80
projects	3
promotions	1
salary	111474.0

Figure 09

5. SQL and NoSQL to achieve the same functionality

Both SQL and NoSQL can be used to get answers to queries. Our NoSQL component ; Neo4j uses it is own Cypher for querying.

However, Cypher queries are much simpler to implement compared to SQL queries. We have implemented query 1 through 10 in both SQL and Cypher. As you can see in section 3.4 cypher queries are much shorter compared to SQL queries. As an example, Cypher query to find the most accident-prone department (Section 3.4 Query # 8), which is a witness type query, can be implemented using Cypher in 4 lines of code whereas it needs more than 10 lines of code to implement using PostgreSQL (Section 2.4 Query #8).

Moreover, neo4j is much faster than RDBMS for querying. We recorded execution time for first 10 queries in neo4j and PostgreSQL (Table 01). Please see the appendix for screenshots of queries in PostgreSQL and Neo4j.

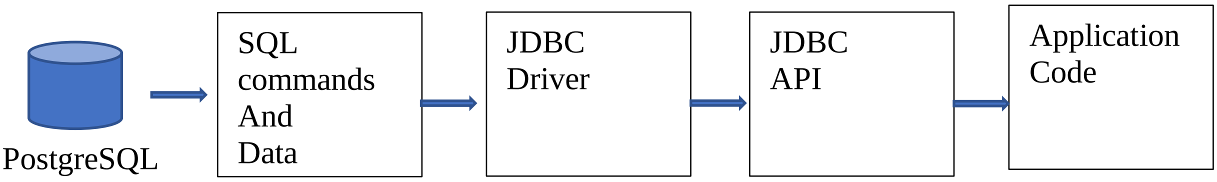
Query No:	Execution time in RDBMS using postgresql (ms)	Execution time in neo4j using cypher (ms)
1	11.08	9
2	14.19	4
3	7.46	3
4	6.41	2
5	65.68	17
6	7.46	17
7	23.03	12
8	40.79	16
9	29.42	16
10	29.43	21

Just like SQL supports CRUD operations, we can implement them in Neo4j as well.

6. System implementation:

(show diagrams that can help you explain the implementation of the project. These diagrams should show software/code components and their relationship (i.e., system architecture). Include everything that can clarify how your application is composed, how the tasks and data are processed, how the software components are organized to work together. You may want to use specific tasks to explain the data flow and steps involved in handling the task.)

We used JDBC (Java Database Connectivity); the Java API to connect to the database and then issue queries and commands, and handle result sets obtained from the database.

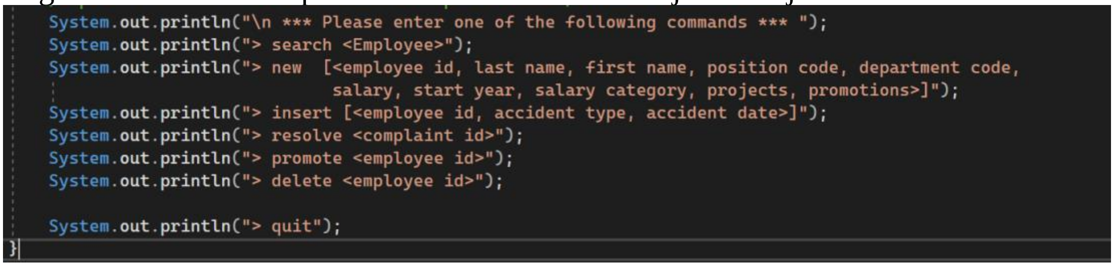


The steps for connecting to PostgreSQL database with JDBC are as follows:

1. locate the database.
2. Include the JDBC library.
3. Ensure the JDBC driver is in the classpath.
4. Use the JDBC library to obtain a connection to the database.
5. Use the connection to issue SQL commands.

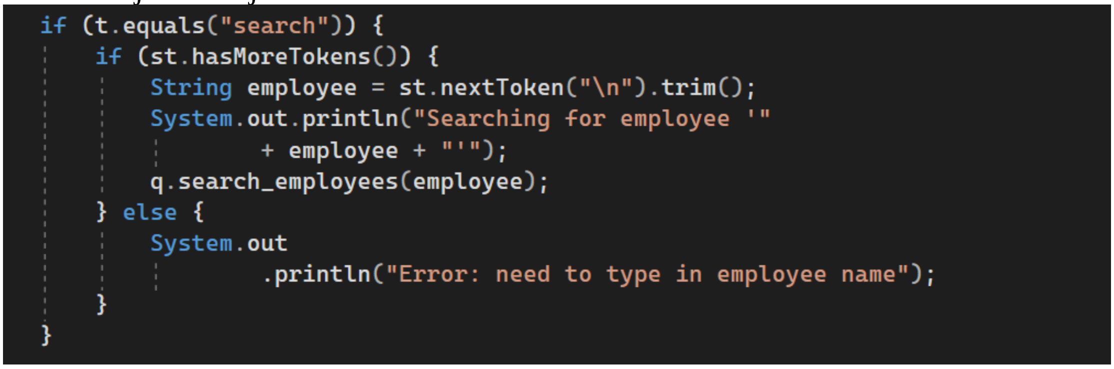
Below screenshots show the code for interface to searching an employee.

Logic that creates the options for the user on the ProjectStart.java



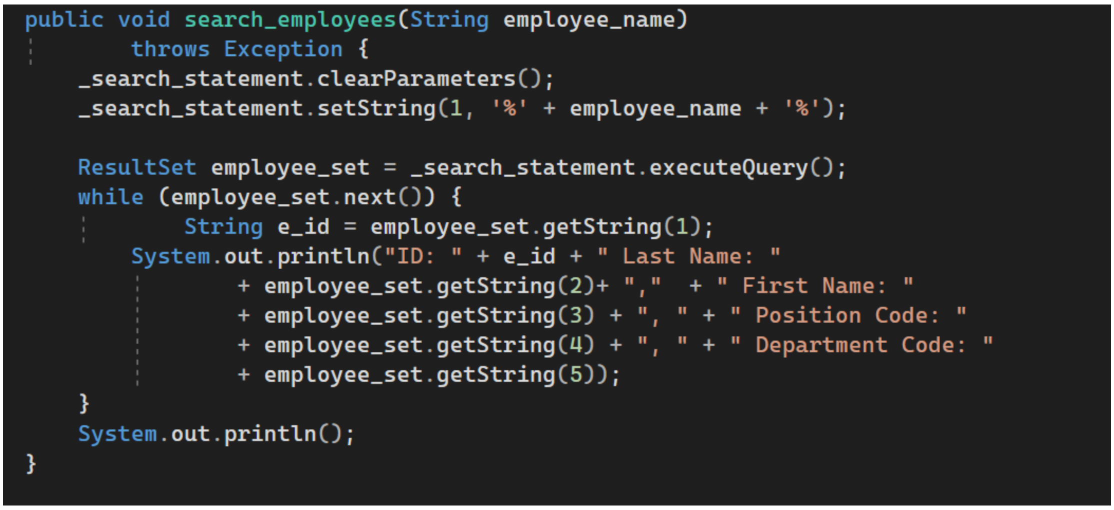
```
    System.out.println("\n *** Please enter one of the following commands *** ");
    System.out.println("> search <Employee>");
    System.out.println("> new  [<employee id, last name, first name, position code, department code,
                                salary, start year, salary category, projects, promotions>]");
    System.out.println("> insert [<employee id, accident type, accident date>]");
    System.out.println("> resolve <complaint id>");
    System.out.println("> promote <employee id>");
    System.out.println("> delete <employee id>");

    System.out.println("> quit");
}
```

Logic that puts the user in the block of code that will call the search function. This is on the ProjectStart.java



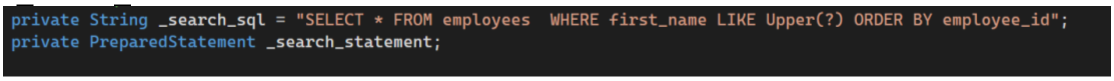
```
if (t.equals("search")) {
    if (st.hasMoreTokens()) {
        String employee = st.nextToken("\n").trim();
        System.out.println("Searching for employee '"
                + employee + "'");
        q.search_employees(employee);
    } else {
        System.out
                .println("Error: need to type in employee name");
    }
}
```

This is the block of code that will run the query that will return all users with anything like the users input “Kevin” for example.



```
public void search_employees(String employee_name)
        throws Exception {
    _search_statement.clearParameters();
    _search_statement.setString(1, '%' + employee_name + '%');

    ResultSet employee_set = _search_statement.executeQuery();
    while (employee_set.next()) {
            String e_id = employee_set.getString(1);
        System.out.println("ID: " + e_id + " Last Name: "
                + employee_set.getString(2)+ ","  + " First Name: "
                + employee_set.getString(3) + ", " + " Position Code: "
                + employee_set.getString(4) + ", " + " Department Code: "
                + employee_set.getString(5));
    }
    System.out.println();
}
```

This shows the query that will run once the previous block of code gets to "_search_statement"

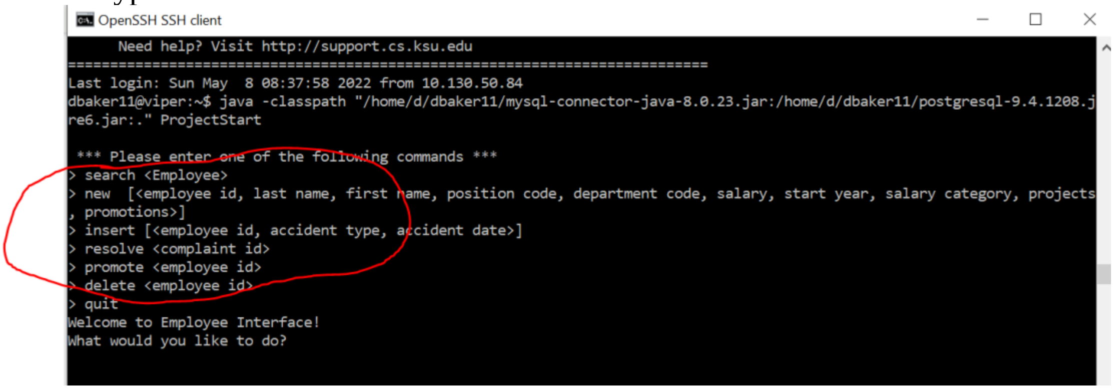


```
private String _search_sql = "SELECT * FROM employees  WHERE first_name LIKE Upper(?) ORDER BY employee_id";
private PreparedStatement _search_statement;
```

Once this block of code is running it will bring back all data with the name "Kevin" in it. If it comes back blank, then we have no data like "Kevin". To make it easier on the user we account for capitalization so the user can input "kevin" , "Kevin", "KEVIN" it will find all the data associated with that first name.

