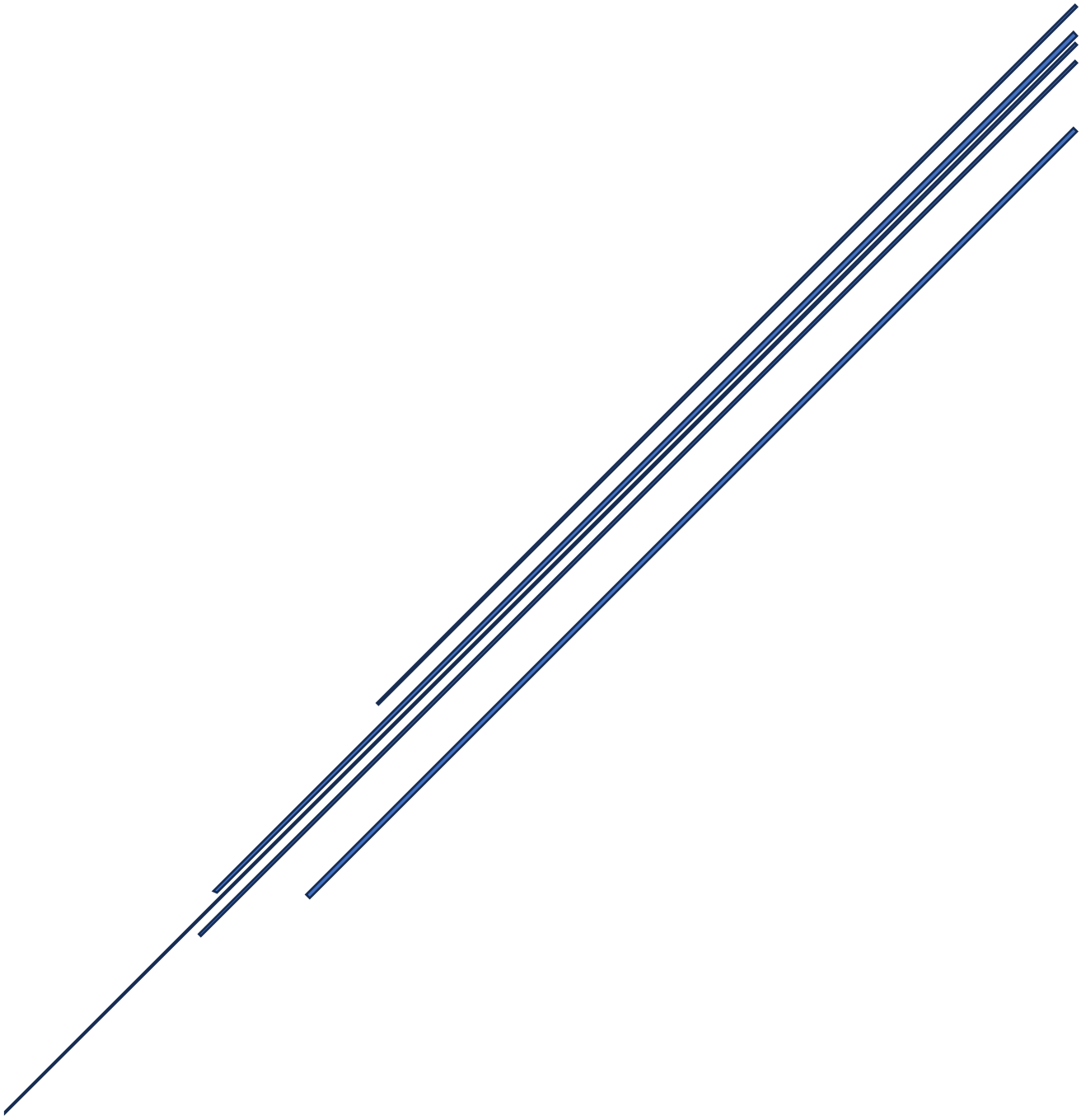
Federico Ariton

Database Mysql



Índex

Part A

1 Conceptual diagram using Chen notation

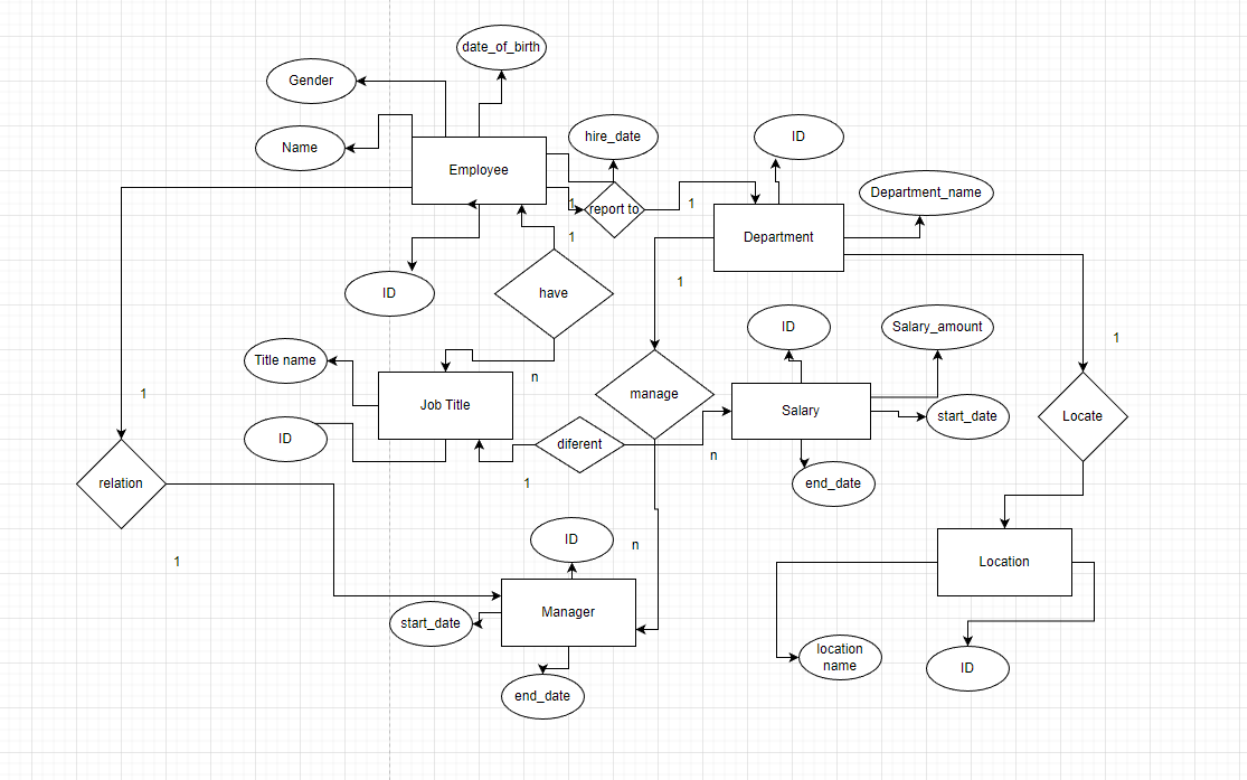


Figure 1 – Chen notation diagram

To initiate the process, we commence by identifying the entities that are to be incorporated in the diagram, namely employee, department, job title, salary, manager, and location. Each entity is depicted by a rectangular shape within the diagram. In Chen notation, a rectangle represents an entity, and its attributes are enlisted within it. Relationships between entities are illustrated by diamonds, with lines linking the entities to the diamonds. The lines signify the cardinality and optionality of the association.

The cardinality of a relationship determines the number of instances of an entity that can be linked with another. Next, we determine the attributes that correspond to each entity and represent them using circles, we establish relationships between the entities.

- An employee can have multiple job titles with different salaries.
- An employe report to one department
- An employe have one relation to one manager
- A department can have multiple managers over time.

- A job title can have multiple employees.
- A department can have multiple employees.
- A department have a one location

