Lebanese American University

Department of Computer Science and Mathematics

Course: Database Management Systems

CSC 375

Section: 14

Instructor: Dr. Khaleel Mershad

project phase: 4

project title: A Space Company Database

Name: Mohammad Adel Alsoheil (202206835)

- Please note that the corresponding SQL queries can be accessed through the files attached aside this report.
- The DML queries are not fully shown in this report, rather they are all found in the DML file attached.
- Some queries may not have an output such as views, or procedures, and functions.
- Find attached the E-R diagram.

Introduction:

This study presents the development of a database designed to effectively manage and organize a wide range of data pertaining to space exploration. The database encompasses diverse entities, including employees, scientists, spaceships, manufacturers, and various celestial objects such as planets, stars, galaxies, meteors, and black holes. Additionally, the paper explores the implementation of an Entity-Relationship (ER) model and a relational model to represent the database structure, providing detailed insights into the relationships and attributes of each entity. Furthermore, the following paper presents the implementation of a comprehensive database for a space company using MySQL Workbench. The database encompasses tables for various aspects of the company's operations, including departments, managers, employees, branches, applicants, trainers, astronauts, missions, manufacturers, services, spaceships, spacecraft locations, and more. The paper also includes basic and advanced queries to showcase the functionality of the database. Additionally, the paper discusses the use of SQL queries, views, triggers, and functions to address data management and integrity issues within the company's database. The solutions presented in the paper cover a range of tasks, such as identifying branches with all departments, managing workload for managers, ensuring data integrity, and creating procedures to track and manage data. Overall, the paper aims to demonstrate the effective use of SQL for database management within the context of a space company.

Description and database requirements:

(Entities, Attributes)

- A Space company needs a database to store data and information on its different types of employees, managers, and the discoveries they reach. The company has branches scattered around the world with various departments belonging to it. The company's employees are trainers, scientists, active and passive astronauts, where each employee is required to have an ID, name, age, phone number (could be multiple), salary, start-date, languages spoken, and work schedule. Each passive astronaut must have a join date and estimate activity date also they must have a training adaptability level as a percentage since passive astronauts are trained by at most one trainer. The database should save data about trainers too, such as achievements, the achievements include university degrees, certificates, and other extracurricular activities. After finishing their training, each passive astronaut can become an active astronaut which is supposed to start space flights and apply missions, therefore the database should start saving their number of flights, number of tackled missions and their estimate return. Also, the former missions are set up by scientists which are hired based on their discoveries and achievements which include university degrees, certificates, and research studies, the missions include the mission's name, status, date of foundation, and its description. Furthermore, Active Astronauts can fly spaceships that features a name, ID, type, release date, Activity status, and the number of flight hours, from certain stations (name, capacity of spaceships, coordinates) where their location can be tracked, the location should show the current location and the previous locations of the spaceships, the location includes ID, coordinates, and the type of location (ex: near a planet, on a moon etc..).Worth mentioning, that every spaceship launched can be launched again in a different station. Moreover, the spaceships are built and maintained by manufacturers which are separate companies, the manufacturers have a name, ID, rank, Activity status, and previous spaceships built, these manufacturers can perform services, each service (name, type ex: oil change, description) can be performed by one manufacturer. On the other hand, the database contains managers who can fire all kinds of employees on a specific date, the managers also can hire new applicants. Applicants should provide their names, age, job records,

nationality, phone number and set of achievements such as: university degrees, certificates, and other activities also they are given a unique ID once they apply. To add, each manager (ID, name, salary, phone number (multivalued), experience level), is responsible for managing a single department that includes different employees that can't work together in the same department. The database should also save the date the manager started managing the department. A department is known for its DID, name, budget, capacity, and description. Each department is branched to various branches around the world that include a name, contacts info such as phone and email, coordinates, and services offered. Those branches are moderated by branch-moderators. Each branch moderator can moderate at most one branch where the database stores their name, age, ID, executive rank (ex: CEO, CTO), salary, date of appointment, duration which is derived from their age and date of appointment.

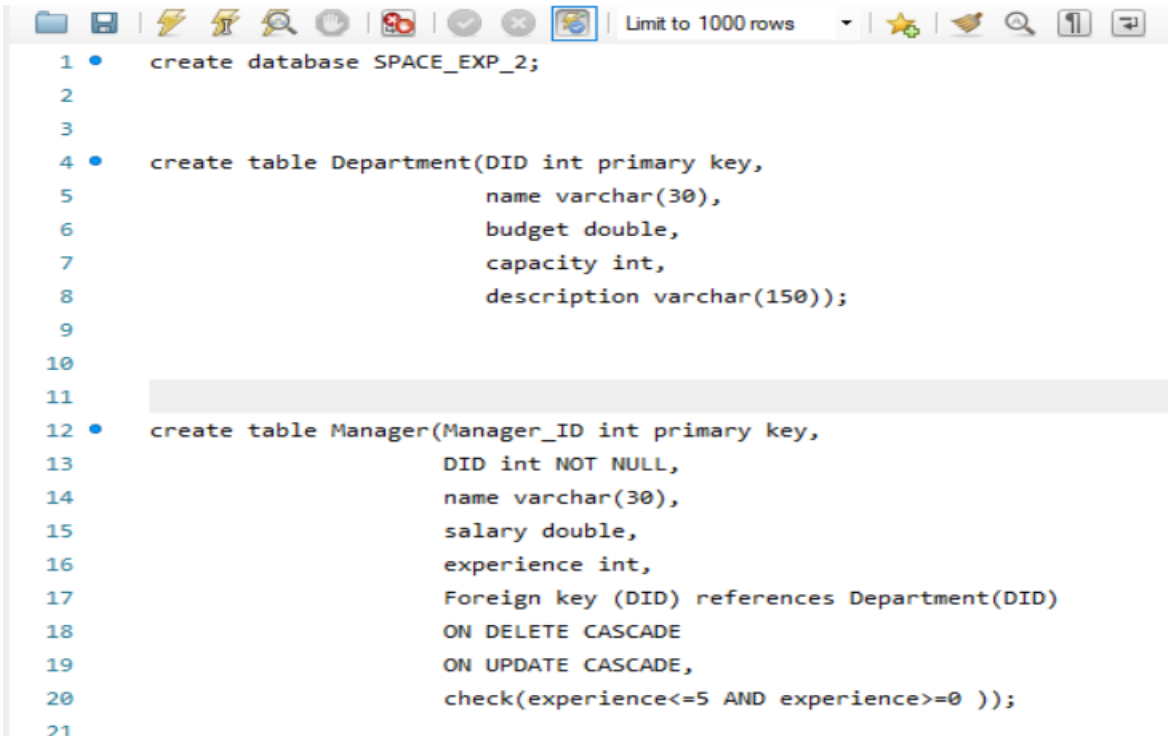
The company also stores specific data for different space objects discovered by active astronauts on a specific date. In this regard, data about planets, stars, meteors, galaxies and blackholes are stored in the database. Each planet orbits one star, planets can store data about their name, radius, mass, temperature, distance from earth, and the number of moons. Whereas stars have a name, mass, type, temperature, and distance away from earth. The latter relationship belongs to a certain galaxy in the universe. Galaxies have data for their names, size, morphology, and velocity. Furthermore, planets can be hit by a meteor (name, mass, velocity, direction). On the other side, blackholes attract stars, multiple blackholes can attract a single star at once. The database saves the name, mass, and the distance away from earth for every blackhole.

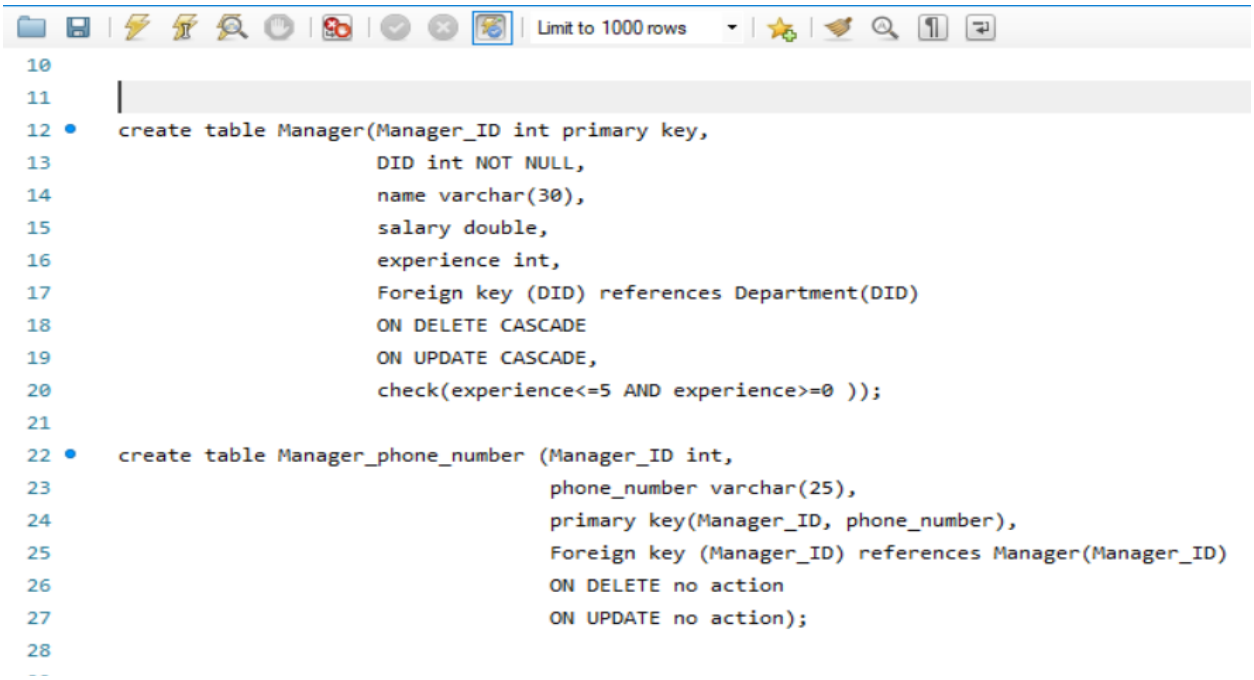
Data Implementation (Derived from Relational Model):

After creating the relational model, the first step was to create the actual database for the following model, I used MYSQL WORKBENCH for the implementation. First, I initialized the

tables using DDL SQL, therefore the following tables and code correspond for the actual tables derived from the relational model:

The first DDL command was: CREATE DATABASE SPACE_EXP_2; this command creates a new database for the project so that we can insert tables.





The first two tables are Department and Manager, the department has DID as primary key, and Department has a one-one relation with Manager, so Manager includes a NOT NULL foreign key corresponding to Department which is DID. A manager has an experience that ranges between 0 and 5, it is demonstrated by a check.

The table Manger_phone_number is a multivalued attribute for Manager therefore it has its separate table.

```
create table Employee(employee_ID int primary key,
                      DID int NOT NULL,
                      Manager_ID int,
                      name varchar(30),
                      age int,
                      salary double,
                      start_date date,
                      work_schedule varchar(100),
                      Foreign key (DID) references Department(DID)
                      ON DELETE CASCADE
                      ON UPDATE CASCADE,
                      Foreign key (Manager_ID) references Manager(Manager_ID)
                      ON DELETE CASCADE
                      ON UPDATE CASCADE,
                      check (age >= 18));
```

This table is the Employee table, it includes different employees in the company, later the ISA relation between some employees and the Table employee will be shown, a check constraint was used to ensure an employee must be older than 17 years old. Also, it contains foreign keys for the Department (one-many) and Manger (one-many) tables. Later in the DML part, I sat an imaginary range for each employee in the table.

```
45
46 •  create table Employee_languages (employee_ID int,
47                                    language varchar(20),
48                                    PRIMARY KEY (employee_ID, language),
49                                    Foreign key (employee_ID) references Employee(employee_ID)
50                                    ON DELETE no action
51                                    ON UPDATE no action);
52
53 •  create table Employee_phone_number (employee_ID int,
54                                        phone_number varchar(25),
55                                        PRIMARY KEY (employee_ID, phone_number),
56                                        Foreign key (employee_ID) references Employee(employee_ID)
57                                        ON DELETE no action
58                                        ON UPDATE no action);
59
60
```

These are multivalued attributes for employee table represented as separate tables.

```
61 •  create table Branch_Moderator(Branch_Moderator_ID int primary key,
62                                  name varchar(20),
63                                  age int,
64                                    executive_rank char(3),
65                                  salary double,
66                                  date_of_appointment date,
67                                  check(age>=25)
68                                  );
69
70 •  create table Branch(coordinates_X decimal(10,8),
71                        coordinates_Y decimal(10,8),
72                        Branch_Moderator_ID int NOT NULL,
73                        location varchar(40),
74                        services_offere varchar(100),
75                        phone_number varchar(25),
76                        email varchar(55),
77                        unique(coordinates_X,coordinates_Y),
78                        foreign key (Branch_Moderator_ID) references Branch_Moderator(Branch_Moderator_ID)
79                        ON DELETE cascade
80                        ON UPDATE cascade);
81
```