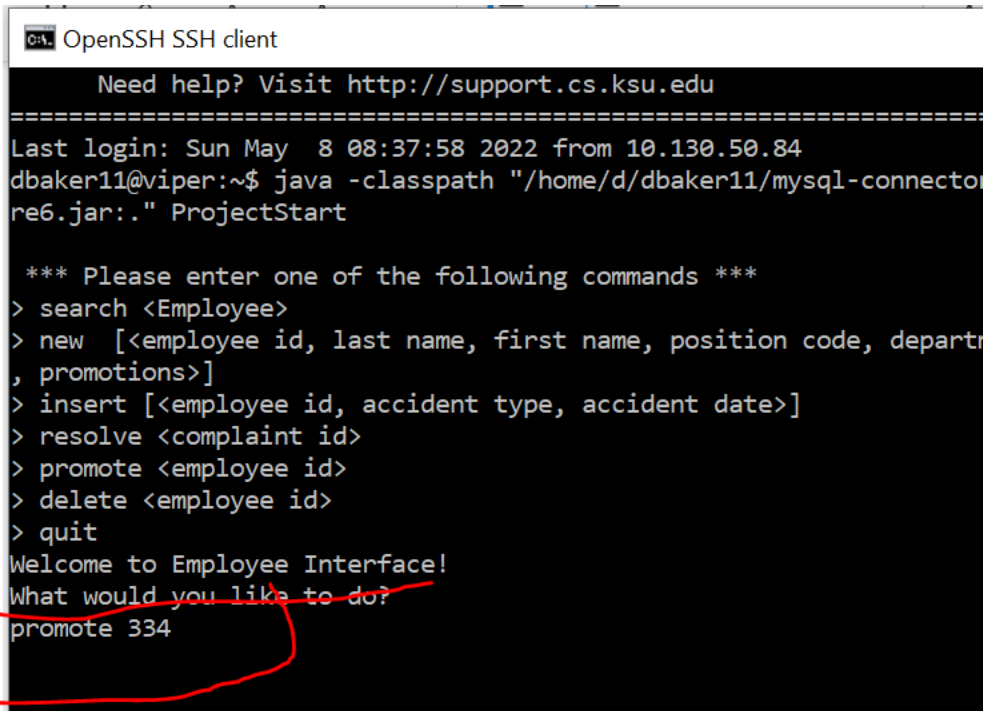
7. System features and usage - use example scenarios (and screenshots) to explain the system. Show how users interact with the system. Highlight the strengths and limitations of your system.

When the user opens the interface, it'll give the options to insert/update/delete various types of records like the screenshot below shows.



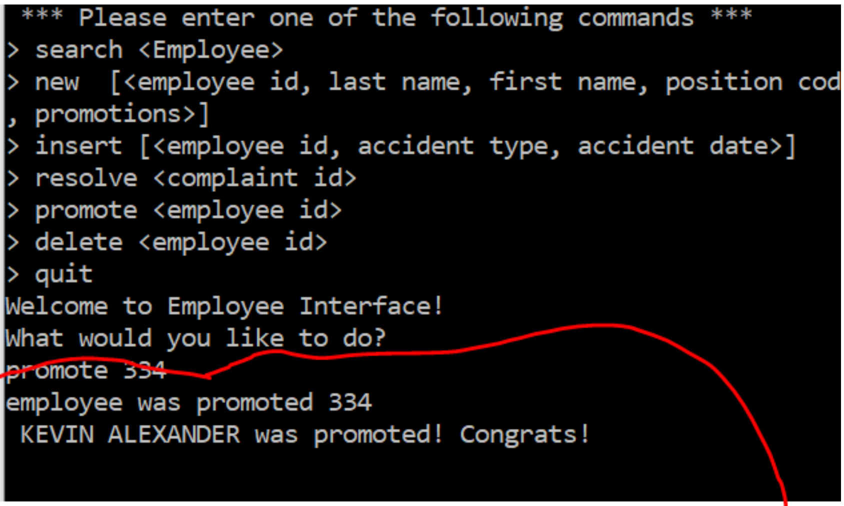
```
      Need help? Visit http://support.cs.ksu.edu
===========================================================================
Last login: Sun May  8 08:37:58 2022 from 10.130.50.84
dbaker11@viper:~$ java -classpath "/home/d/dbaker11/mysql-connector-java-8.0.23.jar:/home/d/dbaker11/postgresql-9.4.1208.j
re6.jar:." ProjectStart

 *** Please enter one of the following commands ***
> search <Employee>
> new  [<employee id, last name, first name, position code, department code, salary, start year, salary category, projects
, promotions>]
> insert [<employee id, accident type, accident date>]
> resolve <complaint id>
> promote <employee id>
> delete <employee id>
> quit
Welcome to Employee Interface!
What would you like to do?
```

The user then inputs what they want to do like the screen shot below "promoting" a user.



```
      Need help? Visit http://support.cs.ksu.edu
=====================================================================
Last login: Sun May  8 08:37:58 2022 from 10.130.50.84
dbaker11@viper:~$ java -classpath "/home/d/dbaker11/mysql-connecto
re6.jar:." ProjectStart

 *** Please enter one of the following commands ***
> search <Employee>
> new  [<employee id, last name, first name, position code, depart
, promotions>]
> insert [<employee id, accident type, accident date>]
> resolve <complaint id>
> promote <employee id>
> delete <employee id>
> quit
Welcome to Employee Interface!
What would you like to do?
promote 334
```

When the user presses enter, if it is valid, it will say so if an invalid input was submitted it will also let the user know that they have inserted a invalid employee id. The screenshot below shows that the user was promoted successfully and updated the users record in the database.



```
 *** Please enter one of the following commands ***
> search <Employee>
> new  [<employee id, last name, first name, position cod
, promotions>]
> insert [<employee id, accident type, accident date>]
> resolve <complaint id>
> promote <employee id>
> delete <employee id>
> quit
Welcome to Employee Interface!
What would you like to do?
promote 334
employee was promoted 334
 KEVIN ALEXANDER was promoted! Congrats!
```

The strength of our system is that it is easy to follow, easy to understand, as well as easy to insert/update records. It is also very face interacting with the database, virtually no latency. Limitations of the current interface may make it easy to lose focus on what you are doing if you have updated many records. A web front end with a record history of what has been changed could be an improvement for a later development.

8. Evaluation – explain how you have evaluated your system (specifically, performance and easiness to use). Did you need to use indexes in SQL? If so, explain what indexes you created and show how they were useful. What about NoSQL? Did you use a technology that is indexing the data? If so, what indexes

were created?

This system runs great, with no latency and most of the queries do run within seconds to gather data or update records depending on the option of choice. The sytem overall is easy to use. It is straight forward of what the user needs to input to get the correct result they are looking for. If the user inputs invalid data, the error handling will let the user know to fix it.

For this specific interface NoSQL was involved due to time and knowledge of linking it to an interface. Indexes were also not involved with this interface because the data already came back very face. With it coming back very face with no issue to the data, we felt it would not be necessary to input indexes on our tables.

9. Technical details about other technologies used in your project and a short justification for your choice.

Java was used to implement a simple command line interface. We have a ProjectStart.Java and a ProjectQuery.Java to run our interface. The ProjectStart.Java has coding logic that displays functionality options to the user on the screen. The ProjectQury.Java has all the logic for our queries which also inserts/updates/deletes depending on the selection the user made prior.

The interface is simple enough for a user to understand what to do on the interface. We chose this implementation because it was something we were familiar with. If more time allowed us, we would have created a web interface.

10. Summary and discussion

10.1 Summary

- Implemented a simple java interface connecting a PostgreSQL database using JDBC.
- Used PostgreSQL to create tables and insert employee data into those tables.
- Implemented queries to insert, update, delete various records in our database.
- Used the interface java to connect to the database and manipulate the data we see fit.
- Implemented neo4j graph database as the NoSQL component.

10.2 Discussion

We learned how to create tables and create various store procedures to insert or update our tables. Along with how to connect the database to a specific interface so a user can run the queries and do various things with the data. The thing we would change would be the style of interface. Preferably a simpler back end or a web front end. It would be a better visual for the users when t hey must use it. Also, would separate specific entities that we had and join them to other tables, but we just did not have that relevant data at the time.

10.3 Future Development

Future development would involve

- Joined search to give user all information about an employee
- Insert New department
- Insert new positions
- Delete positions/ departments
- Update interface to be a web front end.

- Explore more non sql database

11. Team work division and teamwork experience.

Team work division

Douglas Baker : Equally contributed to PostgreSQL database design, queries, report writing (6-10) and presentation. Interface design
Krish Weragalaarachchi : Equally contributed to PostgreSQL database design, queries, report writing (1-5) and presentation. NoSQL component (Neo4j)

Teamwork experience

We worked great as a team. We communicated very well together. The main tool for communication were GroupMe and emails. We knew each other's strengths when it came to specific tasks in the project. We did not hesitate to ask each other for help or ask for opinions on various queries or table structure. Both responded in timely manner and respect whenever the other needs help. The workload was not one-sided both contributed equally. We shared the workload and made sure we got our tasks done in a timely manner.