2 Logical ER Model into a relational data model

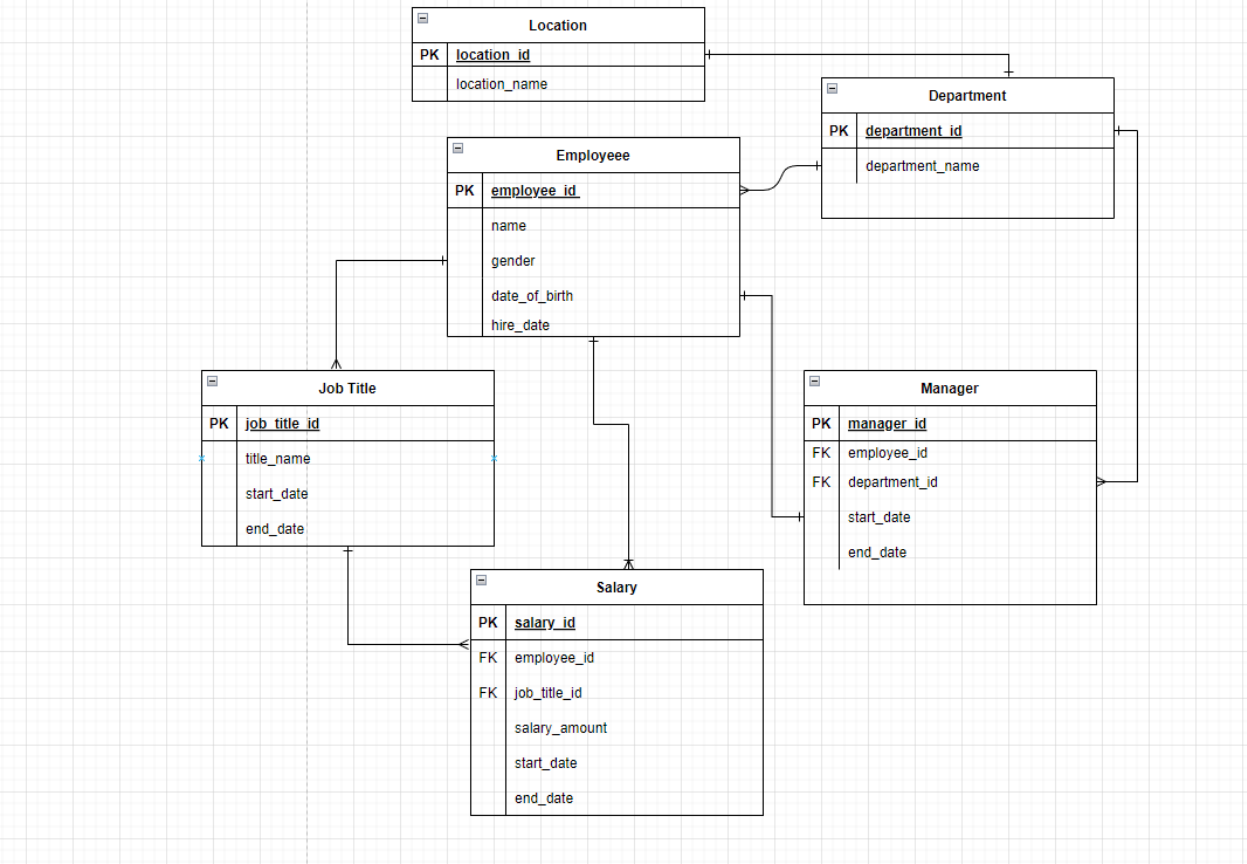


Figure 2 – Er Model

At its core, the ER model represents real-world objects, such as employee, departament, and manager as entities. Each entity has a set of attributes that describe it, and these attributes are listed inside a rectangle that represents the entity. One of the key strengths of the ER model is its ability to identify and describe complex relationships between entities For example, it can represent one-to-one relationships, one-to-many relationships , and many-to-many relationships.

The following entities are related with the figure 2:

The Employee table has a one-to-many relationship with the Salary table, where an employee can have multiple salaries over time.

The Employee table has a one-to-one relationship with the Manager table, where an employee can have at most one manager at a time.

The Manager table has a one-to-many relationship with the Department table, where a manager can manage only one department, but a department can have multiple managers over time.

The Salary table has a many-to-one relationship with the Employee table, where multiple salaries can belong to one employee.

The Salary table has a many-to-one relationship with the Job Title table, where multiple salaries can be associated with one job title.

The "Employee" table has a primary key "employee_id" which uniquely identifies each employee. This key is used as a foreign key in the "Salary" and "Manager" tables to link each salary or manager record to the employee who holds that position.

The "Department" table also has a primary key "department_id" which uniquely identifies each department. This key is used as a foreign key in the "Manager" table to link each manager record to the department they manage.

The "Job Title" table has a primary key "job_title_id" which uniquely identifies each job title. This key is used as a foreign key in the "Salary" table to link each salary record to the job title associated with that salary.

The "Salary" table has two foreign keys: "employee_id" and "job_title_id". These keys link each salary record to the employee who receives that salary and the job title associated with that salary.

The "Manager" table has two foreign keys: "employee_id" and "department_id". These keys link each manager record to the employee who holds that position and the department they manage.

The "Location" table has a primary key "location_id" which uniquely identifies each location. This table does not have any foreign keys linking it to other tables in this schema.