Now, every Department can be branched to different branches across the world, therefore each branch is identified by its coordinates (primary key). Moreover, the branch moderator moderates a single branch (one-one) so Branch_Modertator_ID is a FK in Branch.

```
82 •  create table Branched_to( DID int,
83                             coordinates_X decimal(10,8),
84                             coordinates_Y decimal(10,8),
85                             primary key(DID,coordinates_X, coordinates_y),
86                             foreign key(DID) references Department(DID)
87                             ON DELETE NO ACTION
88                             ON UPDATE CASCADE,
89                             foreign key(coordinates_X,coordinates_Y) references Branch(coordinates_X,coordinates_Y)
90                             ON DELETE NO ACTION
91                             ON UPDATE CASCADE
92                             );
93
```

This is the branched to relation represented as a single table, since branch and departments have a (M-M) relationship, the PK of both tables are taken.

```
•   create table Applicant(applicant_ID int primary key,
                           Manager_ID int,
                           name varchar(30),
                           age int,
                           phone_number varchar(25),
                           job_records varchar(50),
                           university_degree char(12),
                           certificates varchar(40),
                           date_of_application date,
                           other_activities char(20),
                           Foreign key (Manager_ID) references Manager(Manager_ID)
                           ON DELETE NO ACTION
                           ON UPDATE NO ACTION,
                           check(age>=18)
                           );
```

The applicant table is a table ensuring only adult people apply. Also, it is referred to the Manager table through a FK, since a manager must supervise the application of the applicant.

```
111 •  create table Applicant_nationality(applicant_ID int,
112                                       nationality char(20),
113                                       primary key(applicant_ID, nationality),
114                                       foreign key (applicant_ID) references Applicant(applicant_ID)
115                                       ON DELETE NO ACTION
116                                       ON UPDATE NO ACTION);
117
```

This is a multivalued attribute for the applicant table.

```
118 •   create table Trainer(Trainer_ID int primary key,
119                         training_material varchar(50),
120                         university_degree char(12),
121                         certificates varchar(40),
122                         other_activities char(20),
123                         foreign key (Trainer_ID) references Employee(employee_ID)
124                         ON delete cascade
125                         ON update cascade);
126
```

Now this is the first table that has an ISA relation with the employee table, it's the trainer table that is responsible for training passive astronauts below.

```
127 •   create table Passive_Astronaut(Passive_Astronaut_ID int primary key,
128                                   Trainer_ID int NOT NULL,
129                                   adaptability Decimal(5,2),
130                                   join_date date,
131                                   estimate_activity_date date,
132                                   foreign key (Trainer_ID) references Trainer(Trainer_ID)
133                                   ON delete no action
134                                   ON update cascade,
135                                   foreign key (Passive_Astronaut_ID) references Employee(employee_ID)
136                                   ON delete cascade
137                                   ON update cascade,
138                                   check(adaptability BETWEEN 0 AND 99));
139
```

Since a trainer has a (one-M) relationship with the passive astronauts a FK for the trainer is represented in the Passive Astronaut table, also a FK is initiated for the employee table (ISA relation). Moreover, adaptability is measured as a percentage therefore it should be between 0 and 99.

```
141 •   create table Active_Astronaut(Active_Astronaut_ID int primary key,
142                                 Passive_Astronaut_ID int NOT NULL,
143                                 number_of_flight int,
144                                 number_of_tackled_mission int,
145                                 estimated_return date,
146                                 date_of_Activity date,
147                                 foreign key (Passive_Astronaut_ID) references Passive_Astronaut(Passive_Astronaut_ID)
148                                 on delete no action
149                                 on update cascade,
150                                 foreign key (Active_Astronaut_ID) references Employee(employee_ID)
151                                 ON delete cascade
152                                 ON update cascade
153                                 );
154
155
```

The Active Astronaut table has two FKs which are passive Astronaut (a passive astronaut can transform to an active astronaut) and the ISA relationship with employee.

```
157 •   create table Scientist(Scientist_ID int primary key,
158                            discoveries varchar(50),
159                            university_degree char(12),
160                            certificates varchar(40),
161                            research_studies varchar(100),
162                            foreign key (Scientist_ID) references Employee(employee_ID)
163                            ON delete cascade
164                            ON update cascade
165                            );
166
```

The Scientist table also has an ISA relationship with employee.

```
167 •   create table Mission(mission_name varchar(40) primary key,
168                         status boolean,
169                         date_of_foundation date,
170                         description varchar(70));
171
172 •   create table sets_up(mission_name varchar(40),
173                         Scientist_ID int,
174                         primary key(mission_name,Scientist_ID),
175                         foreign key (mission_name) references Mission(mission_name)
176                         on delete cascade
177                         on update cascade,
178                         foreign key (Scientist_ID) references Scientist(Scientist_ID)
179                         on delete no action
180                         on update no action
181                         );
182
```

these are two tables one corresponds to the missions that are initiated By Scientists, the other implements the (M-M) relationship between Scientists and missions by considering the PKs of both tables.

```
183 •   create table Appy(Active_Astronaut_ID int,
184                      mission_name varchar(40),
185                      date_of_application date,
186                      primary key(Active_Astronaut_ID, mission_name),
187                      foreign key(Active_Astronaut_ID) references Active_Astronaut(Active_Astronaut_ID)
188                      on delete cascade
189                      on update cascade,
190                      foreign key(mission_name) references Mission(mission_name)
191                      on delete cascade
192                      on update cascade
193                      );
194
```

This is the apply table that links the (M-M) relationship of Active Astronauts and

missions. In addition to that, it adds the date when the actual mission was applied by the astronaut.

```
195 •  create table Manufacturer(manufacturer_ID int primary key,
196                              name varchar(30),
197                              Rank_ char(20),
198                              isActive boolean,
199                              num_of_previous_ships_built int);
200
201 •  create table Services(name_of_service char(30),
202                          manufacturer_ID int,
203                          type char(10),
204                          description varchar(70),
205                          primary key(name_of_service,manufacturer_ID),
206                          foreign key (manufacturer_ID) references Manufacturer(manufacturer_ID)
207                          on delete no action
208                          on update cascade);
209
```

Now we have two tables: Manufacturer and Services linked by an identifying relationship where the PK of manufacture is the ID, yet the primary key of services is a combination between the manufacturer ID and the specific name of the service. Worth mentioning, the manufacturer is not a employee in the company (no ISA relationship).

```
210 •  create table spacecraft_location (spacecraft_location_ID int primary key,
211                                     type char(10),
212                                     coordinate_X decimal(10,10) unique,
213                                     coordinate_Y decimal(10,10) unique
214                                     );
215 •  create table SpaceShip(spaceShip_ID int primary key,
216                          spacecraft_location_ID int,
217                          name varchar(30),
218                          type char(20),
219                          release_date date,
220                          isActive boolean,
221                          num_of_flight_hours int,
222                          foreign key (spacecraft_location_ID) references spacecraft_location(spacecraft_location_ID)
223                          on delete cascade
224                          on update cascade);
225
```

The table spaceship corresponds to the different spaceships that can be used by Active Astronauts (M-M) and is built by manufacturers (M-M). While the spacecraft location has 2 usages which are: tracking the exact location of a spacecraft through its coordinates (1-1), and WAS_IN relation that will be elaborated down below keeping track of all previous locations of a spacecraft.

```
226 •   create table Was_In (spaceShip_ID int,
227                         spacecraft_location_ID int,
228                         primary key(spaceShip_ID, spacecraft_location_ID),
229                         foreign key(spaceShip_ID) references SpaceShip(spaceShip_ID)
230                         on delete no action
231                         on update no action,
232                         foreign key(spacecraft_location_ID) references spacecraft_location(spacecraft_location_ID)
233                         on delete cascade
234                         on update cascade
235                         );
236
```

This is the WAS_IN table that keeps track of all the previous locations of a spacecraft.

```
•   create table Builds(spaceShip_ID int,
                        manufacturer_ID int,
                        primary key(spaceShip_ID,manufacturer_ID),
                        foreign key(spaceShip_ID) references SpaceShip(spaceShip_ID)
                        on delete cascade
                        on update cascade,
                        foreign key (manufacturer_ID) references Manufacturer(manufacturer_ID)
                        on delete no action
                        on update cascade
                        );
```

The builds table that links the spaceships and manufacturer, it was represented by a single table since it is a (M-M) relationship.

```sql
create table station(station_name varchar(25) primary key,
                    capacity int,
                    coordinat_X decimal(10,8),
                    coordinat_Y decimal(10,8),
                    check(capacity>=0));


create table Fly(spaceShip_ID int,
                 Active_Astronaut_ID int,
                 station_name varchar(25),
                 primary key(spaceShip_ID,Active_Astronaut_ID,station_name),
                 foreign key(spaceShip_ID) references SpaceShip(spaceShip_ID)
                 on delete no action
                 on update cascade,
                 foreign key(Active_Astronaut_ID) references Active_Astronaut(Active_Astronaut_ID)
                 on delete no action
                 on update no action,
                 foreign key (station_name) references station(station_name)
                 on delete cascade
                 on update cascade
                 );
```

Woth mentioning here that fly is a ternary relationship between station, spaceship, and active astronauts so it's (M-M-M). station is identified by the coordinates X and Y and of course its capacity is always positive.

```sql

create table Planets(planet_name varchar(25) primary key,
                     radius float,
                     mass float,
                     temperature float,
                     num_of_moons int,
                     distance_from_earth double,
                     check(distance_from_earth>0));

create table star(star_name varchar(25) primary key,
                  mass float,
                  type char(20),
                  temperature float,
                  distance_from_earth double,
                  check(distance_from_earth>0)
                  );

```

```
create table orbits(planet_name varchar(25) primary key,
                    star_name varchar(25),
                    foreign key (planet_name) references Planets(planet_name)
                    on delete cascade
                    on update cascade,
                    foreign key (star_name) references star(star_name)
                    on delete cascade
                    on update cascade);
```

Orbits is a table that connects both planets and star, worth noting there is a (one-M) relationship between planets and star, but orbits is needed to satisfy the aggregation later on.

```
294
295 •   create table Meteors(meteor_name varchar(25) primary key,
296                         planet_name varchar(25),
297                         mass float,
298                         velocity double,
299                         direction char(4),
300                         foreign key (planet_name) references Planets(planet_name)
301                         on delete cascade
302                         on update cascade);
303
304
```

The meteors table has a (M-one) relationship with planets, therefore FK planet_name appears in Meteors.

```
304
305 •   create table Blackholes(blackhole_name varchar(25) primary key,
306                            mass float,
307                            distance_from_earth double);
308
309 •   create table attracts(blackhole_name varchar(25),
310                          star_name varchar(25),
311                          primary key(blackhole_name,star_name),
312                          foreign key (star_name) references star(star_name)
313                          on delete cascade
314                          on update cascade,
315                          foreign key (blackhole_name) references Blackholes(blackhole_name)
316                          on delete cascade
317                          on update cascade);
318
```

The blackhole is a separate table where the relation attracts (M-M) joins blackholes with stars.

```
318
319 •  create table Galaxy(galaxy_name varchar(25) primary key,
320                        size double,
321                        morphology varchar(25),
322                        velocity double,
323                        check(size>0));
324
325
326 •  create table Belong_to(planet_name varchar(25),
327                           galaxy_name varchar(25),
328                           primary key(planet_name,galaxy_name),
329                           foreign key (planet_name) references Planets(planet_name)
330                           on delete cascade
331                           on update cascade,
332                           foreign key (galaxy_name) references Galaxy(galaxy_name)
333                           on delete cascade
334                           on update cascade);
335
```

Also, here galaxy is a separate table. However, belong to corresponds to aggregation orbits, since a galaxy also contains the orbiting of the planet around a star.

```
337 • create table Discovers(galaxy_name varchar(25),
338                         Active_Astronaut_ID int,
339                         since date,
340                         primary key(galaxy_name,Active_Astronaut_ID),
341                         foreign key (galaxy_name) references Galaxy(galaxy_name)
342                         on delete cascade
343                         on update cascade,
344                         foreign key(Active_Astronaut_ID) references Active_Astronaut(Active_Astronaut_ID)
345                         on delete no action
346                         on update no action);
347
```

This table links an active astronaut with the galaxy they are discovering in.

```
alter table Manager_phone_number
add constraint phone_num unique(phone_number);

alter table Branch
add constraint phone_num unique(phone_number);

ALTER TABLE spacecraft_location
MODIFY COLUMN coordinate_X decimal(11,8);

ALTER TABLE spacecraft_location
MODIFY COLUMN coordinate_Y decimal(11,8);


ALTER TABLE branch
CHANGE COLUMN services_offere services_offered VARCHAR(100);
```

Finally, these are some alters that I changed to ensure a well-structured DBMS. Off course every manager has a unique phone number, so I added a new constraint to apply that.