Appendix:

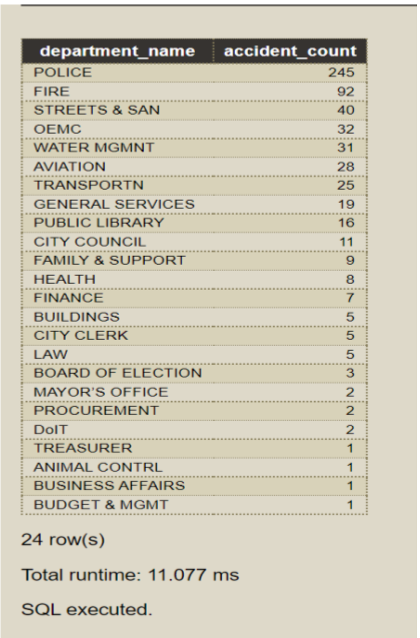
Neo4j graph database design

```
-- create constraints for all nodes
CREATE CONSTRAINT ON (d:Department) ASSERT d.department_code is UNIQUE;
CREATE CONSTRAINT ON (p:Position) ASSERT p.position_code is UNIQUE;
CREATE CONSTRAINT ON (e:Employee) ASSERT e.employee_id is UNIQUE;
CREATE CONSTRAINT ON (a:AccidentID) ASSERT a.accident_id is UNIQUE;
CREATE CONSTRAINT ON (c:ComplaintID) ASSERT c.complaint_id is UNIQUE;
CREATE CONSTRAINT ON (cc:ComplaintCode) ASSERT cc.complaint_code is UNIQUE;

/* create Department nodes and insert properties */
LOAD CSV WITH HEADERS FROM 'file:///departments.csv' AS r1
WITH r1
MERGE (d:Department {department_code: r1.`Department_Code`})
ON CREATE SET d.department_name = r1.`Department_Name`,d.address = r1.`Address`,d.phone =
r1.`Dept._Phone`;

/* create positions nodes and insert properties*/
LOAD CSV WITH HEADERS FROM 'file:///PositionCodes.csv' AS r2
WITH r2
MERGE (p:Position {position_code: r2.`Position_Code`})
ON CREATE SET p.position_name = r2.`Position_Description`;

/* create employees nodes and insert properties*/
LOAD CSV WITH HEADERS FROM 'file:///WindyCityEmployees.csv' AS r3
WITH r3
MERGE (e:Employee {employee_id: toInteger(r3.`Employee_ID`)})
ON CREATE SET e.last_name =r3.`Last_Name`,e.first_name = r3.`First_Name`,
e.department_code = r3.`Department_Code`,e.position_code = r3.`Position_Code`,
e.salary = toFloat(r3.`Employee_Annual_Salary`),
e.start_year = toInteger(r3.`Start_Year`);

/* create employee - dep employee - position relationships */
LOAD CSV WITH HEADERS FROM 'file:///WindyCityEmployees.csv' AS r3
WITH r3
MERGE (e:Employee {employee_id: toInteger(r3.`Employee_ID`)})
MERGE(d:Department {department_code:r3.`Department_Code`})
MERGE (p:Position {position_code: r3.`Position_Code`})
MERGE (e)-[:Works_For]->(d)
MERGE (e)-[:Works_As]->(p);

/* adding more properties to employee nodes */
LOAD CSV WITH HEADERS FROM 'file:///HR_Data_and_AccidentCodes.csv' AS r4
WITH r4
```

```
MERGE (e:Employee {employee_id: toInteger(r4.`Employee_ID`)})
ON CREATE SET e.salary_category =r4.`salary`, e.projects =
toInteger(r4.`number_project`),e.promotions = toInteger(r4.`promotion_last_5years`)
ON MATCH SET e.salary_category =r4.`salary`, e.projects =
toInteger(r4.`number_project`),e.promotions = toInteger(r4.`promotion_last_5years`);

/* create accidents nodes */
LOAD CSV WITH HEADERS FROM 'file:///accidents.csv' AS r5
WITH r5,SPLIT(r5.`Accident_Date`,'/') AS a_date
MERGE (a:AccidentID {accident_id: toInteger(r5.`Accident_ID`)})
MERGE (e:Employee {employee_id: toInteger(r5.`Employee_ID`)})
MERGE (e)-[i:Involved_In ]->(a)
SET i.date_happen =
date({year:toInteger(a_date[2]),month:toInteger(a_date[0]),day:toInteger(a_date[1])}),
i.accident_type = r5.`Accident_Type`;

/* create complaints code nodes */
LOAD CSV WITH HEADERS FROM 'file:///ComplaintCodes.csv' AS r7
WITH r7
MERGE (cc:ComplaintCode {complaint_code: r7.`Complaint_Code`})
ON CREATE SET cc.description = r7.`Complaint_Description`, cc.class = r7.`Complaint_Class`;

/* create complaints nodes  and relations to employees and complaint codes*/
LOAD CSV WITH HEADERS FROM 'file:///Complaint.csv' AS r6
WITH r6, SPLIT(r6.`Date`,'/') AS c_date
MERGE (c:ComplaintID {complaint_id: toInteger(r6.`ComplaintID`)})
MERGE (e:Employee {employee_id: toInteger(r6.`Employee_ID`)})
MERGE (cc:ComplaintCode {complaint_code: r6.`Complain_Code`})
SET c.complaint_date =
date({year:toInteger(c_date[2]),month:toInteger(c_date[0]),day:toInteger(c_date[1])})
MERGE (c)-[r:Received ]->(e)
SET r.resolution_status = r6.`Resolved`, r.resolution_code = r6.`Resolution_Term_Code`
MERGE (c)-[f:describes ]->(cc);


/* create complaints nodes  and relations to employees and complaint codes*/
LOAD CSV WITH HEADERS FROM 'file:///Complaint.csv' AS r6
WITH r6, SPLIT(r6.`Date`,'/') AS c_date
MERGE (c:ComplaintID {complaint_id: toInteger(r6.`ComplaintID`)})
MERGE (e:Employee {employee_id: toInteger(r6.`Employee_ID`)})
MERGE (cc:ComplaintCode {complaint_code: r6.`Complain_Code`})
SET c.complaint_date =
date({year:toInteger(c_date[2]),month:toInteger(c_date[0]),day:toInteger(c_date[1])})
MERGE (c)-[r:Received ]->(e)
SET r.resolution_status = r6.`Resolved`, r.resolution_code = r6.`Resolution_Term_Code`
MERGE (c)-[f:describes]->(cc)
MERGE (e)-[h:having]->(cc)
```

Screenshots for queries in PostgrSQL and Neo4J

1.) Accident count by department for year 2020

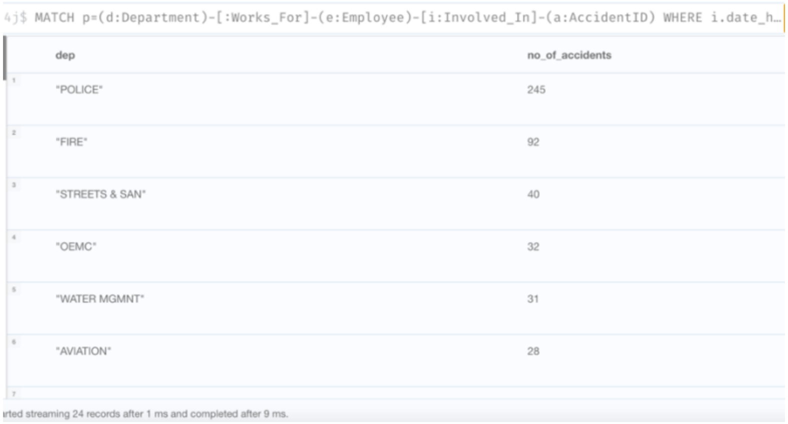


department_name	accident_count
POLICE	245
FIRE	92
STREETS & SAN	40
OEMC	32
WATER MGMNT	31
AVIATION	28
TRANSPORTN	25
GENERAL SERVICES	19
PUBLIC LIBRARY	16
CITY COUNCIL	11
FAMILY & SUPPORT	9
HEALTH	8
FINANCE	7
BUILDINGS	5
CITY CLERK	5
LAW	5
BOARD OF ELECTION	3
MAYOR'S OFFICE	2
PROCUREMENT	2
DoIT	2
TREASURER	1
ANIMAL CONTRL	1
BUSINESS AFFAIRS	1
BUDGET & MGMT	1

24 row(s)

Total runtime: 11.077 ms

SQL executed.

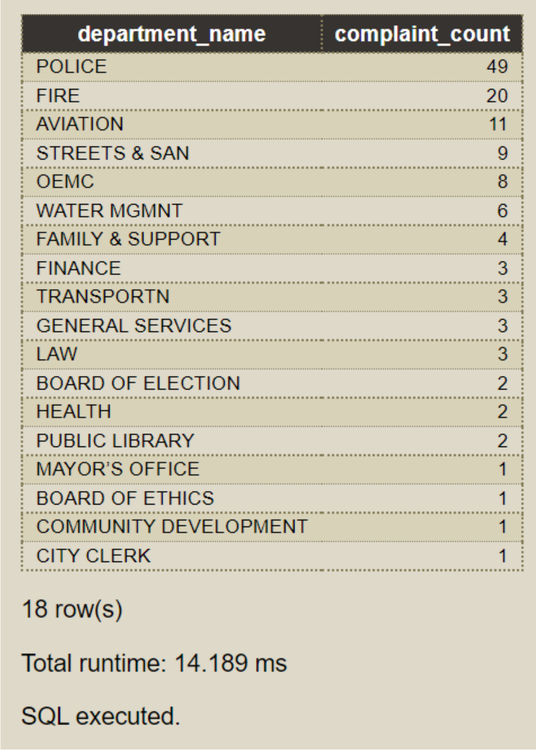


```
4j$ MATCH p=(d:Department)-[:Works_For]-(e:Employee)-[i:Involved_In]-(a:AccidentID) WHERE i.date_h…
```

	dep	no_of_accidents
1	"POLICE"	245
2	"FIRE"	92
3	"STREETS & SAN"	40
4	"OEMC"	32
5	"WATER MGMNT"	31
6	"AVIATION"	28
7		

rted streaming 24 records after 1 ms and completed after 9 ms.