3 Normalisation

UNF	1NF	2NF	3NF

Employee=
Employee ID+
Name+
Gender+
Date_of_birth+
Hire_Date+
Departament+
Location+
{
Job title+
Start_Date+
End_Date+
}
{
Salary+
Start_Date+
End_Date+
}
{
Manager+
Start_Date+
End_Date+
}
	Employee=
	Employee ID+
	Name+
	Gender+
	Date_of_birth+
	Hire_Date+
	Departament+
	Location+
	Job_History=
	Job title+
	Start_Date+
	End_Date+
	Salary_History
	Salary+
	Start_Date+
	End_Date+
	Manager_History=
	Manager+
	Start_Date+
	End_Date+
		Employee=
		Employee ID+
		Name+
		Gender+
		Date_of_birth+
		Hire_Date+
		Departament=
		Departament_ID+
		Departament_name
		Location+
		Job_History=
		Job_ID
		Job title_name+
		Start_Date+
		End_Date+
		Salary_History=
		Salary_ID
		Salary_amount+
		Start_Date+
		End_Date+
		Manager_History=
		Manager_ID+
		Manager_name+
		Start_Date+
		End_Date+
			Employee=
			Employee ID+
			Name+
			Gender+
			Date_of_birth+
			Hire_Date+
			Departament=
			Departament_ID+
			Location=
			Location_ID+
			Location_name
			Job_History=
			Job_ID
			Job title_name+
			Start_Date+
			End_Date+
			Salary_History=
			Salary_ID
			Salary_amount+
			Start_Date+
			End_Date+
			Manager_History=
			Manager_ID+
			Manager_name+
			Start_Date+
			End_Date+

Figure 3 - Normalisation

The table shown represents the normalization process for an Employee table, starting from an Unnormalized Form (UNF) to the Third Normal Form (3NF).

In the UNF form, there is a single Employee table that includes all the attributes such as Employee ID, Name, Gender, Date_of_birth, Hire_Date, Department, Location, Job title, Salary, and Manager with their respective start and end dates.

In the first normal form (1NF), the table is split into multiple tables with each table representing a single entity. The Employee table is split into four tables, namely Employee_Details, Job_History, Salary_History, and Manager_History. The Employee_Details table contains employee-related details such as ID, Name, Gender, Date_of_birth, Hire_Date, Department, and Location. The Job_History table contains the job history of the employee such as job title, start date, and end date. The Salary_History table contains the salary details of the employee such as salary amount, start date, and end date. The Manager_History table contains the manager details of the employee such as manager name, start date, and end date.

In the second normal form (2NF), each table should have a primary key, and all non-key attributes should depend on the primary key. As we have already created separate tables for each entity, each table has a primary key, and all attributes in each table depend on that primary key.

In the third normal form (3NF), we eliminate the transitive dependency between non-key attributes. In this step, we split the Department attribute from the Employee_Details table and create a new table called Departament with attributes such as Departament_ID and Departament_name. Similarly, we split the Location attribute from the Employee_Details table and create a new table called Location with attributes such as Location_ID and Location_name. Now, the Employee_Details table only contains Employee_ID, Name, Gender, Date_of_birth, Hire_Date, and foreign keys for Department and Location. The Job_History, Salary_History, and Manager_History tables remain the same.

The final 3NF schema includes the following tables:

Employee_Details (Employee_ID, Name, Gender, Date_of_birth, Hire_Date, Department_ID, Location_ID)

Job_History (Employee_ID, Job_ID, Start_Date, End_Date)

Salary_History (Employee_ID, Salary_ID, Salary_amount, Start_Date, End_Date)

Manager_History (Employee_ID, Manager_ID, Manager_name, Start_Date, End_Date)

Departament (Department_ID, Department_name)

Location (Location_ID, Location_name)

The normalized schema allows for efficient storage, retrieval, and maintenance of data while minimizing data redundancy and inconsistency.