Basic Queries:

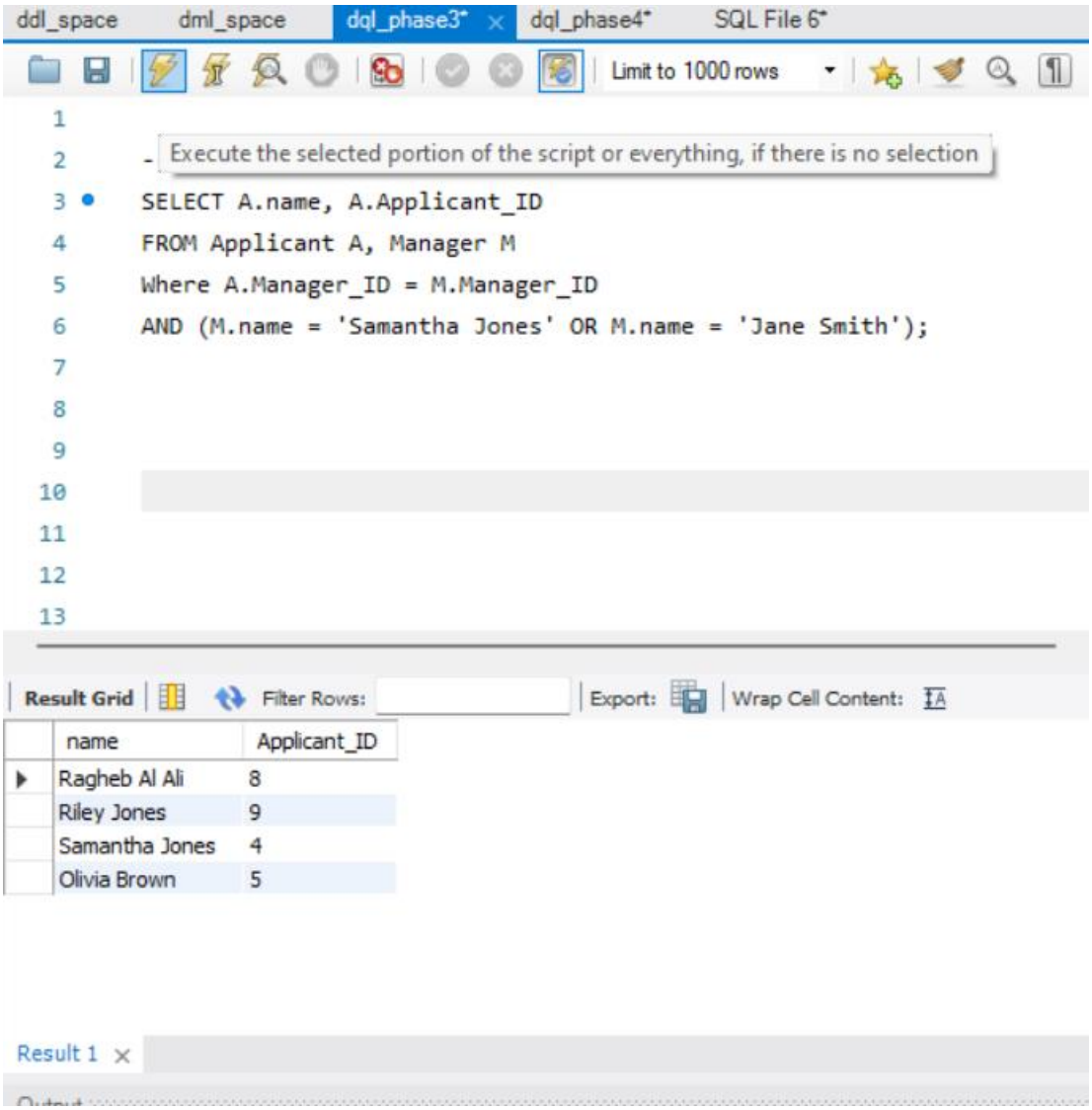
The following queries are related to basic queries, phase 3 of the project.

Phase 3)

Q1- Two managers Samantha Jones and Jane Smith have reported the possibility of hiring some applicants. Using cartesian product find the names and Ids of all

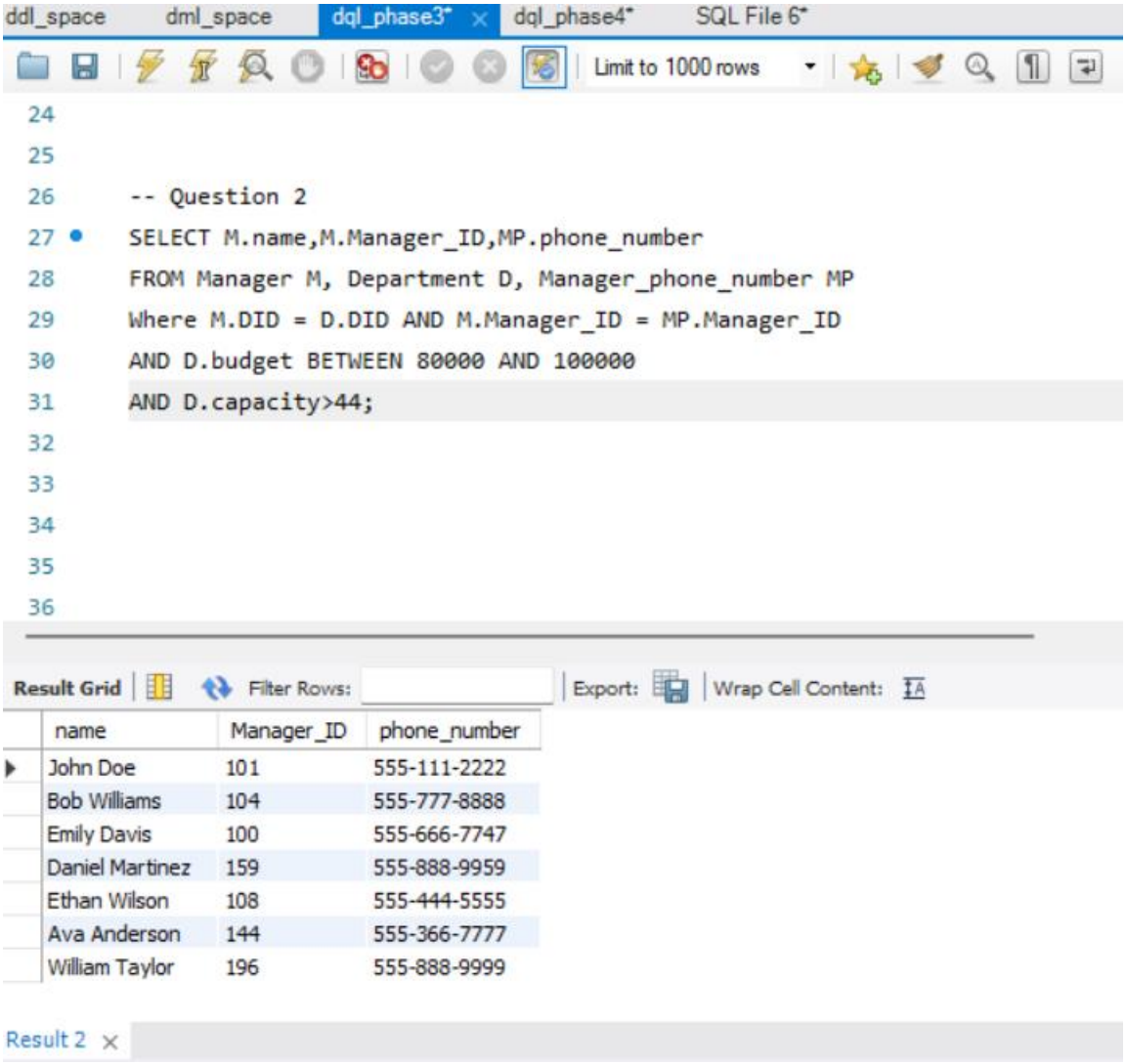
applicants that are managed by (may be hired by) by managers whose name is Samantha Jones or, Jane Smith.

Solution: use cartesian join in the where statement and identify the specific names.



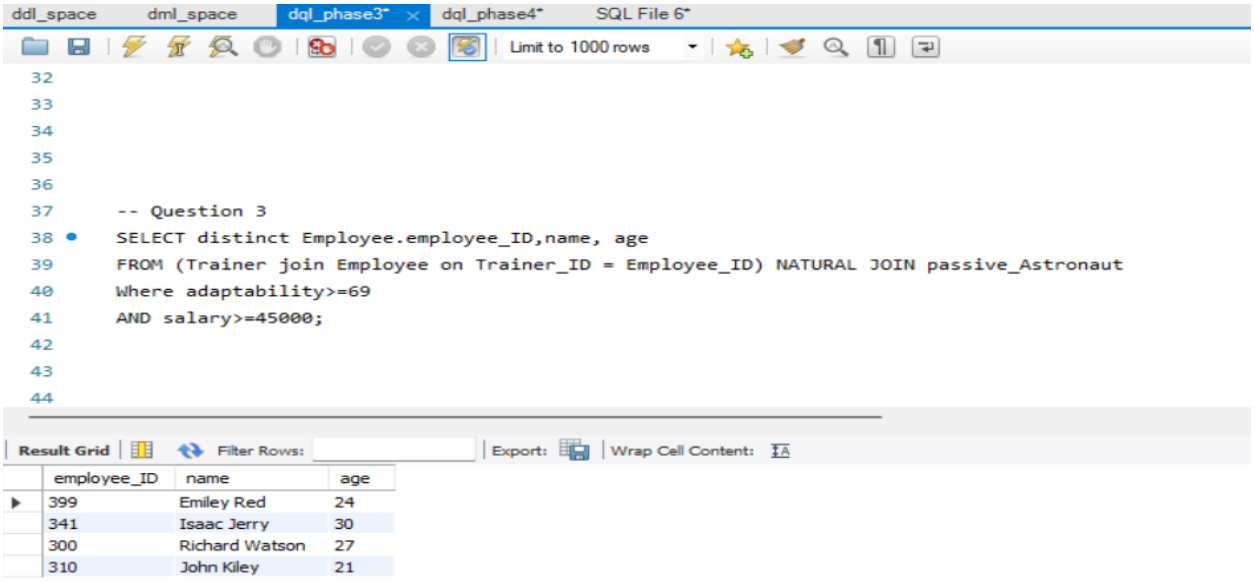
Q2- The company wants to make some changes regarding certain departments. However, these changes are limited to a certain budget. Using cartesian product find the names, IDs, and phone number of all managers who manage departments that have a budget between 80,000 and 100,00$ and has capacity greater than 44.

Solution: use cartesian join in the where statement and identify the inclusive range using between.



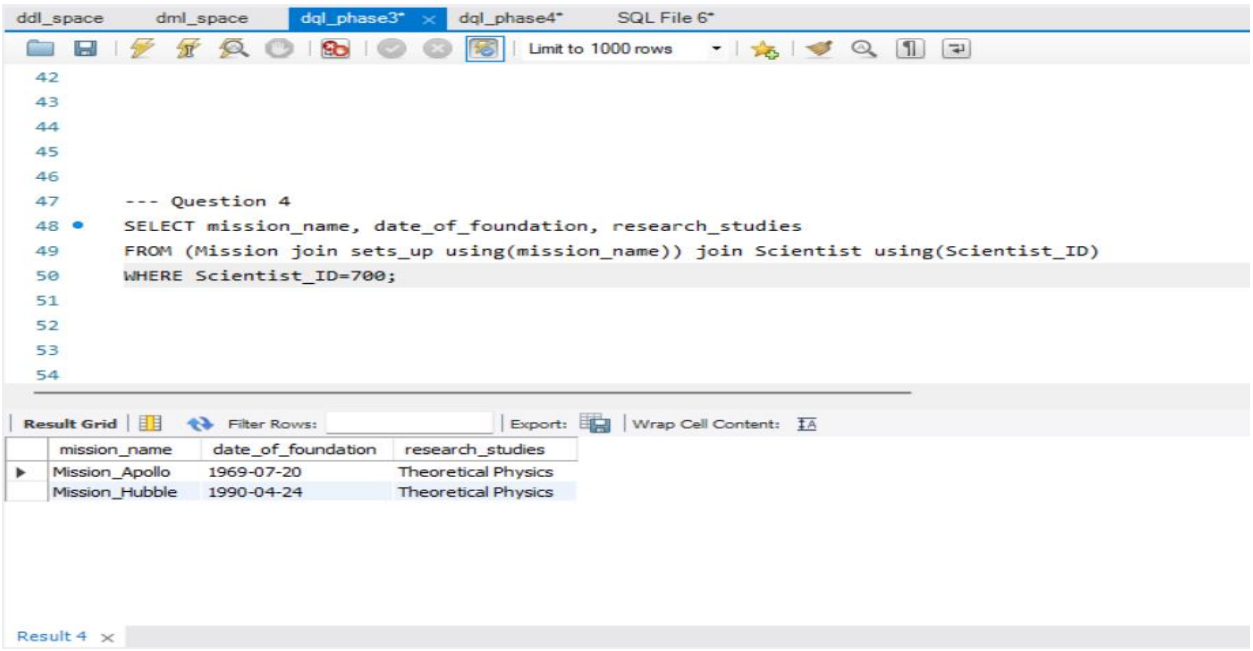
Q3-The company wants to give a raise to the trainers that have the greatest influence on the astronauts but only if they have a certain salary. Using Natural join Get the names of all trainers having a salary greater than or equal 45000, who train at least 1 astronaut who has an adaptability level greater than or equal 69%.