2.) Complaint counts by department for year 2020

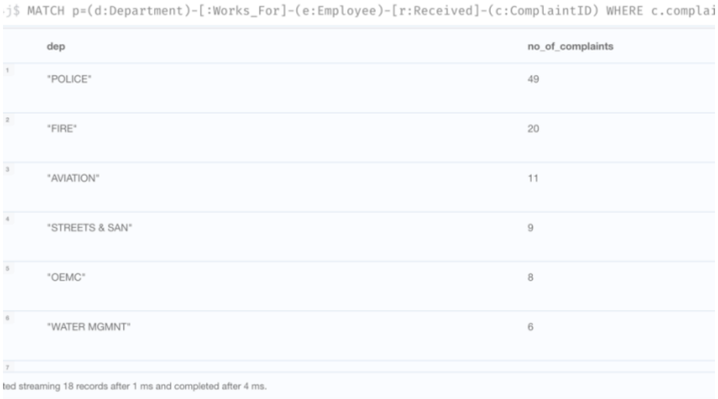


department_name	complaint_count
POLICE	49
FIRE	20
AVIATION	11
STREETS & SAN	9
OEMC	8
WATER MGMNT	6
FAMILY & SUPPORT	4
FINANCE	3
TRANSPORTN	3
GENERAL SERVICES	3
LAW	3
BOARD OF ELECTION	2
HEALTH	2
PUBLIC LIBRARY	2
MAYOR'S OFFICE	1
BOARD OF ETHICS	1
COMMUNITY DEVELOPMENT	1
CITY CLERK	1

18 row(s)

Total runtime: 14.189 ms

SQL executed.

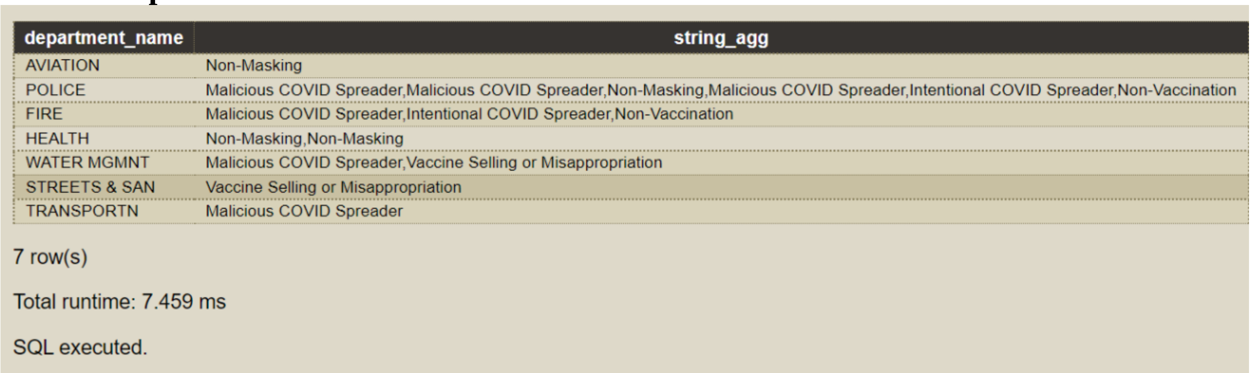


```
j$ MATCH p=(d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID) WHERE c.complai
```

	dep	no_of_complaints
1	"POLICE"	49
2	"FIRE"	20
3	"AVIATION"	11
4	"STREETS & SAN"	9
5	"OEMC"	8
6	"WATER MGMNT"	6
7		

ted streaming 18 records after 1 ms and completed after 4 ms.

3.) List of complaint description of complaints from each department in 2020 where compliant class is 'COVID'

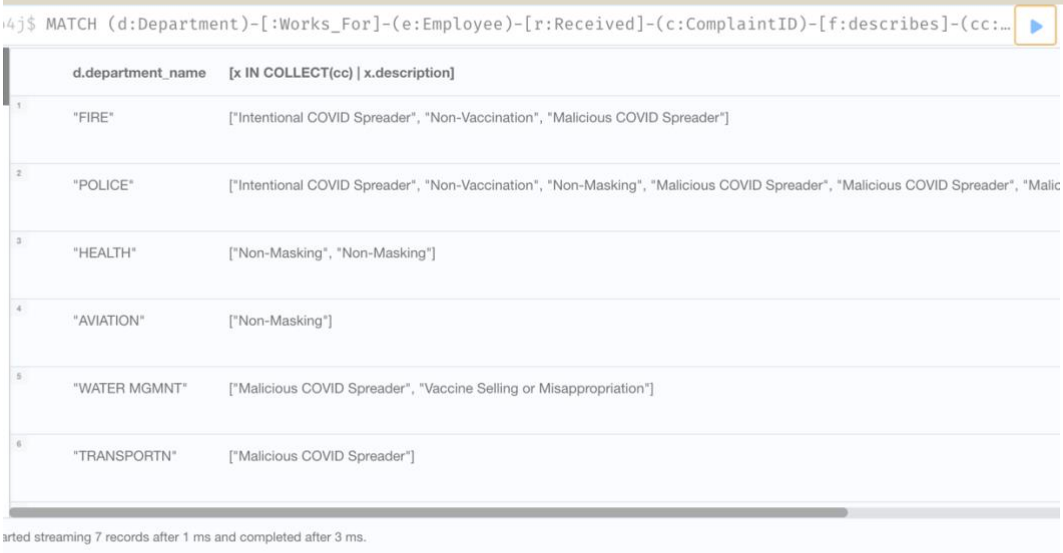


department_name	string_agg
AVIATION	Non-Masking
POLICE	Malicious COVID Spreader,Malicious COVID Spreader,Non-Masking,Malicious COVID Spreader,Intentional COVID Spreader,Non-Vaccination
FIRE	Malicious COVID Spreader,Intentional COVID Spreader,Non-Vaccination
HEALTH	Non-Masking,Non-Masking
WATER MGMNT	Malicious COVID Spreader,Vaccine Selling or Misappropriation
STREETS & SAN	Vaccine Selling or Misappropriation
TRANSPORTN	Malicious COVID Spreader

7 row(s)

Total runtime: 7.459 ms

SQL executed.

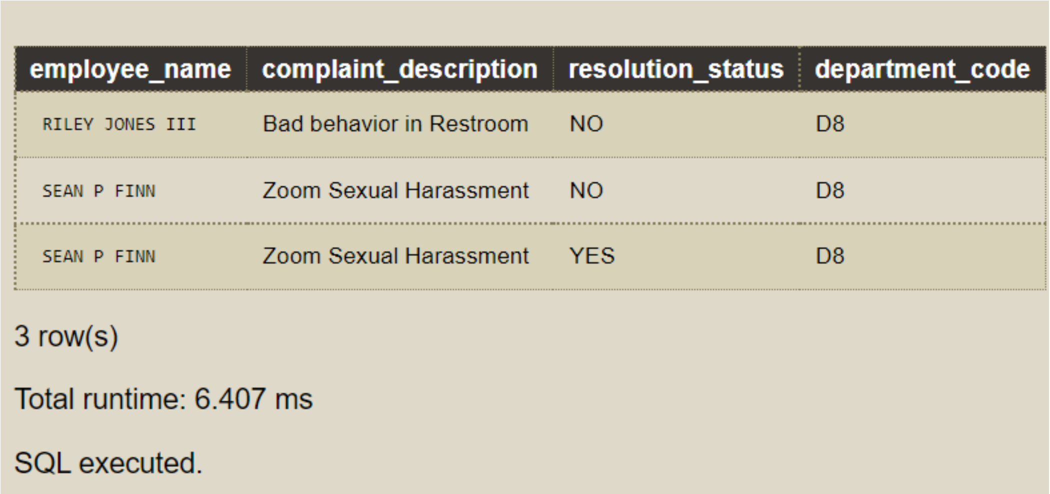


```
4j$ MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID)-[f:describes]-(cc:…
```

	d.department_name	[x IN COLLECT(cc) | x.description]
1	"FIRE"	["Intentional COVID Spreader", "Non-Vaccination", "Malicious COVID Spreader"]
2	"POLICE"	["Intentional COVID Spreader", "Non-Vaccination", "Non-Masking", "Malicious COVID Spreader", "Malicious COVID Spreader", "Malic
3	"HEALTH"	["Non-Masking", "Non-Masking"]
4	"AVIATION"	["Non-Masking"]
5	"WATER MGMNT"	["Malicious COVID Spreader", "Vaccine Selling or Misappropriation"]
6	"TRANSPORTN"	["Malicious COVID Spreader"]

arted streaming 7 records after 1 ms and completed after 3 ms.

4.) Concatenated (first name and last name) of employees who works for fire department who has received a complaint class as 'SexOffense' in 2020 with the resolved status and complaint description, order alphabetically.

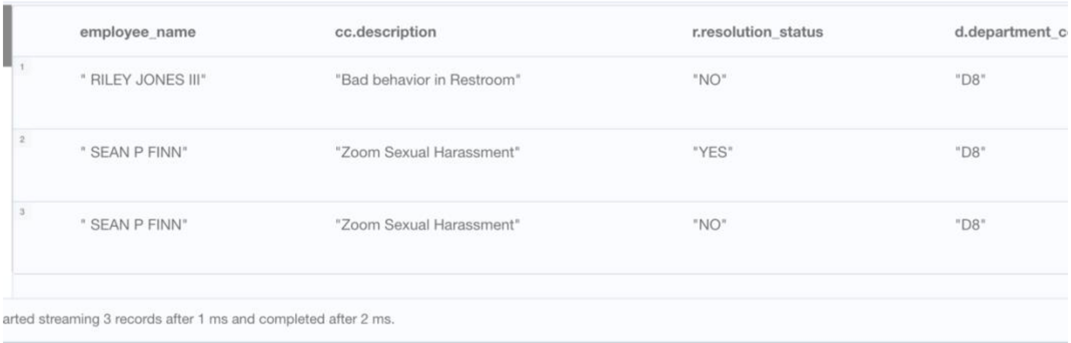


employee_name	complaint_description	resolution_status	department_code
RILEY JONES III	Bad behavior in Restroom	NO	D8
SEAN P FINN	Zoom Sexual Harassment	NO	D8
SEAN P FINN	Zoom Sexual Harassment	YES	D8

3 row(s)

Total runtime: 6.407 ms

SQL executed.