Part B

Question 1

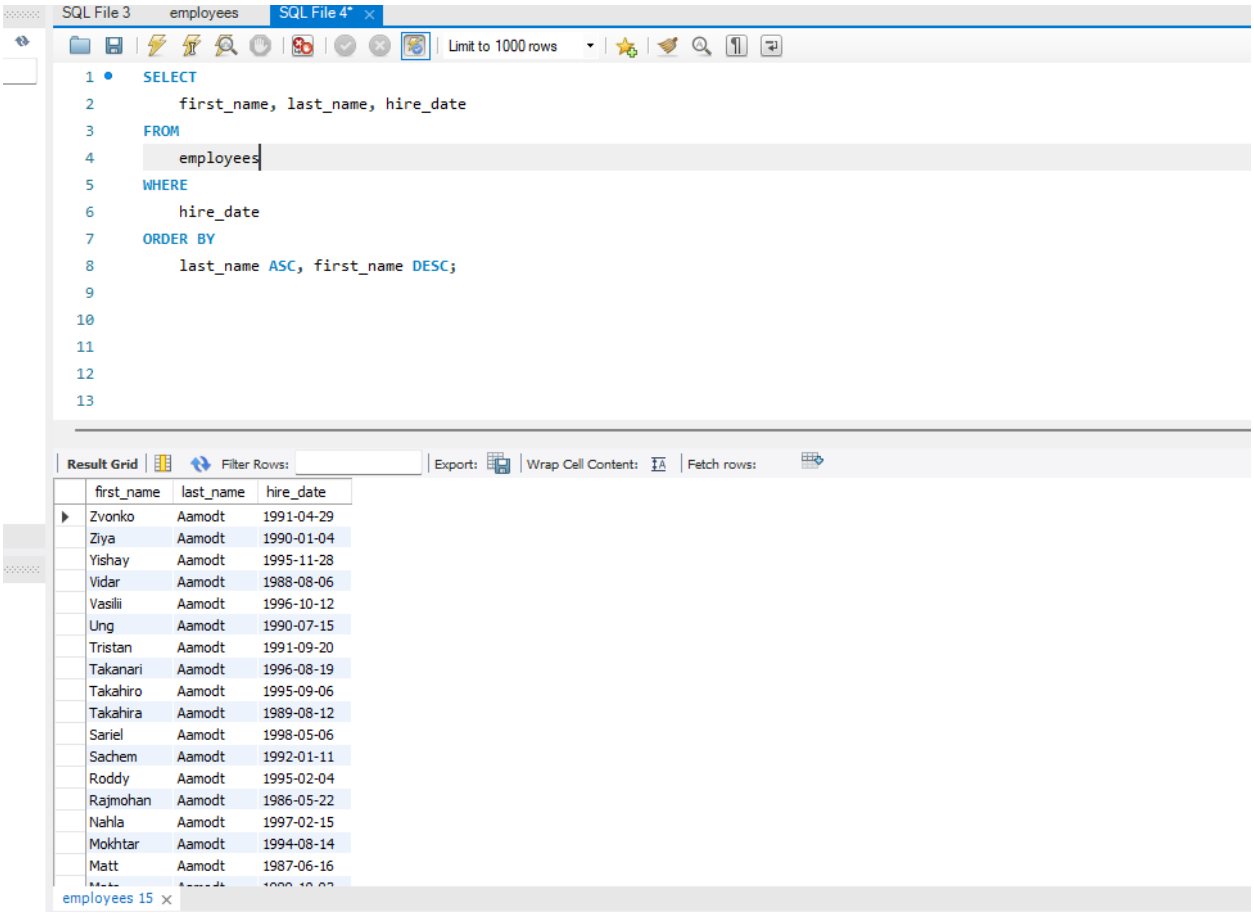


Figure 4 – Ordering the last name in ascending order and first name in descending order

This query will select the first name, last name, and hire date for all employees who are currently employed (i.e., whose hire date is in the past), and it will order the results first by last name in ascending order, and then by first name in descending order. The COUNT(*) function counts the number of records that meet the condition specified in the WHERE clause, and the num_employees_hired is assigned to the count result. The WHERE clause filters the results to only include records where the hire_date is greater than or equal to '1992-12-01'.

Question 2

```
9
10
11 •  SELECT
12        COUNT(*) as num_employees_hired
13    FROM
14        employees
15    WHERE
16        hire_date >= '1992-12-01';
17
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

num_employees_hired
13971

Figure 5- Employee hired after December 1st 1992

This query will count the number of employees who were hired on or after December 1st, 1992, by selecting all rows from the "employees" table where the hire date is greater than or equal to December 1st, 1992. The result will be a single row with a single column named "num_employees_hired" containing the count of employees hired.

Question 3

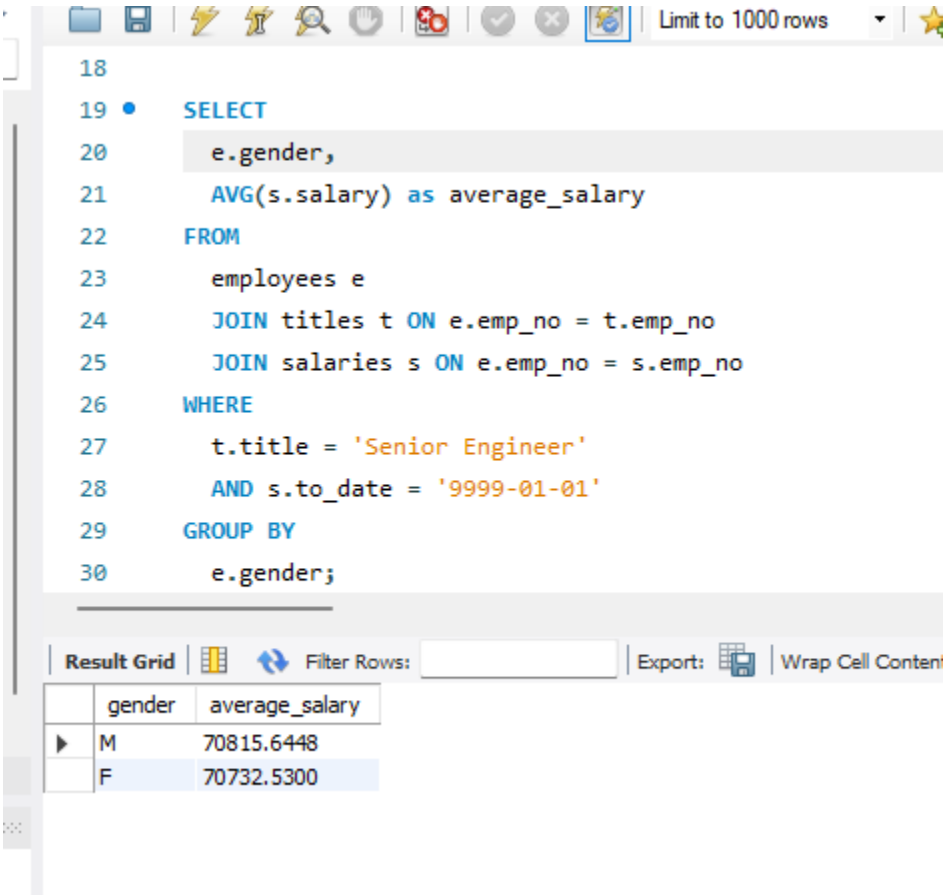


Figure 6- Average salary of current salary for the employees has the title Senior Engineer.

In this query, we're selecting the gender column from the employees table and the average salary of each gender group using the AVG() function. We're joining the employees, titles, and salaries tables to get the required data. We're also filtering the results to only include employees with the title of "Senior Engineer" and with the most recent salary information. Finally, we're grouping the results by gender.

Question 4