Solution: natural join is used with passive astronaut after joining employee with trainer since there is an ISA relationship (get the name, age), then adaptability is and salary are compared



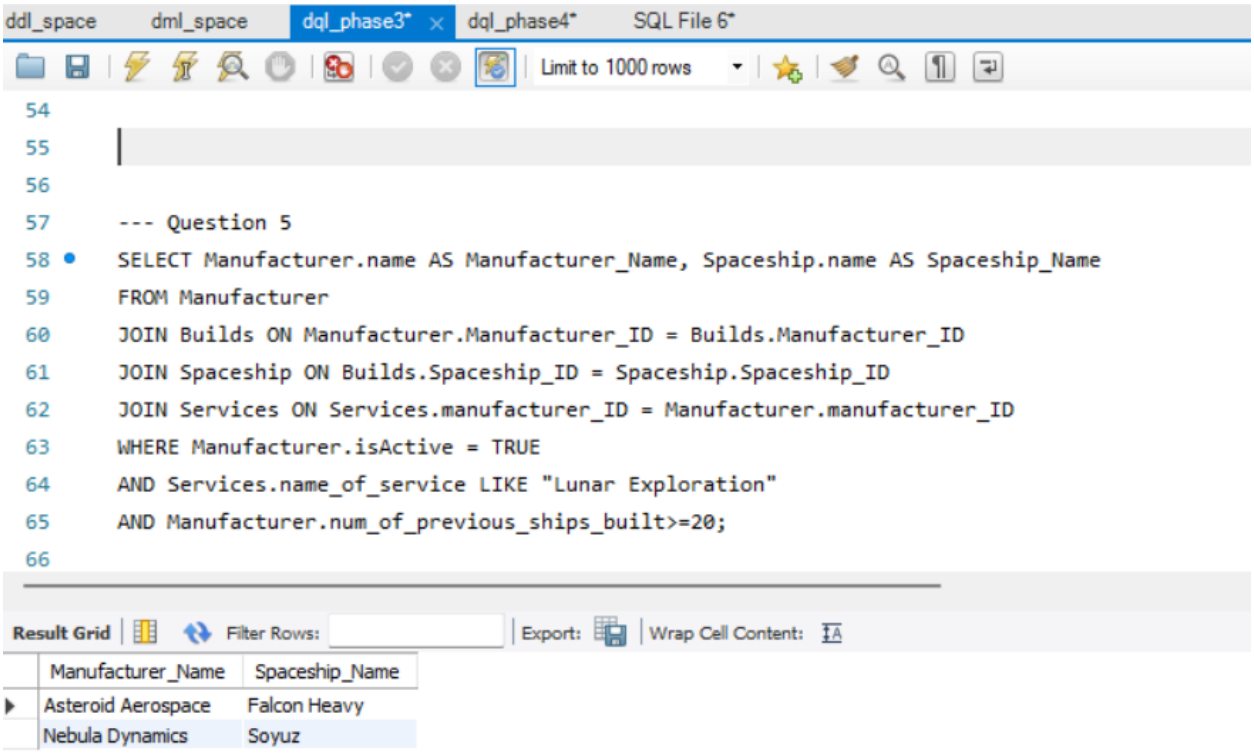
Q4- There is an active certain mission that has burdens and is set by a scientist whose ID is 700. Using Theta join (Using) Get the names of all the missions which are currently under execution and set by a scientist whose ID is 700. Make sure also to include in your result the date of foundation of the following mission and the scientist research studies.

Solution: use theta join (using) between mission and sets up, also join the result with scientists so that we can get the IDs, finally select the wanted attributes.



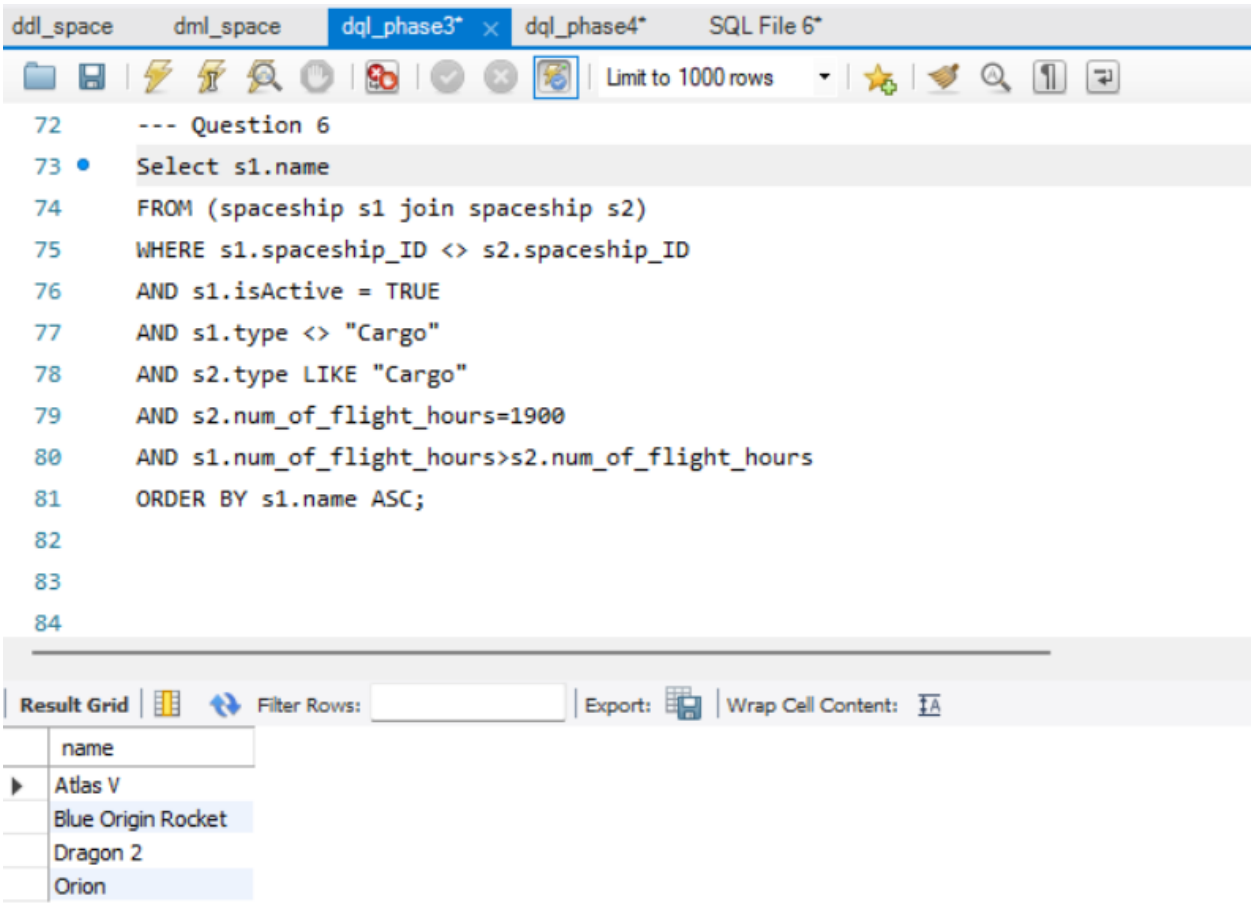
Q-5 The company wants to increase its production for spaceships related to Lunar exploration so it must contact its corresponding experienced manufacturers. Get the name of spaceships and their manufacturer which are still active and perform a service called “Lunar Exploration”. Make sure also to select only the manufacturer that built more than 20 spaceships.

Solution: First, we want to join (ON) the needed tables, so we join manufacturers with builds, spaceship, and services, now we can compare the activity of the manufacturer, and number of ships built from the manufacturer table, check the name of the service from the services table. Finally, because we want to distinguish between the name of the manufacturer and the name of the spaceship, we use the AS keyword.



Q-6 The company wants to gather statistics about Cargo spaceships. Find the names of all spaceships that are currently active and are not “Cargo” and have flight hours greater than or equal to a spaceship whose type is “Cargo” and has 1900 flight hours. (make sure to order the results by alphabetical order).

Solution: here we utilize the aspect of self-join because we are comparing tuples inside the same table. So, we name two instances of spaceships s1, s2 where s1<>s2 because we don’t want to compare the spaceship with itself. Then we put the description of s1 and s2 (Cargo, True …) to order the result alphabetically at the end.



Q-7 The company noticed that some scientists work every day but still does not have a decent income. Find the names, IDs, and the university degrees of all scientists with a salary greater than or equal to $500000 and having a work schedule of “Monday to Friday”, OR scientists with a salary less than $500000 but work “Everyday”.

Solution: create a table of each case and then merge the two tables using the union keyword, we must make sure that the same columns are present in both tables. We also join scientists with employee because there is an ISA.

ddl_space dml_space dql_phase3* dql_phase4* SQL File 6*

Limit to 1000 rows

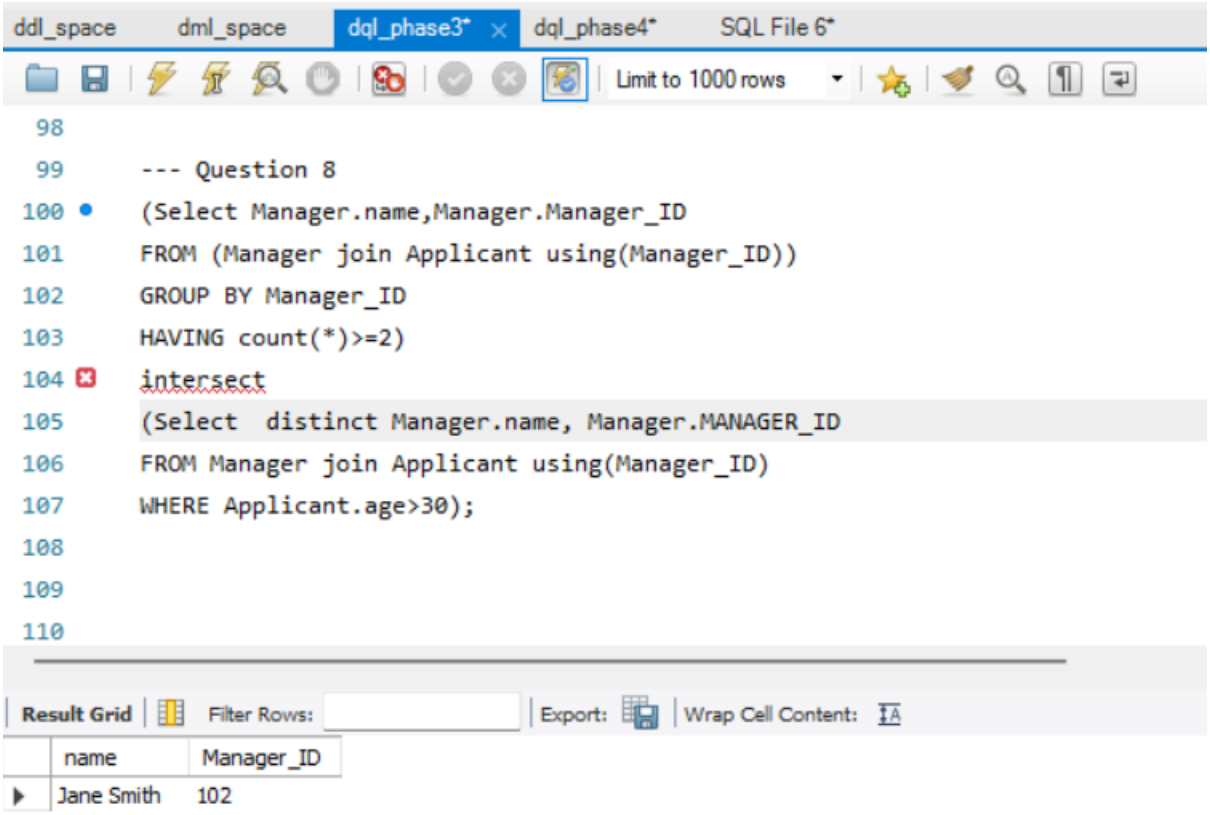
```
84
85
86    --- Question 7
87    (Select employee.employee_ID,employee.name,Scientist.university_degree
88    FROM (Scientist join employee on employee_ID=Scientist_ID)
89    WHERE employee.salary>=500000
90    AND  employee.work_schedule LIKE "Monday to Friday")
91    UNION
92    (Select employee.employee_ID,employee.name,Scientist.university_degree
93    FROM (Scientist join employee on employee_ID=Scientist_ID)
94    WHERE employee.salary<500000
95    AND  employee.work_schedule LIKE "Everyday");
96
```

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

employee_ID	name	university_degree
700	Hassa Aloosh	PhD
710	thomas yellow	BSc
719	Dina Menaam	MSc
709	khaled Abed AL Ali	BSc