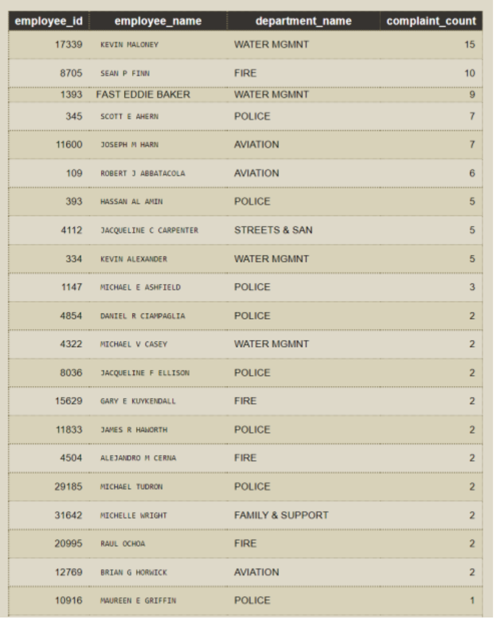


employee_name	cc.description	r.resolution_status	d.department_c
" RILEY JONES III"	"Bad behavior in Restroom"	"NO"	"D8"
" SEAN P FINN"	"Zoom Sexual Harassment"	"YES"	"D8"
" SEAN P FINN"	"Zoom Sexual Harassment"	"NO"	"D8"

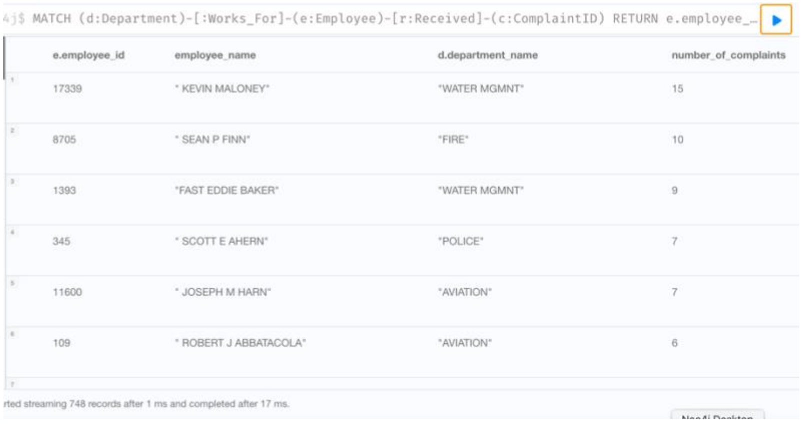
```
arted streaming 3 records after 1 ms and completed after 2 ms.
```

5.) Number of complaint_ids issued for each employee, ordered by number of complaints.
-- output full name, employee_id, department name along with complaint_id count.

***(Result was very long – screenshot of first page) ***



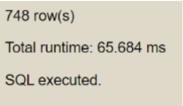
employee_id	employee_name	department_name	complaint_count
17339	KEVIN MALONEY	WATER MGMNT	15
8705	SEAN P FINN	FIRE	10
1393	FAST EDDIE BAKER	WATER MGMNT	9
345	SCOTT E AHERN	POLICE	7
11600	JOSEPH M HARN	AVIATION	7
109	ROBERT J ABBATACOLA	AVIATION	6
393	HASSAN AL AMIN	POLICE	5
4112	JACQUELINE C CARPENTER	STREETS & SAN	5
334	KEVIN ALEXANDER	WATER MGMNT	5
1147	MICHAEL E ASHFIELD	POLICE	3
4854	DANIEL R CIAMPAGLIA	POLICE	2
4322	MICHAEL V CASEY	WATER MGMNT	2
8036	JACQUELINE F ELLISON	POLICE	2
15629	GARY E KUYKENDALL	FIRE	2
11833	JAMES R HAWORTH	POLICE	2
4504	ALEJANDRO M CERNA	FIRE	2
29185	MICHAEL TUDRON	POLICE	2
31642	MICHELLE WRIGHT	FAMILY & SUPPORT	2
20995	RAUL OCHOA	FIRE	2
12769	BRIAN G HORWICK	AVIATION	2
10916	MAUREEN E GRIFFIN	POLICE	1



```
4j$ MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID) RETURN e.employee_…
```

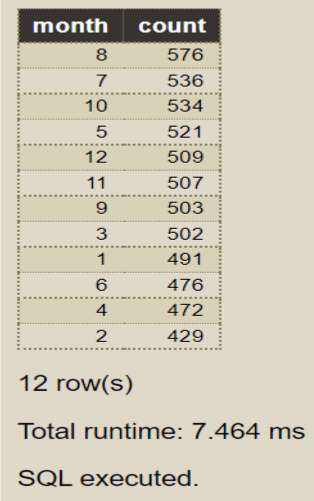
e.employee_id	employee_name	d.department_name	number_of_complaints
17339	" KEVIN MALONEY"	"WATER MGMNT"	15
8705	" SEAN P FINN"	"FIRE"	10
1393	"FAST EDDIE BAKER"	"WATER MGMNT"	9
345	" SCOTT E AHERN"	"POLICE"	7
11600	" JOSEPH M HARN"	"AVIATION"	7
109	" ROBERT J ABBATACOLA"	"AVIATION"	6

```
rted streaming 748 records after 1 ms and completed after 17 ms.
```



```
748 row(s)

Total runtime: 65.684 ms

SQL executed.
```

6.) Accident count by month ordered by month with highest count to lowest count.

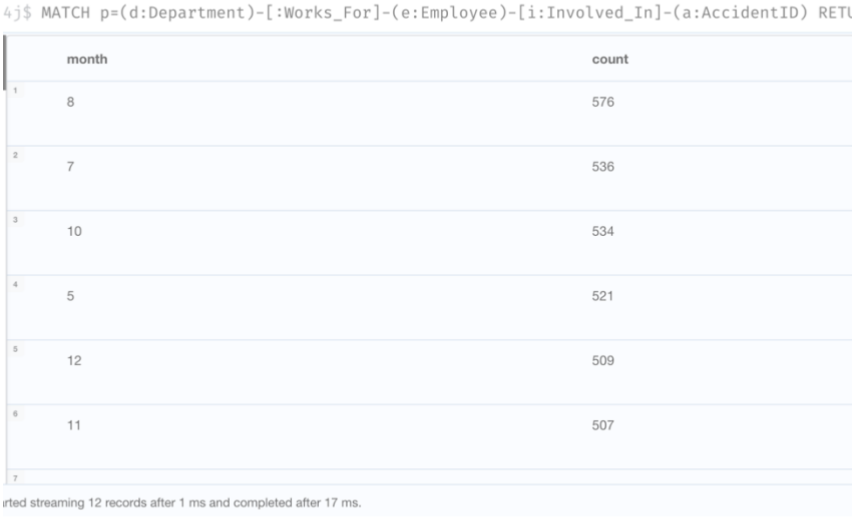


month	count
8	576
7	536
10	534
5	521
12	509
11	507
9	503
3	502
1	491
6	476
4	472
2	429

12 row(s)

Total runtime: 7.464 ms

SQL executed.



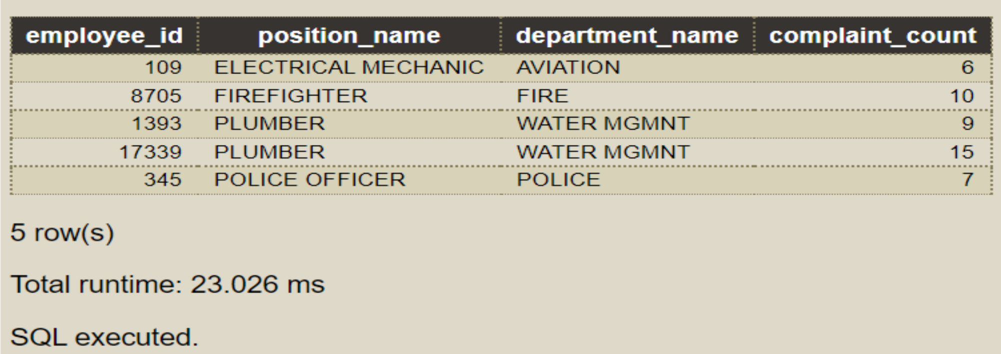
```
4j$ MATCH p=(d:Department)-[:Works_For]-(e:Employee)-[i:Involved_In]-(a:AccidentID) RETU
```

month	count
8	576
7	536
10	534
5	521
12	509
11	507

rted streaming 12 records after 1 ms and completed after 17 ms.

7.) Employees who get salary more than 70000 and received more than 5 complaint-ids.

-- Output employee_id, their position, department that they are working for and complaint count.

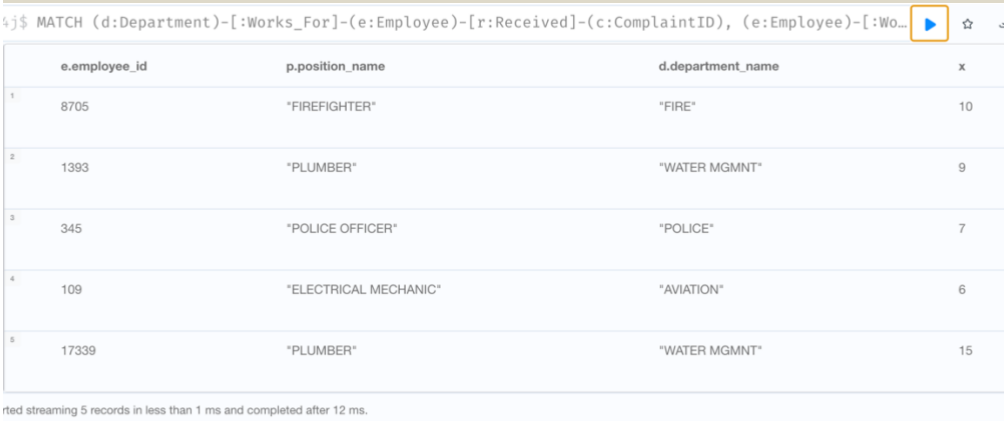


employee_id	position_name	department_name	complaint_count
109	ELECTRICAL MECHANIC	AVIATION	6
8705	FIREFIGHTER	FIRE	10
1393	PLUMBER	WATER MGMNT	9
17339	PLUMBER	WATER MGMNT	15
345	POLICE OFFICER	POLICE	7

5 row(s)

Total runtime: 23.026 ms

SQL executed.