```
31
32 •  SELECT
33      COUNT(*) as employee_count
34    FROM
35      employees e
36      JOIN dept_emp de ON e.emp_no = de.emp_no
37    WHERE
38      e.last_name LIKE 'E%'
39    AND
40      de.to_date >= NOW();
41
42
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_count
1242

Figure 7- Number of all employees that contain the letter E

In this query, we're selecting the COUNT(*) function to count the number of employees whose last name begins with the letter E and are current or past employees. We're joining the employees and dept_emp tables to get the required data. We're also filtering the results to only include employees whose last name begins with the letter E using the LIKE operator with the pattern 'E%'. Additionally, we're filtering the results to only include current or past employees by checking that the to_date column in the dept_emp table is greater than or equal to the current date and time using the NOW() function.

Question 5

```
41
42 •    SELECT
43      t.title as job_title,
44      AVG(s.salary) as avg_salary
45    FROM
46      employees e
47      JOIN titles t ON e.emp_no = t.emp_no
48      JOIN salaries s ON e.emp_no = s.emp_no
49    WHERE
50      s.to_date >= NOW()
51      AND t.to_date >= NOW()
52    GROUP BY
53      t.title;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

job_title	avg_salary
Senior Engineer	70782.3411
Staff	67335.1836
Senior Staff	80617.6160
Engineer	59710.1997
Assistant Engineer	57838.5634
Technique Leader	67152.0822

Figure 8 Return the job title with the average salary

In this query, we're selecting the job title from the titles table and the average salary from the salaries table for all past or present employees who held each title. We're joining the employees, titles, and salaries tables to get the required data. We're also filtering the results to only include current or past employees by checking that the to_date column in both the titles and salaries tables is greater than or equal to the current date and time using the NOW() function. Additionally, we're grouping the results by job title using the GROUP BY clause.

Question 6

SQL File 3 employees SQL File 4*

Limit to 1000 rows

```
55 •  SELECT
56      e.first_name,
57      e.last_name,
58      e.hire_date
59    FROM
60      employees e
61      JOIN titles t ON e.emp_no = t.emp_no
62    WHERE
63      e.first_name = 'Georgi'
64      AND e.birth_date BETWEEN '1958-01-01' AND '1961-12-31'
65      AND t.to_date = '9999-01-01';
66
67
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

first_name	last_name	hire_date
Georgi	Barinka	1985-06-04
Georgi	Varley	1987-04-14
Georgi	Maksimenko	1990-02-21
Georgi	Delgrossi	1987-08-13
Georgi	Selvestrel	1988-06-26
Georgi	Gecsel	1989-01-31
Georgi	Matzen	1991-06-28
Georgi	Zaumen	1991-11-30
Georgi	Worfolk	1992-04-01
Georgi	Hambrick	1989-12-12
Georgi	Grospietsch	1993-02-12
Georgi	Luce	1987-07-20

Figure 9 Return the name Georgi and date between January 1st 1958 and December 31st 1961.