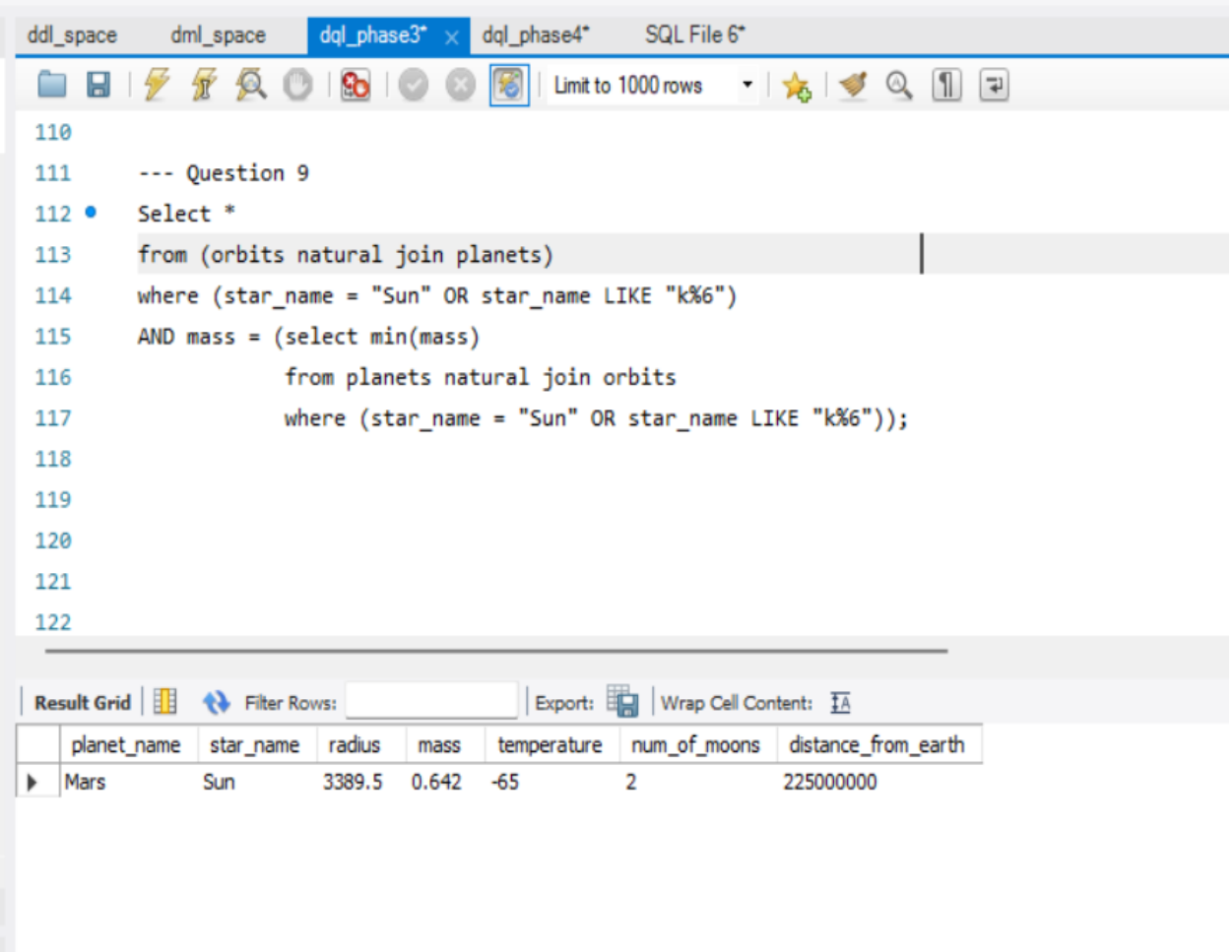
Q-8 The company wants to see if old applicants still apply to the company where it needs to compare their application probability between 1 in every 2 applicants Find the names of all managers who are managing at least 2 applicants hiring process, where at least one of the following applicants is older than 30 years.

Solution: here also we must create two separate and use the intersect operator between them. The first table gets the ID of Managers who manage at least 2 applicants using the group by and count (*), the second retrieves the IDs of managers who manage an applicant older than 29 years old. So, if the ID appears in both tables it belongs to the solution set.



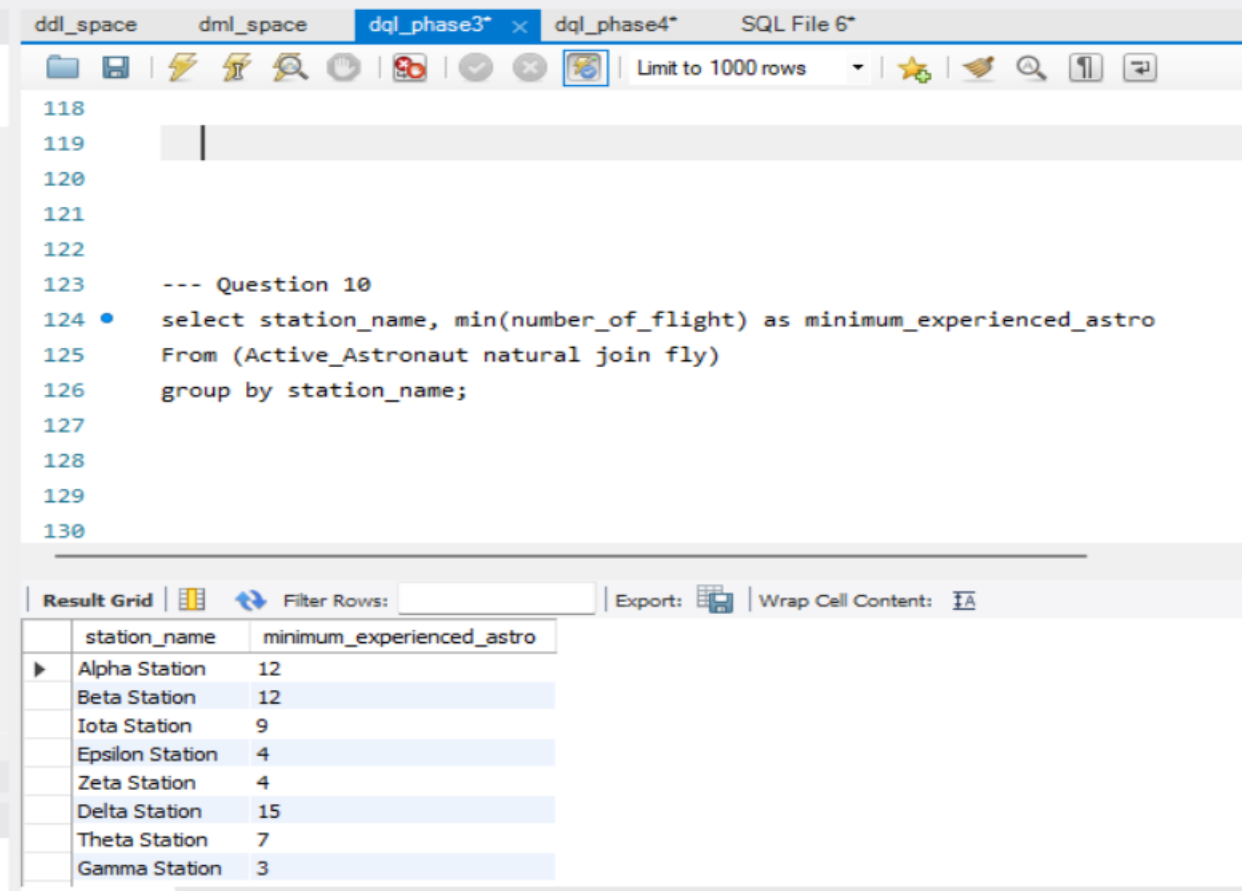
Q-9 while studying star orbiting and planets. An employee forgot the name of a certain star, luckily, she still remember its first and last character. Find the name of the planet with all its info and the star that the following planet belongs to, where this planet has the smallest mass and is orbiting either a star called “Sun” or a star that starts with K and ends with 6.

Solution: create a fixed table containing the minimum mass of planets that orbit either the sun or “k%6”, a star that starts with k and ends with 6, the go over the tuples in orbits and planets joined if one tuple’s mass is equal to the minimum mass. So, it belongs to the solution set.



Q-10 The company wants to see the least utilized station so that it can add more spaceships to its ground. From each station return the names of stations with the minimal flight hours, make sure to include the flight hours in your final table.

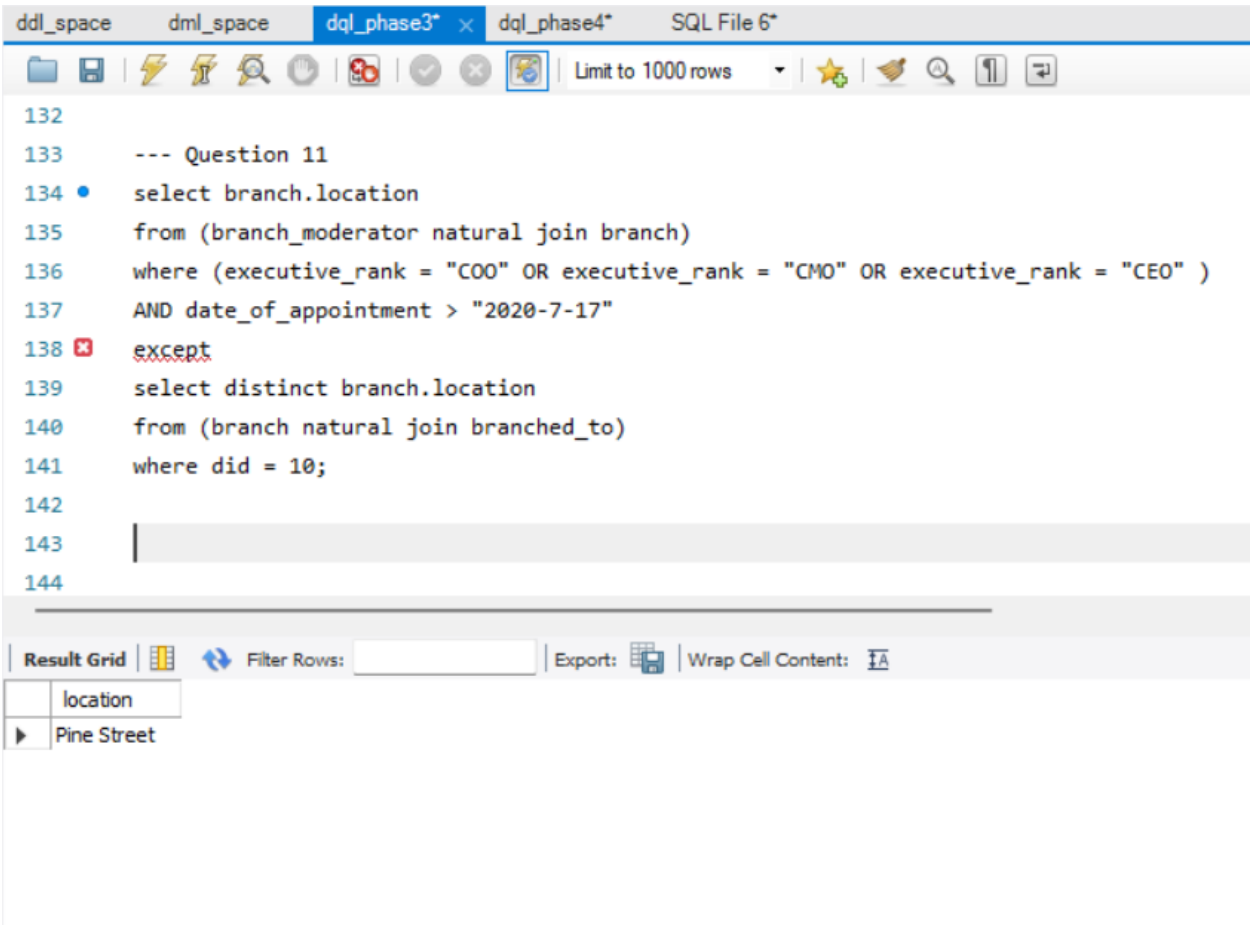
Solution: use group by with an aggregate function min() in the select statement.



Q-11 A study needs to be conducted on the variety of new executive ranks among departments and their locations. Find the location of branches whose branch moderators are either a “COO”, “CMO” or “CEO” and appointed after “2020-7-17”, yet these branches does not have a department whose DID is equal to 10.

Solution: The first calculated table contains moderators which are: “COO”, “CMO”, or “CEO” and appointed after 2020-7-17, a natural join is used between branch moderator and branch.

The second table only selects branches that contain a Department with DID = 1. The except keyword is used to deduct similar tuples from both tables.


ddl_space
dml_space
dql_phase3*
dql_phase4*
SQL File 6*
Limit to 1000 rows
132
133 --- Question 11
134 select branch.location
135 from (branch_moderator natural join branch)
136 where (executive_rank = "COO" OR executive_rank = "CMO" OR executive_rank = "CEO")
137 AND date_of_appointment > "2020-7-17"
138 except
139 select distinct branch.location
140 from (branch natural join branched_to)
141 where did = 10;
142
143
144
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
location
Pine Street

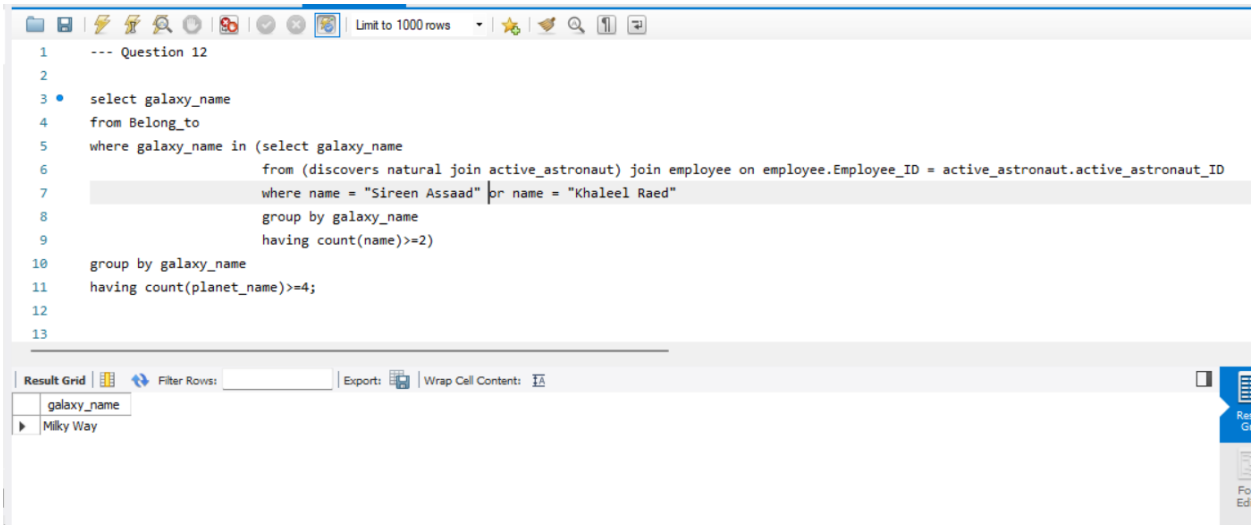
Advanced Queries:

The following queries are related to advanced queries, phase 4 of the project.