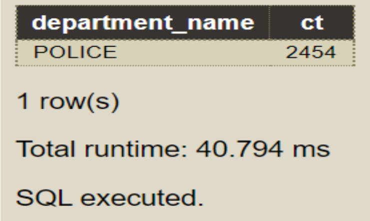


```
4j$ MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:ComplaintID), (e:Employee)-[:Wo…
```

e.employee_id	p.position_name	d.department_name	x
8705	"FIREFIGHTER"	"FIRE"	10
1393	"PLUMBER"	"WATER MGMNT"	9
345	"POLICE OFFICER"	"POLICE"	7
109	"ELECTRICAL MECHANIC"	"AVIATION"	6
17339	"PLUMBER"	"WATER MGMNT"	15

rted streaming 5 records in less than 1 ms and completed after 12 ms.

8.) Department with highest number of accidents

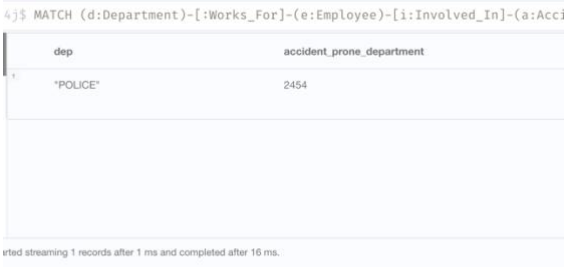


department_name	ct
POLICE	2454

1 row(s)

Total runtime: 40.794 ms

SQL executed.

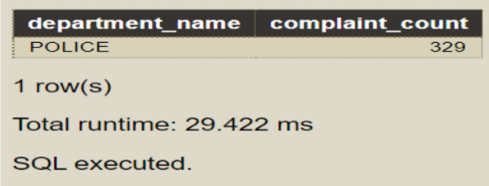


```
4j$ MATCH (d:Department)-[:Works_For]-(e:Employee)-[i:Involved_In]-(a:Acci
```

dep	accident_prone_department
"POLICE"	2454

rted streaming 1 records after 1 ms and completed after 16 ms.

9.) Department with highest number of complaints

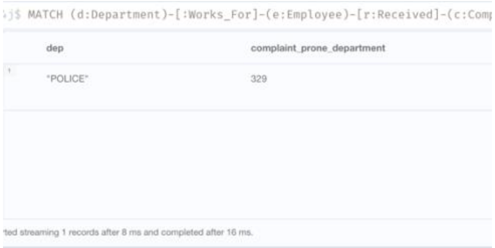


department_name	complaint_count
POLICE	329

1 row(s)

Total runtime: 29.422 ms

SQL executed.

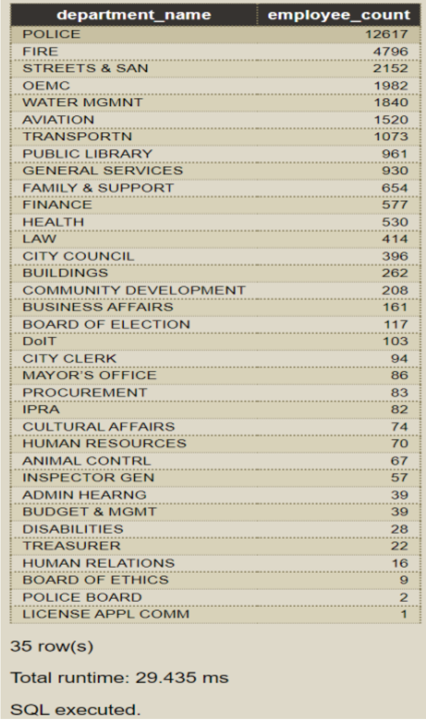


```
4j$ MATCH (d:Department)-[:Works_For]-(e:Employee)-[r:Received]-(c:Comp
```

dep	complaint_prone_department
"POLICE"	329

ted streaming 1 records after 8 ms and completed after 16 ms.

10.) Number of employees in each department in descending order

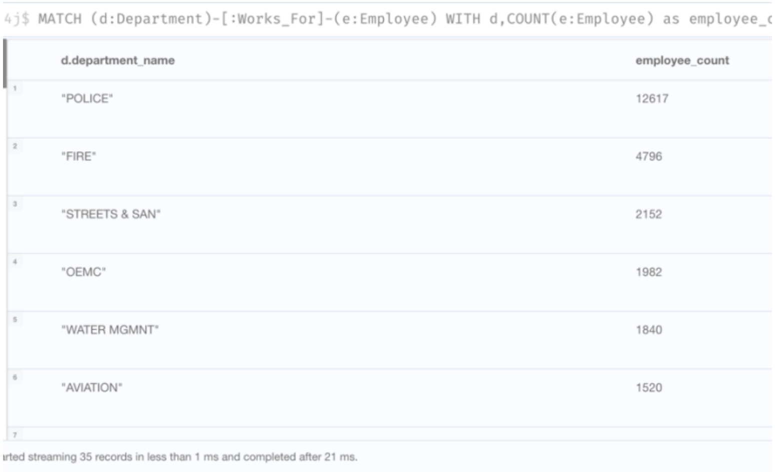


department_name	employee_count
POLICE	12617
FIRE	4796
STREETS & SAN	2152
OEMC	1982
WATER MGMNT	1840
AVIATION	1520
TRANSPORTN	1073
PUBLIC LIBRARY	961
GENERAL SERVICES	930
FAMILY & SUPPORT	654
FINANCE	577
HEALTH	530
LAW	414
CITY COUNCIL	396
BUILDINGS	262
COMMUNITY DEVELOPMENT	208
BUSINESS AFFAIRS	161
BOARD OF ELECTION	117
DoIT	103
CITY CLERK	94
MAYOR'S OFFICE	86
PROCUREMENT	83
IPRA	82
CULTURAL AFFAIRS	74
HUMAN RESOURCES	70
ANIMAL CONTRL	67
INSPECTOR GEN	57
ADMIN HEARNG	39
BUDGET & MGMT	39
DISABILITIES	28
TREASURER	22
HUMAN RELATIONS	16
BOARD OF ETHICS	9
POLICE BOARD	2
LICENSE APPL COMM	1

35 row(s)

Total runtime: 29.435 ms

SQL executed.



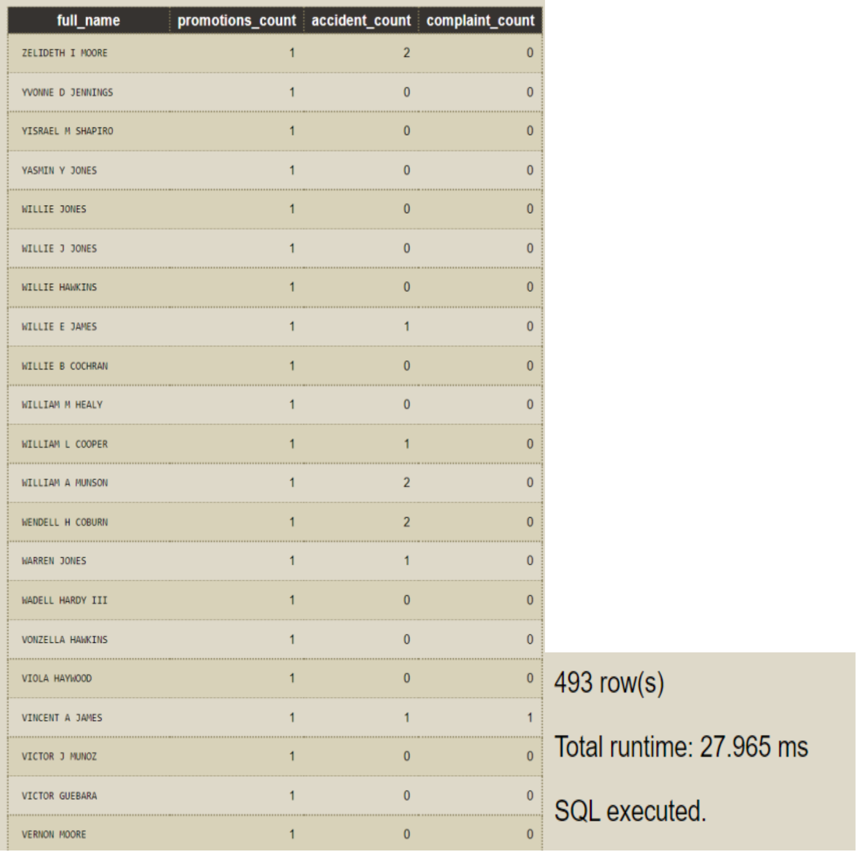
```
4j$ MATCH (d:Department)-[:Works_For]-(e:Employee) WITH d,COUNT(e:Employee) as employee_c
```

d.department_name	employee_count
"POLICE"	12617
"FIRE"	4796
"STREETS & SAN"	2152
"OEMC"	1982
"WATER MGMNT"	1840
"AVIATION"	1520

rted streaming 35 records in less than 1 ms and completed after 21 ms.

11.) Promotion counts for employees who got at least 1 promotion (with their full name) in descending order with their accident count and complaint count.

***(Result was very long – screenshot of first page) ***



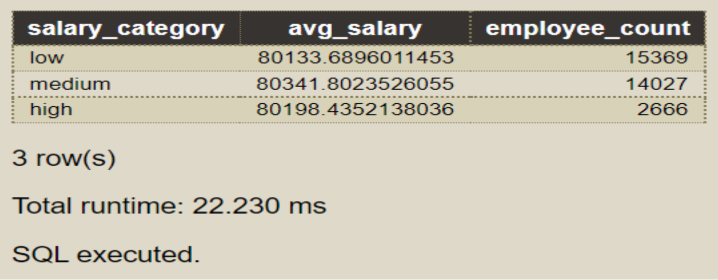
full_name	promotions_count	accident_count	complaint_count
ZELIDETH I MOORE	1	2	0
YVONNE D JENNINGS	1	0	0
YISRAEL M SHAPIRO	1	0	0
YASMIN Y JONES	1	0	0
WILLIE JONES	1	0	0
WILLIE J JONES	1	0	0
WILLIE HAWKINS	1	0	0
WILLIE E JAMES	1	1	0
WILLIE B COCHRAN	1	0	0
WILLIAM M HEALY	1	0	0
WILLIAM L COOPER	1	1	0
WILLIAM A MUNSON	1	2	0
WENDELL H COBURN	1	2	0
WARREN JONES	1	1	0
WADELL HARDY III	1	0	0
VONZELLA HAWKINS	1	0	0
VIOLA HAYWOOD	1	0	0
VINCENT A JAMES	1	1	1
VICTOR J MUNOZ	1	0	0
VICTOR GUEBARA	1	0	0
VERNON MOORE	1	0	0

493 row(s)

Total runtime: 27.965 ms

SQL executed.