In this query, we're selecting the first name, last name, and hiring date of all employees whose first name is "Georgi" and whose date of birth is between January 1st 1958 and December 31st 1961. We're joining the employees and titles tables to get the required data. We're also filtering the results to only include employees who meet the specified conditions by using the WHERE clause. Specifically, we're checking that the first name is "Georgi", the birth date is between January 1st 1958 and December 31st 1961 using the BETWEEN operator, and the title is current as of today's date by checking that the

to_date column in the titles table is equal to the maximum possible date (i.e., '9999-01-01').

Question 7

```
67
68
69  CREATE VIEW large_departments2 AS
70  SELECT
71   departments.dept_no, departments.dept_name, COUNT(*) as employee_count
72  FROM
73   departments
74  JOIN
75   dept_emp
76   ON departments.dept_no = dept_emp.dept_no
77  WHERE
78   dept_emp.to_date = '9999-01-01' -- only consider current employees
79  GROUP BY
80      departments.dept_no, departments.dept_name
81  HAVING
82      COUNT(*) >= 5000;
83
84
85
```

Figure 10- Create a view that return the departments where the employee count is 5,000 or greater

In this SQL code, we're creating a view named large_departments that returns the department number, department name, and employee count for all departments where the employee count is 5,000 or greater. To achieve this, we're joining the departments, dept_emp, and employees tables to get the necessary information. We're also filtering the results to only include current employees by checking that the to_date column in the employees table is equal to the maximum possible date (i.e., '9999-01-01'). Then, we're grouping the results by department number and department name and filtering the results to only include departments with an employee count of 5,000 or greater using the HAVING clause.

Question 8

SQL File 3 employees SQL File 4*

Limit to 1000 rows

```
84  SELECT
85      d.dept_name, e.first_name, e.last_name, e.birth_date
86  FROM
87      employees e
88  JOIN
89      dept_emp de
90      ON e.emp_no = de.emp_no
91  JOIN
92      departments d
93      ON de.dept_no = d.dept_no
94  JOIN large_departments ld
95      ON d.dept_no = ld.dept_no
96  ORDER BY
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

dept_name	first_name	last_name	birth_date
Development	Georgi	Facello	1953-09-02
Development	Anneke	Preusig	1953-04-20
Development	Saniya	Kalloufi	1958-02-19
Development	Patricio	Bridgland	1960-10-04
Development	Kazuhide	Peha	1954-06-19
Development	Shahaf	Famili	1952-07-08
Development	Bojan	Montemayor	1953-09-29
Development	Prasadram	Heyers	1958-10-31
Development	Divier	Reistad	1962-07-10
Development	Weiyi	Meriste	1959-09-13
Development	Yishay	Tzvieli	1960-09-19
Development	Florian	Syrotiuk	1963-07-11
Development	Ebbe	Callaway	1954-05-30
Development	Anoosh	Peyn	1961-11-02
Development	Satosi	Awdeh	1963-04-14
Development	Kwee	Schusler	1952-11-13
Development	Reuven	Garigliano	1955-08-20

Result 29

Figure 11 Return all employees who work for large departments and ordered by department name

This code is selecting the department name, first name, last name, and birth date of employees who work in departments that have more than or equal to 5000 employees. The data is obtained by joining four tables: employees, dept_emp, departments, and large_departments.

The first table employees stores the information about employees including their first name, last name, and birth date.

The second table dept_emp stores the information about which employee works in which department and for what time period.

The third table departments stores the information about departments including the department name and its ID.

The fourth table large_departments is a view that we created earlier in the database and it selects only the departments that have more than or equal to 5000 employees.

`

Question 9

```
92          departments d
93          ON de.dept_no = d.dept_no
94      JOIN large_departments ld
95          ON d.dept_no = ld.dept_no
96      ORDER BY
97          d.dept_name;
98
99  •   ALTER TABLE
100         employees
101      ADD
102         email_address VARCHAR(120);
103
104
105
106
107
108
```

Figure 12- Adding a new column called email_address with length of 120 characters.

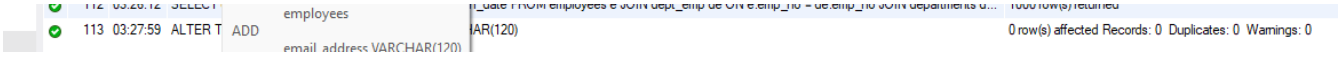


Figure 13- Validation that was added in the system.

This will add a new column named email_address to the employees table with a maximum length of 120 characters. You can then update the values in this column for each employee as needed.

Question 10

```
103
104 •   UPDATE
105         employees
106     SET
107      email_address = 'sumant.peac@mycompany.com'
108     WHERE
109         first_name = 'Sumant'
110      AND
111         last_name = 'Peac';
112
113
114
115
116
117
```

Figure 14- Update the address to the employee name Sumant Peac.