Phase 4)

Q-12. “Sireen Assaad” and “Khaleel Raed” are initiating new research about a certain galaxy. Using a set membership nested query, return the names of galaxies which are discovered by both “Sireen Assaad” and “Khaleel Raed” together, and has at least 4 planets in that galaxy (a galaxy can have more discoverers but we only care about these two names).

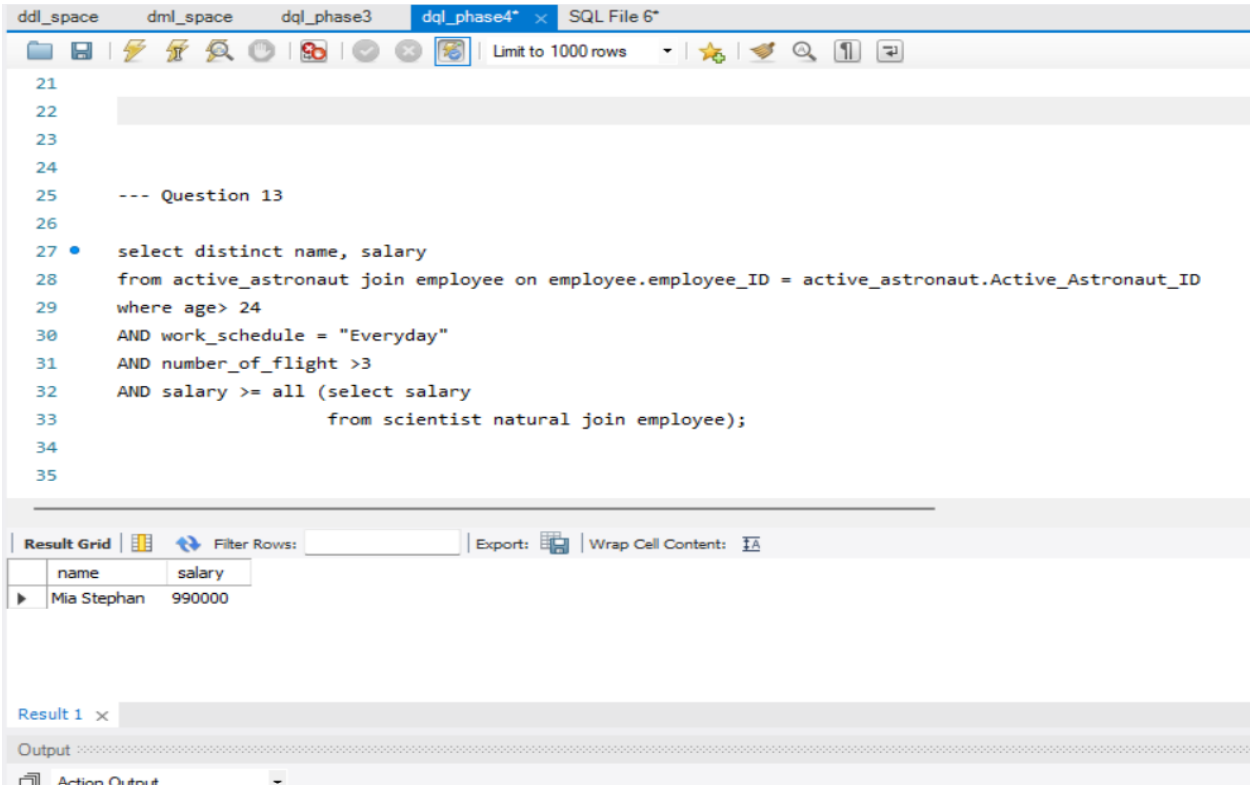
Solution: First, we calculate a fixed table which is the nested query done by joining discovers with active astronauts so that we can group the galaxy with the same name, and we want to keep only tuples that are discovered by two different astronauts, so count is greater than or equal two. Also, we ensure that Sireen and Khaleel are there by comparing the name. Last, we group by galaxy name ensure that a galaxy has more than four planets orbiting it.



Q-13 The space company wants to provide rewards for certain astronauts that discovered a well-discovered galaxy and are older than some scientists. Find the names and the corresponding salary of active astronauts whose salary is greater than all scientists and are older than 24 years old with an “Everyday” work schedule. Also, their number of flights must be greater than 3.

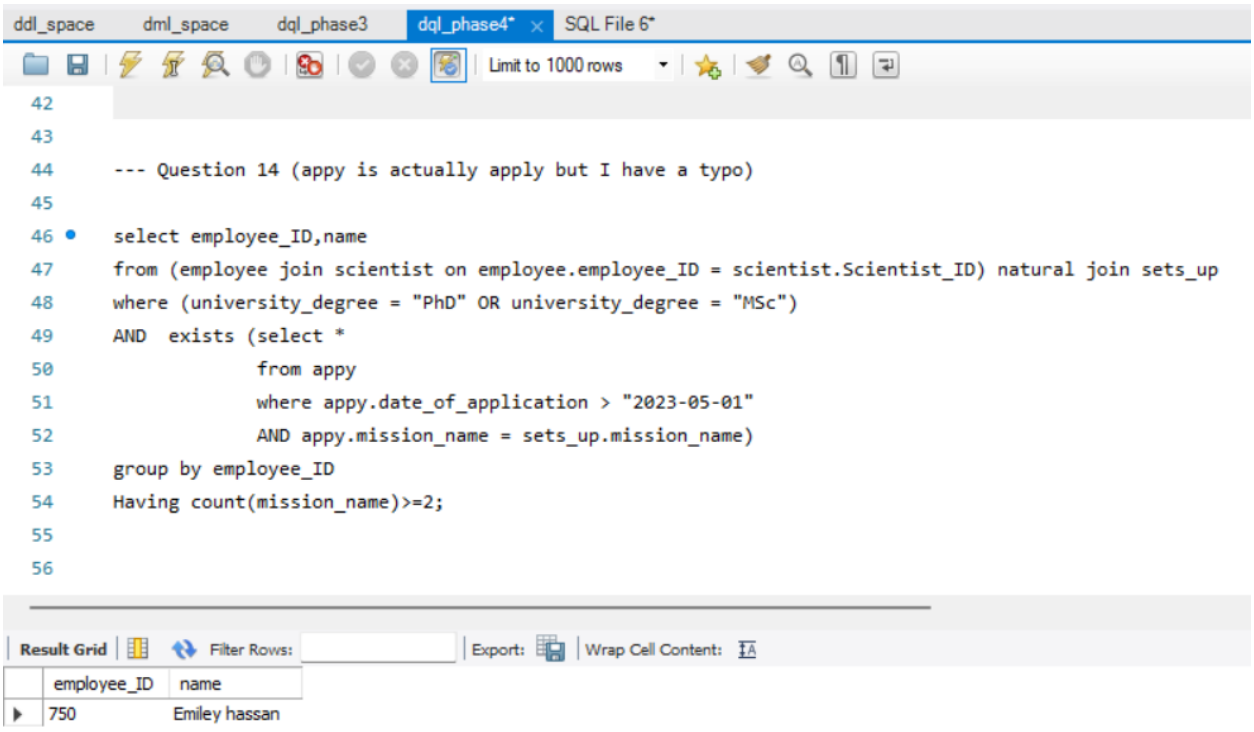
Solution: here we utilize the set comparison feature by using the “All” keyword, but first we have to consider the description of the astronauts, so we use comparisons in the where statements (age, work_schedule, ...), and then we check

that the salary is greater than or equal all salaries in a fixed table corresponding to scientists. We go a tuple at a time and compare it with all salaries in the fixed table.



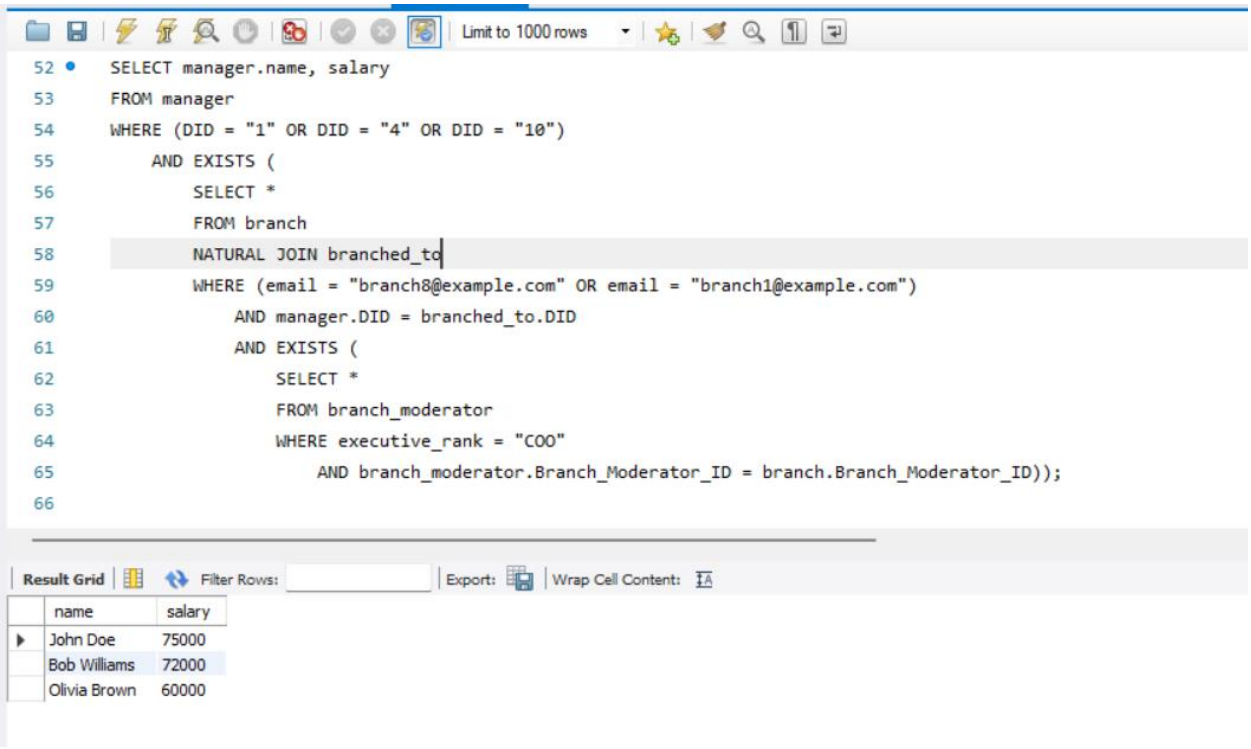
Q-14 The department of planning wants to see if missions sat up by Scientists are being applied or not. Find the name, IDs of scientists that hold either a “PhD” or “MSc” and sat up at least 2 missions, where at least one of these missions is being applied by an active astronaut after the month of MAY 2023.

Solution: here we first want to consider grouping the scientists based on their ID after joining the scientists table with the sets up relationship. So, that we can ensure only scientists that set up 2 mission or more. Then, in the where statements we pick only scientists with PhDs and MSc, and most importantly we check dynamically the existence of every scientist satisfying the description in a table through a nested query utilizing the exists keyword by a correlation between the apply and scientist tables. If the table is non-empty the tuple containing the scientist is added to the solution set.



Q-15 it appears that the company is suspecting a certain “COO” is always associated with departments whose DID is 1,4 , 5, 10. Find the name and the corresponding salary of managers who manage a department whose DID is 1,4, or 10, branched to a branch whose email is either “branch1@example.com” or “branch8@example.com”, where the branch moderator there is a “COO”.

Solution: The Primary Table is manager where the Conditions are Managers selection based on their department IDs (DID) being either "1", "4", or "10". The query uses EXISTS subqueries to validate the presence of specific conditions in related tables: Checks if the manager's department matches the DID in the branched_to table, associated with branches branch8@example.com or branch1@example.com. Confirms that the branch associated with the manager's department has a moderator with executive rank of "COO".