12.) Average salary, number of employees for each salary category



salary_category	avg_salary	employee_count
low	80133.6896011453	15369
medium	80341.8023526055	14027
high	80198.4352138036	2666

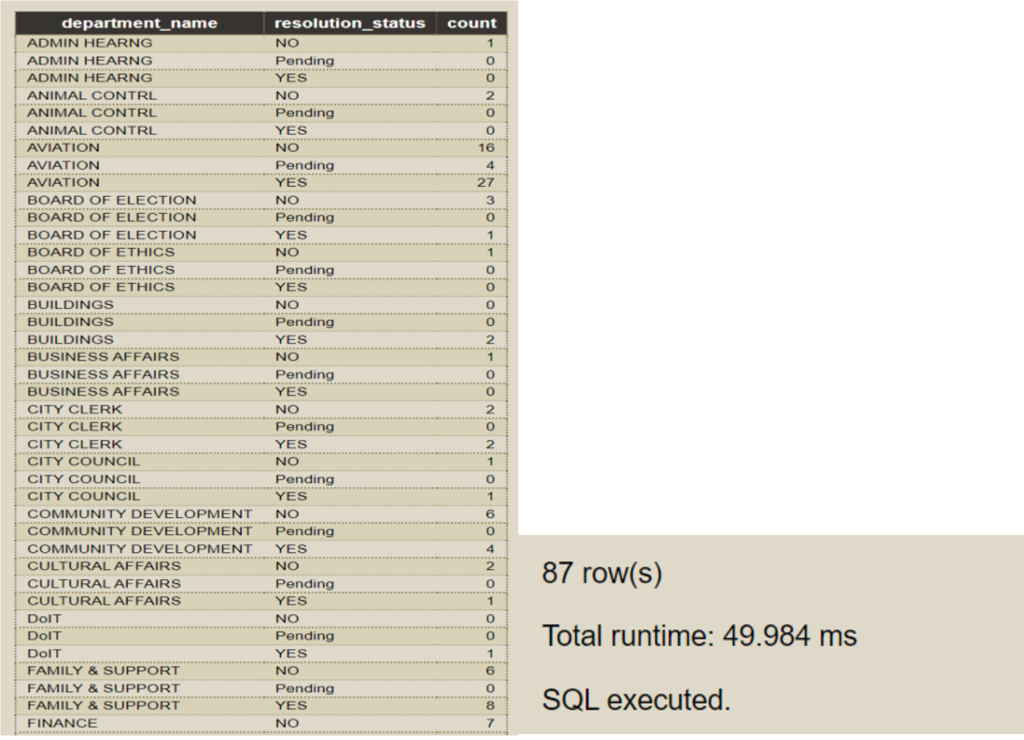
3 row(s)

Total runtime: 22.230 ms

SQL executed.

13.) Number of pending, resolved and non-resolved complaint count for each department.

***(Result was very long – screenshot of first page) ***



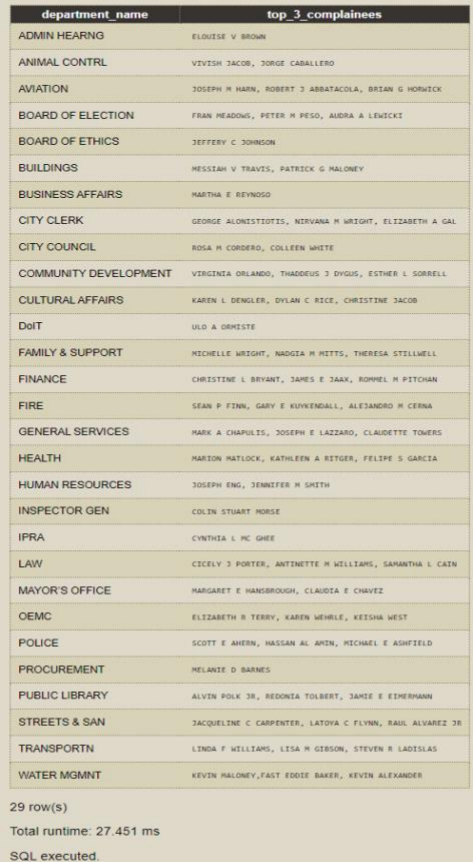
department_name	resolution_status	count
ADMIN HEARNG	NO	1
ADMIN HEARNG	Pending	0
ADMIN HEARNG	YES	0
ANIMAL CONTRL	NO	2
ANIMAL CONTRL	Pending	0
ANIMAL CONTRL	YES	0
AVIATION	NO	16
AVIATION	Pending	4
AVIATION	YES	27
BOARD OF ELECTION	NO	3
BOARD OF ELECTION	Pending	0
BOARD OF ELECTION	YES	1
BOARD OF ETHICS	NO	1
BOARD OF ETHICS	Pending	0
BOARD OF ETHICS	YES	0
BUILDINGS	NO	0
BUILDINGS	Pending	0
BUILDINGS	YES	2
BUSINESS AFFAIRS	NO	1
BUSINESS AFFAIRS	Pending	0
BUSINESS AFFAIRS	YES	0
CITY CLERK	NO	2
CITY CLERK	Pending	0
CITY CLERK	YES	2
CITY COUNCIL	NO	1
CITY COUNCIL	Pending	0
CITY COUNCIL	YES	1
COMMUNITY DEVELOPMENT	NO	6
COMMUNITY DEVELOPMENT	Pending	0
COMMUNITY DEVELOPMENT	YES	4
CULTURAL AFFAIRS	NO	2
CULTURAL AFFAIRS	Pending	0
CULTURAL AFFAIRS	YES	1
DoIT	NO	0
DoIT	Pending	0
DoIT	YES	1
FAMILY & SUPPORT	NO	6
FAMILY & SUPPORT	Pending	0
FAMILY & SUPPORT	YES	8
FINANCE	NO	7

87 row(s)

Total runtime: 49.984 ms

SQL executed.

14.) Top 3 complainees in each department



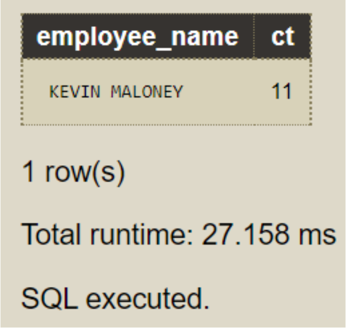
department_name	top_3_complainees
ADMIN HEARNG	ELOUISE V BROWN
ANIMAL CONTRL	VIVISH JACOB, JORGE CABALLERO
AVIATION	JOSEPH M HARN, ROBERT J ABBATACOLA, BRIAN G HORWICK
BOARD OF ELECTION	FRAN MEADOWS, PETER M PESO, AUDRA A LEWICKI
BOARD OF ETHICS	JEFFERY C JOHNSON
BUILDINGS	MESSIAH V TRAVIS, PATRICK G MALONEY
BUSINESS AFFAIRS	MARTHA E REYNOSO
CITY CLERK	GEORGE ALONISTIOTIS, NIRVANA M WRIGHT, ELIZABETH A GAL
CITY COUNCIL	ROSA M CORDERO, COLLEEN WHITE
COMMUNITY DEVELOPMENT	VIRGINIA ORLANDO, THADDEUS J DYGUS, ESTHER L SORRELL
CULTURAL AFFAIRS	KAREN L DENGLER, DYLAN C RICE, CHRISTINE JACOB
DoIT	ULO A ORMISTE
FAMILY & SUPPORT	MICHELLE WRIGHT, NADGIA M MITTS, THERESA STILLWELL
FINANCE	CHRISTINE L BRYANT, JAMES E JAAX, ROMMEL M PITCHAN
FIRE	SEAN P FINN, GARY E KUYKENDALL, ALEJANDRO M CERNA
GENERAL SERVICES	MARK A CHAPULIS, JOSEPH E LAZZARO, CLAUDETTE TOWERS
HEALTH	MARION MATLOCK, KATHLEEN A RITGER, FELIPE S GARCIA
HUMAN RESOURCES	JOSEPH ENG, JENNIFER M SMITH
INSPECTOR GEN	COLIN STUART MORSE
IPRA	CYNTHIA L MC GHEE
LAW	CICELY J PORTER, ANTINETTE M WILLIAMS, SAMANTHA L CAIN
MAYOR'S OFFICE	MARGARET E HANSBROUGH, CLAUDIA E CHAVEZ
OEMC	ELIZABETH R TERRY, KAREN WEHRLE, KEISHA WEST
POLICE	SCOTT E AHERN, HASSAN AL AMIN, MICHAEL E ASHFIELD
PROCUREMENT	MELANIE D BARNES
PUBLIC LIBRARY	ALVIN POLK JR, REDONIA TOLBERT, JAMIE E EIMERMANN
STREETS & SAN	JACQUELINE C CARPENTER, LATOYA C FLYNN, RAUL ALVAREZ JR
TRANSPORTN	LINDA F WILLIAMS, LISA M GIBSON, STEVEN R LADISLAS
WATER MGMNT	KEVIN MALONEY,FAST EDDIE BAKER, KEVIN ALEXANDER

29 row(s)

Total runtime: 27.451 ms

SQL executed.

15.) Employee who has received highest number of distinct complaint codes

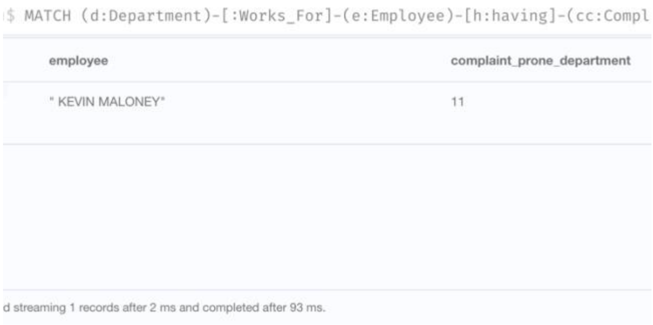


employee_name	ct
KEVIN MALONEY	11

1 row(s)

Total runtime: 27.158 ms

SQL executed.