```
8  12:21:09  UPDATE employees SET email_address = 'sumant.peac@mycompany.com' WHERE first_name = 'Sumant' AND last_name = 'Peac'   1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0
```

Figure 15- Validation that the address it was updated

This is an SQL code to update the email address of an employee named Sumant Peac in the employees table. The code uses the UPDATE statement to modify the email_address field for the employee record that matches the conditions specified in the WHERE clause. Specifically, the WHERE clause filters the employee records where the first_name is 'Sumant' and last_name is 'Peac', and updates the email_address field with the new email address 'sumant.peac@mycompany.com'.

Question 11

```
113 •   DELETE FROM
114         dept_emp
115     WHERE
116         to_date != '9999-01-01';
117
118
119
```

Figure 16- Delete all records in the dept_emp table where the value in to_date field is not '9999-01-01'.

Figure 17- Validation that was deleted all the records from the table dept_emp

This code uses the DELETE statement to remove all records from the dept_emp table where the value in the to_date field is not equal to '9999-01-01'. The != operator is used to specify the condition that the to_date field should not have the value '9999-01-01'.

Question 12

```
118
119 • CREATE TABLE training (
120     trainer_no int NOT NULL AUTO_INCREMENT,
121     first_name varchar(30) NOT NULL,
122     last_name varchar(30) NOT NULL,
123     subject varchar(20),
124     PRIMARY KEY (trainer_no)
125 );
126
```

Figure 18- Creating a new table called training with 4 columns.

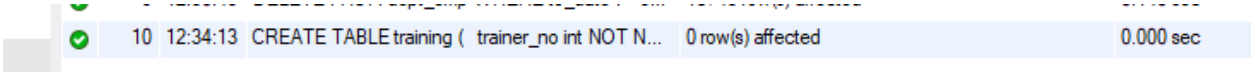


Figure 19- Validation that was created the table

This will create a new table called training with four columns: trainer_no, first_name, last_name, and subject. The trainer_no column is an auto-incrementing integer and is set as the primary key. The first_name and last_name columns are set as varchar(30) and are set to not allow NULL values. The subject column is also set as varchar(20).

Question 13

```
126
127 • ALTER TABLE training MODIFY COLUMN subject varchar(50);
128
129 • INSERT INTO
130     training (first_name, last_name, subject)
131 VALUES
132     ('Mariam', 'Hambrick', 'Accounting Principles'),
133     ('Mary', 'Moore', 'Spreadsheets');
134
135
```

Figure 20- Modified the table and adding 2 rows.

First I modify the subject varchar 20 to subject varchar 50 because the value of 20 is exceeding for the word ‘Accounting Principles’

This code will insert 2 new rows into the training table with the specified values for the first_name, last_name, and subject columns. Since the trainer_no column is set to auto-increment, it will automatically generate a new unique integer value for each new row inserted

Question 14

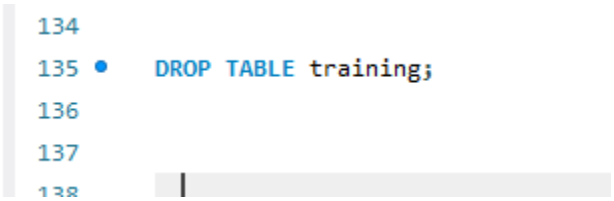


Figure 21- Delete table training.

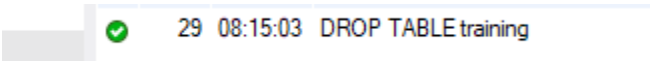


Figure 22- Validation that was deleted the table training

Using this code drops the table named “training from the database”, DROP TABLE + table name; this will delete the entire "training" table and all of its data.

Question 15

```
DROP TABLE IF EXISTS dept_manager;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE dept_manager (
  emp_no int NOT NULL,
  dept_no char(4) NOT NULL,
  from_date date NOT NULL,
  to_date date NOT NULL,
  PRIMARY KEY (emp_no,dept_no),
  KEY dept_no (dept_no),
  CONSTRAINT dept_manager_ibfk_1 FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
  CONSTRAINT dept_manager_ibfk_2 FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

Figure 23- Delete the table dept_manager and create a new table with 4 columns.

1. The degree of the tables is 4,because has 4 columns: emp_no, dept_no, from_date and to_date
2. Two columns are included in the primary key: emp_no and dept_no
3. One column is included in the foreign key named dept_manager_ibfk_2 and the name of the columnms:dept_no