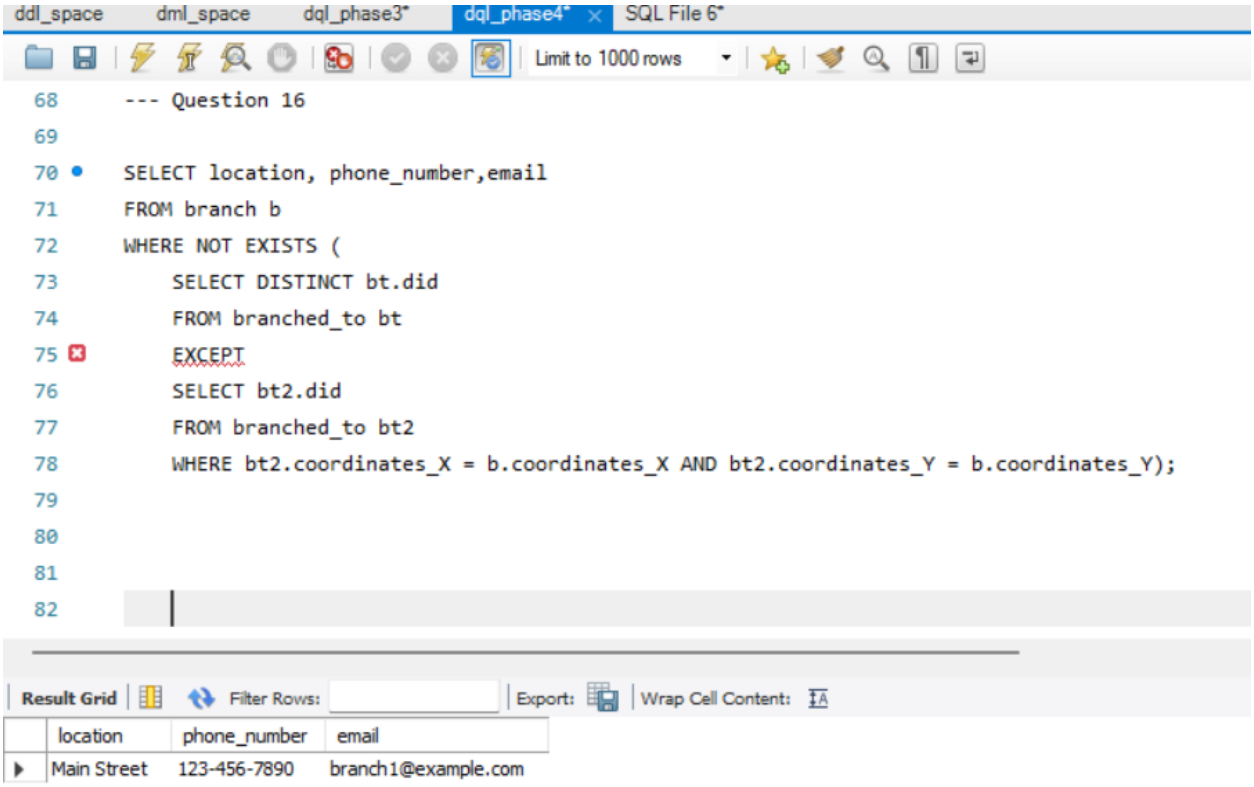
```
SELECT manager.name, salary
FROM manager
WHERE (DID = "1" OR DID = "4" OR DID = "10")
    AND EXISTS (
        SELECT *
        FROM branch
        NATURAL JOIN branched_to
        WHERE (email = "branch8@example.com" OR email = "branch1@example.com")
            AND manager.DID = branched_to.DID
            AND EXISTS (
                SELECT *
                FROM branch_moderator
                WHERE executive_rank = "COO"
                    AND branch_moderator.Branch_Moderator_ID = branch.Branch_Moderator_ID));
```

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

name	salary
John Doe	75000
Bob Williams	72000
Olivia Brown	60000

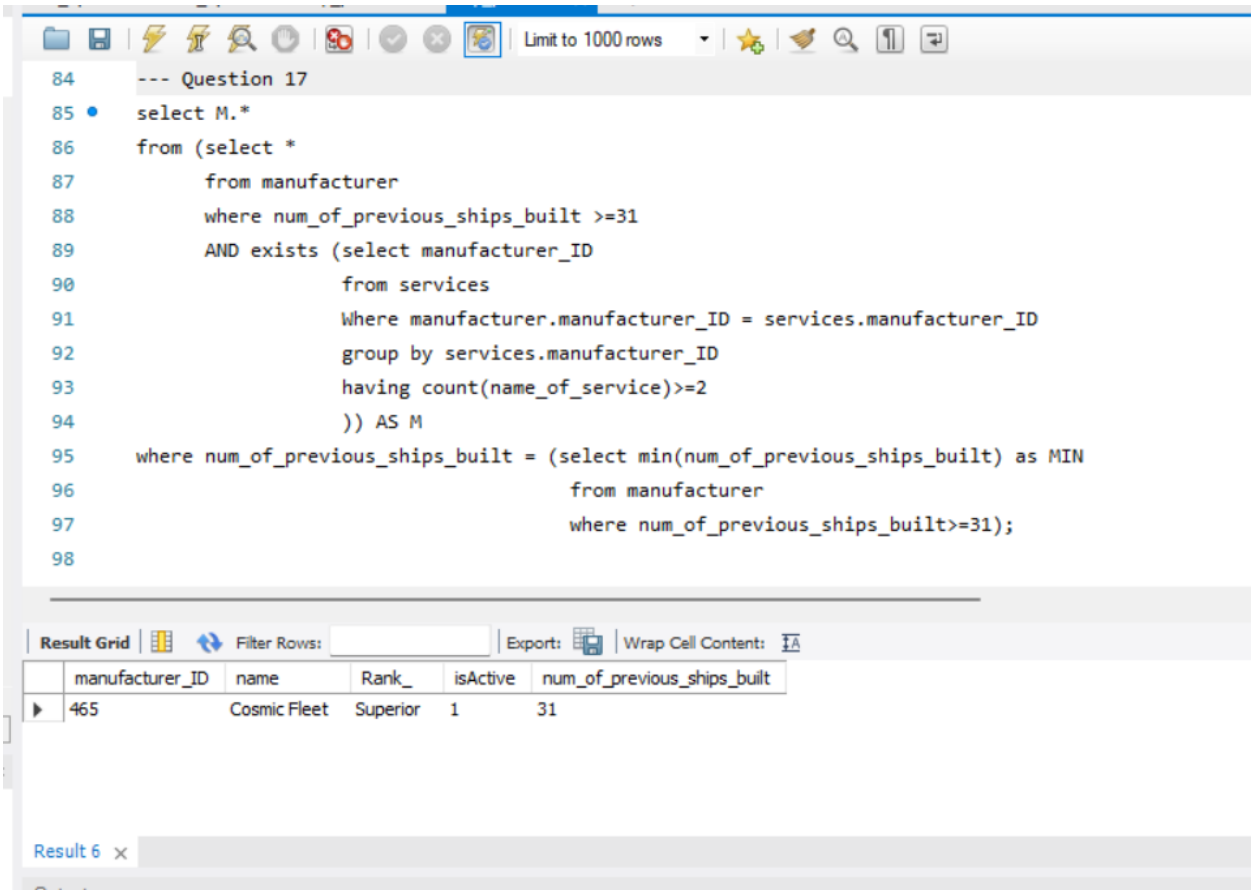
Q-16 A certain manager wants to perform some changes in branches infrastructure, yet these branches must contain all departments. Return the location, phone number, email of the branches that have all departments.

Solution: we utilize the division procedure in SQL by using the except and not exists keywords. The nested query first gets the DIDs of all possible departments and compares it (except) with the departments appearing in a specific branch correlated by comparing the IDs of the outer query with the inner query. If the inner query return an empty table so that every possible department appears in the branch and by that the branch is added to the solution set. Else, the tuple is not added to the solution set.



Q-17 some departments are concerned on some manufactures that are not working well on building spaceships. return all the information on the manufacturer with smallest number of spaceships built, taking into consideration only manufacturers who built 31 spaceships or more, and offered at least 2 services.

Solution: first we use a nested query in the from statement to remove any tuple not satisfying the minimal number of built spaceships. Inside that query we also insert a nested query using the "exists" keyword to check if a certain manufacturer is at least providing two services. The group by and count() words ensure this description. Surely a correlation is made between the outer query and the inner most query through comparing IDs. If the inner most query is non-empty the first constraints are checked ;so, in the where statement we calculate another nested query to retrieve the minimal number of spaceships built between (>=31 built) manufacturers if the manufacturer has the same minimal number, the tuple is added to the solution set.



```
--- Question 17
select M.*
from (select *
      from manufacturer
      where num_of_previous_ships_built >=31
      AND exists (select manufacturer_ID
                  from services
                  Where manufacturer.manufacturer_ID = services.manufacturer_ID
                  group by services.manufacturer_ID
                  having count(name_of_service)>=2
                  )) AS M
where num_of_previous_ships_built = (select min(num_of_previous_ships_built) as MIN
                                     from manufacturer
                                     where num_of_previous_ships_built>=31);
```

Limit to 1000 rows

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

manufacturer_ID	name	Rank_	isActive	num_of_previous_ships_built
465	Cosmic Fleet	Superior	1	31

Result 6

Q-18 The company suspects an overload on certain managers that are working on hiring applicants. To check if that is true, return a list containing managers names, IDs with the corresponding number of applicants they are managing to hire.

Solution: This query can be solved using a nested query in the select statement. After selecting the name of the manager, we traverse applicants table by correlating the manager ID in the managers table with Manger ID in the applicants table, after grouping the tuples based on the manager ID, we return the count of each group and represent it as "number of applicants managed".

```
                                        where num_of_previous_ships_built>=31);

    --- Question 18

    select Manager.name, Manager_ID, (select count(*)
                                      from applicant
                                      where applicant.Manager_ID = Manager.Manager_ID
                                      group by Manager_ID) As number_of_applicants_managed
    from Manager;
```

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

manufacturer_ID	name	Rank_	isActive	num_of_previous_ships_built
465	Cosmic Fleet	Superior	1	31

Q-19 update the blackholes table by multiplying the distance away from earth to blackholes away less than or equal 180000 by 1.05, blackholes away between 180000 and 220000 by 1.099, else multiply it by 1.1.

Solution: here update is used to update the distance related to blackholes, the case is used to differentiate between certain cases we have. The order here is important, we start by the greatest case and go downward to ensure a tuple does not appear in two consecutive cases.

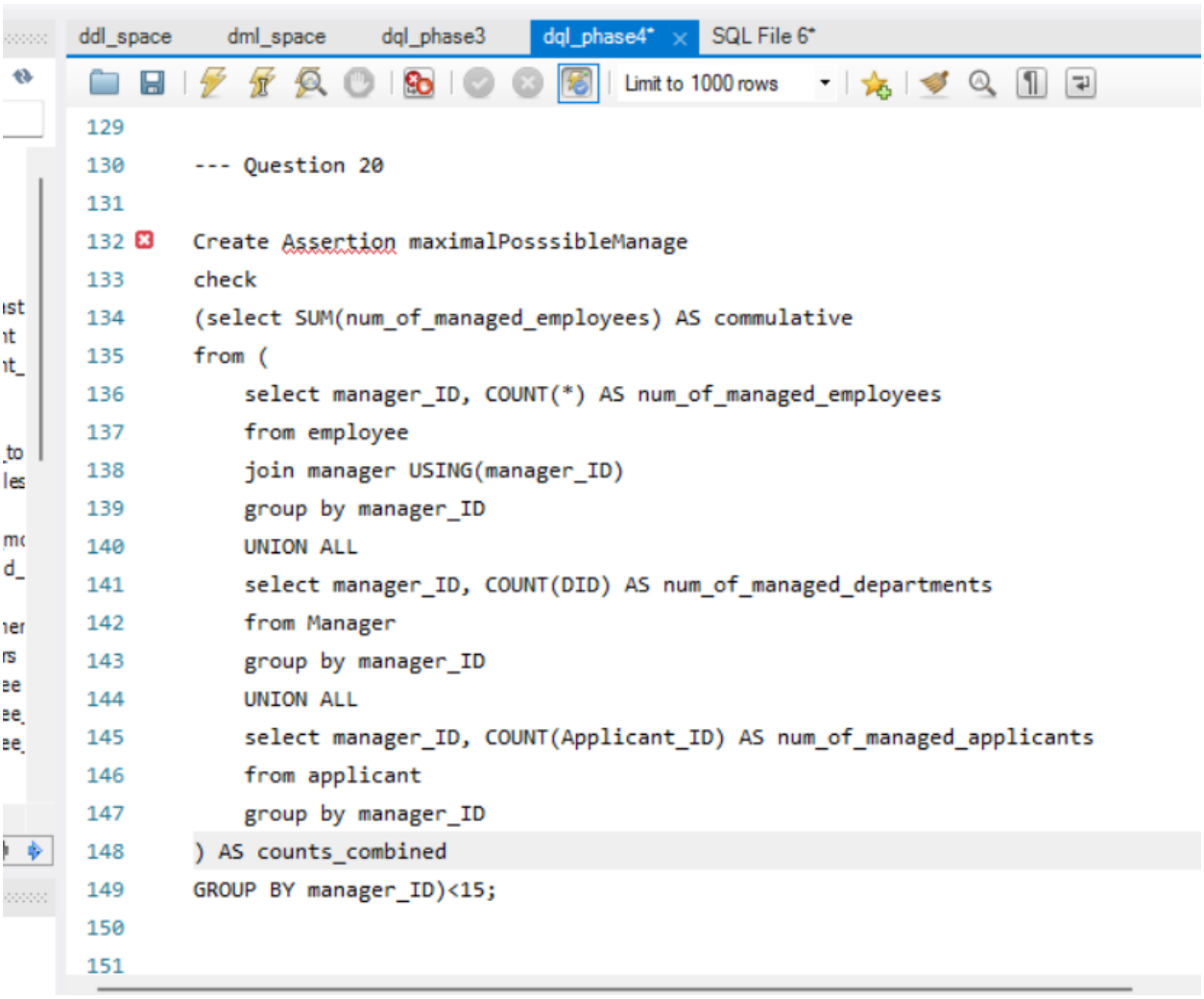
```
    --- Question 19

    update blackholes
    set distance_from_earth = CASE
                                WHEN distance_from_earth > 220000 THEN distance_from_earth*1.1
                                WHEN distance_from_earth > 180000 AND distance_from_earth <= 220000 THEN distance_from_earth*1.099
                                ELSE distance_from_earth*1.05
                              END
```

Q-20 The company does not want to create overload on its managers therefore it wants to limit the number of tasks the manager is responsible for managing to limit

it to maximum of 12. Create an assertion that ensures that every manager can manage a (cumulative of applicants, departments, employees) less than 15.