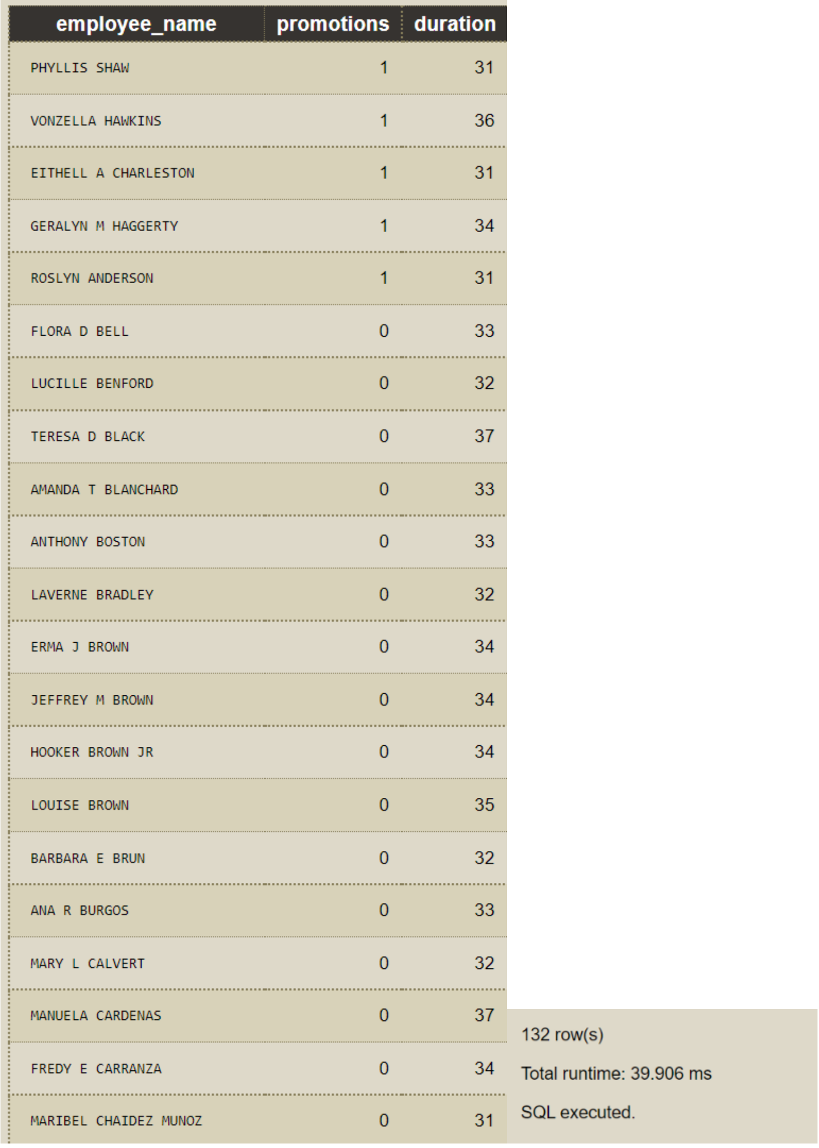
```
$ MATCH (d:Department)-[:Works_For]-(e:Employee)-[h:having]-(cc:Compl
```

employee	complaint_prone_department
" KEVIN MALONEY"	11

d streaming 1 records after 2 ms and completed after 93 ms.

16.) Employees working for 'FAMILY & SUPPORT' department for more than 30 years, with their promotion count, duration

***(Result was very long – screenshot of first page) ***



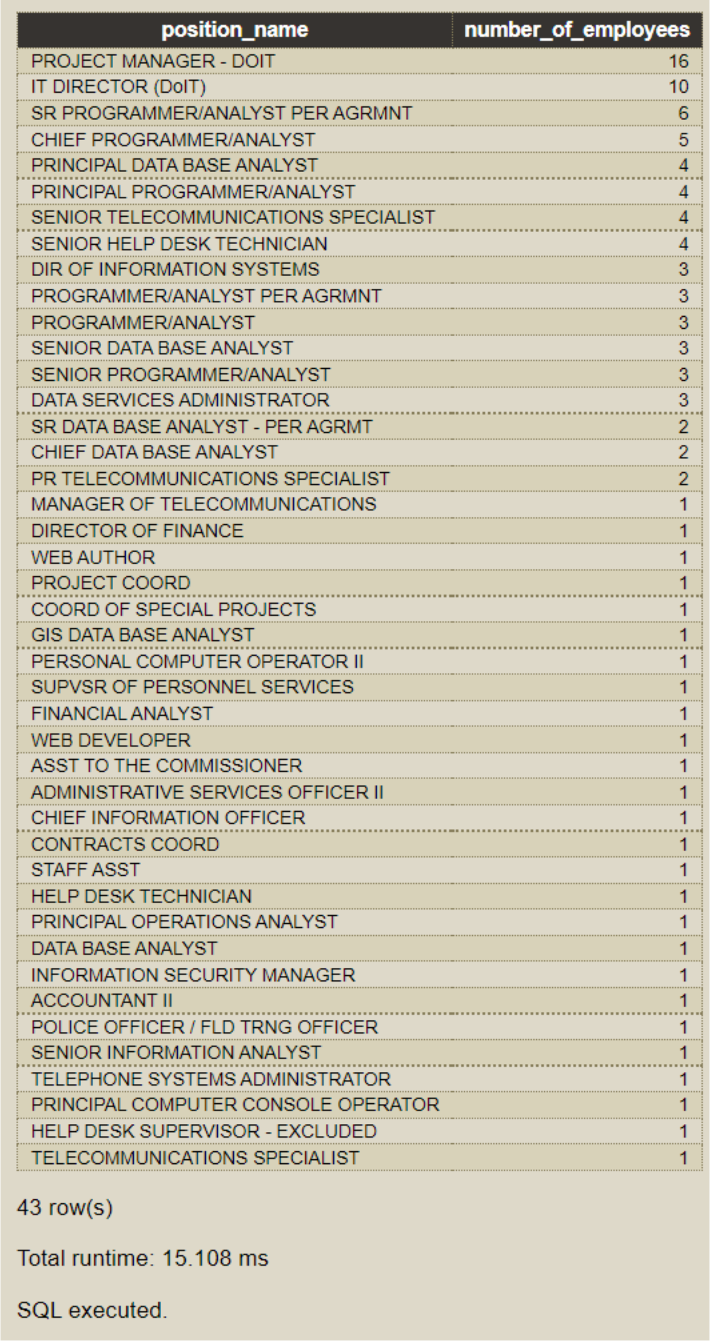
employee_name	promotions	duration
PHYLLIS SHAW	1	31
VONZELLA HAWKINS	1	36
EITHELL A CHARLESTON	1	31
GERALYN M HAGGERTY	1	34
ROSLYN ANDERSON	1	31
FLORA D BELL	0	33
LUCILLE BENFORD	0	32
TERESA D BLACK	0	37
AMANDA T BLANCHARD	0	33
ANTHONY BOSTON	0	33
LAVERNE BRADLEY	0	32
ERMA J BROWN	0	34
JEFFREY M BROWN	0	34
HOOKER BROWN JR	0	34
LOUISE BROWN	0	35
BARBARA E BRUN	0	32
ANA R BURGOS	0	33
MARY L CALVERT	0	32
MANUELA CARDENAS	0	37
FREDY E CARRANZA	0	34
MARIBEL CHAIDEZ MUNOZ	0	31

132 row(s)

Total runtime: 39.906 ms

SQL executed.

17.) All the positions offered from Department of IT (DoIT) with head count for each position



position_name	number_of_employees
PROJECT MANAGER - DOIT	16
IT DIRECTOR (DoIT)	10
SR PROGRAMMER/ANALYST PER AGRMNT	6
CHIEF PROGRAMMER/ANALYST	5
PRINCIPAL DATA BASE ANALYST	4
PRINCIPAL PROGRAMMER/ANALYST	4
SENIOR TELECOMMUNICATIONS SPECIALIST	4
SENIOR HELP DESK TECHNICIAN	4
DIR OF INFORMATION SYSTEMS	3
PROGRAMMER/ANALYST PER AGRMNT	3
PROGRAMMER/ANALYST	3
SENIOR DATA BASE ANALYST	3
SENIOR PROGRAMMER/ANALYST	3
DATA SERVICES ADMINISTRATOR	3
SR DATA BASE ANALYST - PER AGRMT	2
CHIEF DATA BASE ANALYST	2
PR TELECOMMUNICATIONS SPECIALIST	2
MANAGER OF TELECOMMUNICATIONS	1
DIRECTOR OF FINANCE	1
WEB AUTHOR	1
PROJECT COORD	1
COORD OF SPECIAL PROJECTS	1
GIS DATA BASE ANALYST	1
PERSONAL COMPUTER OPERATOR II	1
SUPVSR OF PERSONNEL SERVICES	1
FINANCIAL ANALYST	1
WEB DEVELOPER	1
ASST TO THE COMMISSIONER	1
ADMINISTRATIVE SERVICES OFFICER II	1
CHIEF INFORMATION OFFICER	1
CONTRACTS COORD	1
STAFF ASST	1
HELP DESK TECHNICIAN	1
PRINCIPAL OPERATIONS ANALYST	1
DATA BASE ANALYST	1
INFORMATION SECURITY MANAGER	1
ACCOUNTANT II	1
POLICE OFFICER / FLD TRNG OFFICER	1
SENIOR INFORMATION ANALYST	1
TELEPHONE SYSTEMS ADMINISTRATOR	1
PRINCIPAL COMPUTER CONSOLE OPERATOR	1
HELP DESK SUPERVISOR - EXCLUDED	1
TELECOMMUNICATIONS SPECIALIST	1

43 row(s)

Total runtime: 15.108 ms

SQL executed.