Solution: using the create assertion keyword and a check that uses the sum aggregate function between nested queries in the from statement. Each table in the inner query calculates a certain aspect and a union all joins all these tables to contain all possible counts. And then the SUM aggregate function sums these columns.

Note: MYSQL workbench does not support assertions so I could not test this query.



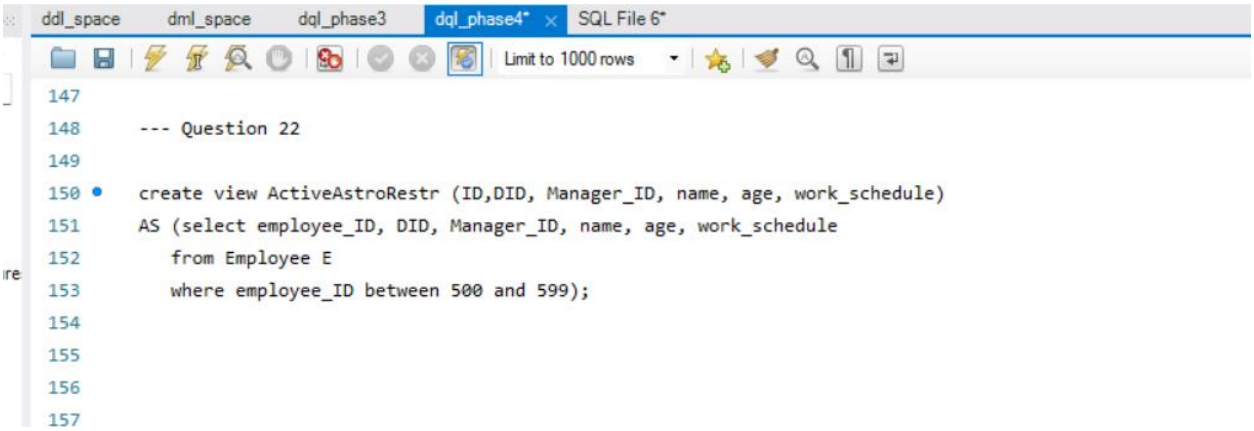
Q-21 New scientists should not be able to view the research studies of another scientist. The company wants to ensure the integrity of scientists, preventing other scientists from stealing their ideas.

Solution: take advantage of the create view keyword as illustrated below.

```
137
138
139
140
141     --- Qustion 21
142
143 •   create view protectedScintists (Scientist_ID, discoveries, university_degree, certificates)
144     AS (select S.Scientist_ID, S.discoveries, S.university_degree, S.certificates
145        from Scientist S);
146
147
148
149
150
151
152
153
```

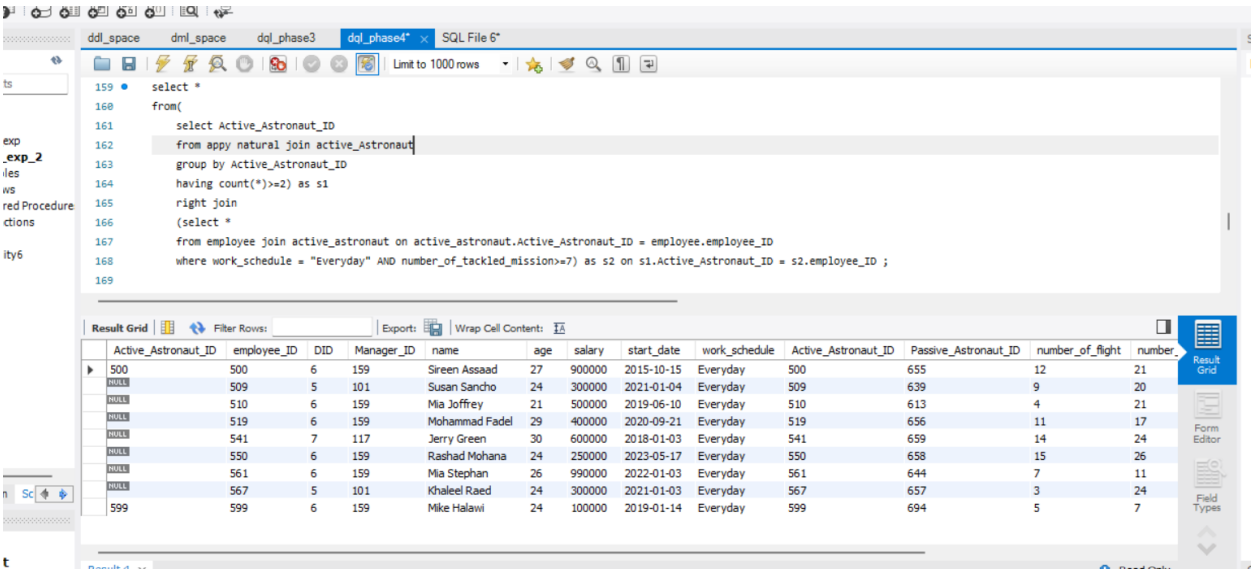
Q-22 Given that the range of IDs of Active Astronauts is from 500 => 599, ensure that no active astronaut can access the start date or salary of any employee to avoid possible comparisons that can lead to conflicts inside the company.

Solution: take advantage of the create view keyword as illustrated below.



Q-23 Display the tuples of all active astronauts that are applying 2 or more missions taking into consideration astronauts' number of tackled missions must greater or equal 7 ;and having an "Everyday" work schedule. Return all Tuples possible.

Solution: here it is necessary to use an outer join to ensure every tuple appears in the result table, first we select astronauts applying two or more missions in a separate table, and the calculate the astronauts with everyday schedule and tackled missions > 7 in another table. After that an outer RIGHT join is made to ensure that tuples from the second table respectively appear in the final result.(A null value denotes their appearance).



Q-24 The company now has a new update on upcoming managers. It wants to make sure that managers are getting a fair salary, they concluded that pay must be calculated based on experience of managers. Create a trigger that will ensure managers get their salary based on this formula: new salary = 10000 * experience + 65000.

Solution: we create a trigger to traverse the rows of the table managers when we update a certain tuple's experience a change for that must appear in the table. An example is stated below, the first picture denotes the salary of the manager "John Doe" initially before the trigger is 75000. After increasing the experience of John from 4 to 5 a change appears in his salary in the second table 100000.

```
167
168     --- Question 24
169
170 •   create trigger Manager_Salary
171     before update On manager
172     For each row
173     set new.salary = (10000 * old.experience) + 60000;
174
175
176
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Manager_ID	DID	name	salary	experience
100	8	Emily Davis	95000	5
101	1	John Doe	75000	4
102	2	Jane Smith	70000	3
103	3	Alice Johnson	80000	2
104	4	Bob Williams	72000	5
106	6	Samantha Jones	72000	5
108	11	Ethan Wilson	45000	5
117	15	Alexander Garcia	88000	5
120	5	Emma Thompson	68000	1
129	7	Michael Johnson	72000	5
130	10	Olivia Brown	60000	5
144	12	Ava Anderson	68000	4

manager 2 ×

Output

Action Output

```
171     before update On manager
172     For each row
173     set new.salary = (10000 * old.experience) + 60000;
174
175 •   update manager
176     set manager.experience = 5
177     where Manager_ID = 101;
178
179
180 •   select * from manager
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conte

Manager_ID	DID	name	salary	experience
100	8	Emily Davis	95000	5
101	1	John Doe	100000	5
102	2	Jane Smith	70000	3
103	3	Alice Johnson	80000	2
104	4	Bob Williams	72000	5
106	6	Samantha Jones	72000	5
108	11	Ethan Wilson	45000	5
117	15	Alexander Garcia	88000	5
120	5	Emma Thompson	68000	1
129	7	Michael Johnson	72000	5
130	10	Olivia Brown	60000	5
144	12	Ava Anderson	68000	4

manager 3 ×

Output

Q-25 The database administrators wants a function to keep track of the number of missions each scientist has come up with so far.

Solution: here a function is created with an integer return type. Then, after the BEGIN, we declare a variable called counter to return it. The function takes as a parameter the ID of the manager we want to calculate the number of missions they sat up, so with a correlation between this ID and a query calculated using the count(*) aggregate function the counter is returned at the end.

```
174
175
176     --- Question 25
177
178 •   create function get_missions_done_by_scientist (ID int)
179             returns Integer
180         BEGIN
181             Declare counter int;
182 •               Select count(*) into counter
183                 from sets_up
184                 where sets_up.Scientist_ID = ID
185             return counter;
186         END;
187
188
189
190
191
192
193
194
195
196
```

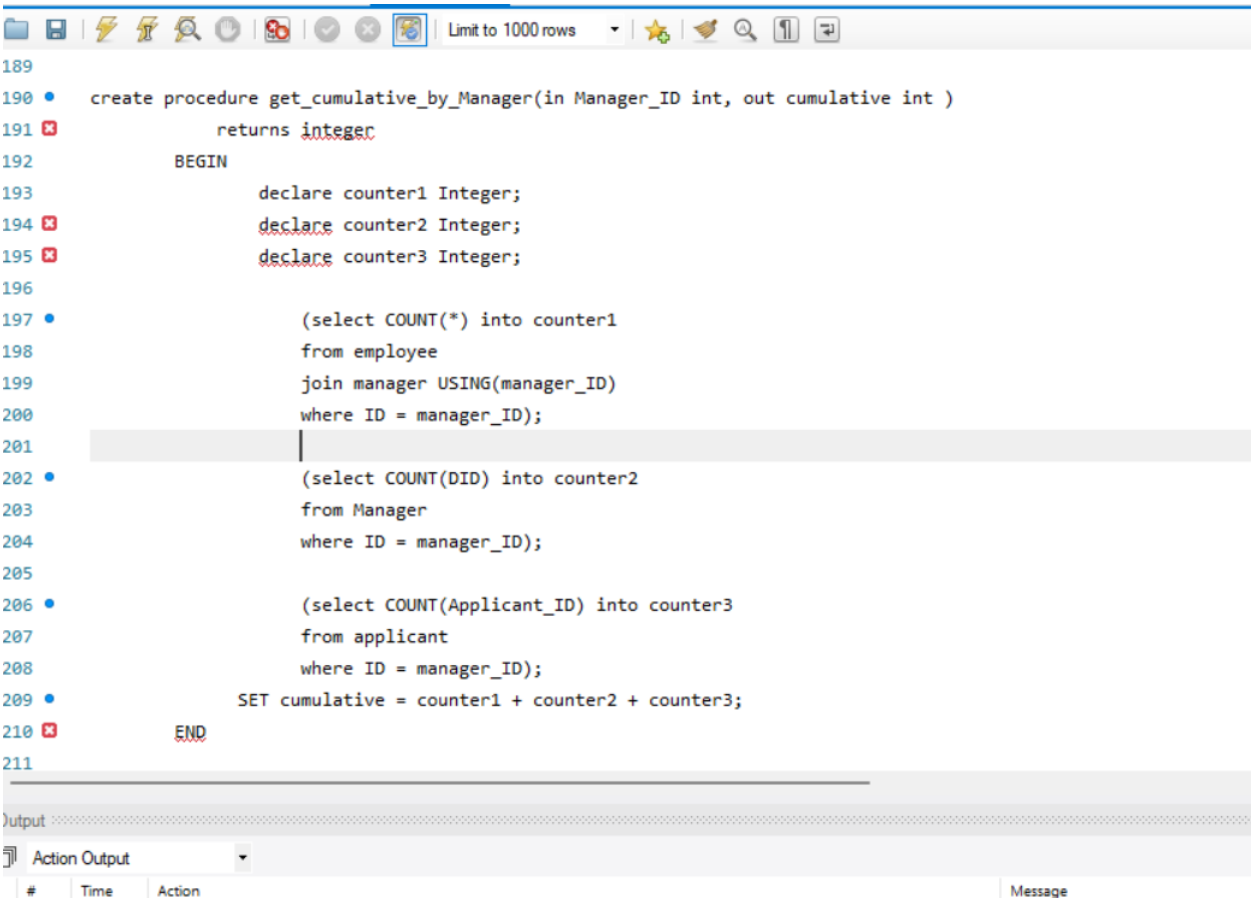
Output

Action Output

Q-26 since the database administrators are interested in getting the cumulative managed departments, applicants, and employees by each manager. Make their lives easier by creating a procedure for that.

Solution: In this query a procedure is programmed first by specifying its return type which is integer, the procedure takes on two parameters which are input and

output, the IN represents the Manager_ID which distinguishes every manager, while the output is for the final result. We declare three counters to store the departments, applicants, and employees for a specific manager using a correlation in each of them. At the end the cumulative is set to the sum of these counters, and the procedure ends.



Conclusion :

The paper discusses the development of an Entity-Relationship (ER) model and a relational model to represent the database structure. It delves into the relationships and attributes of each entity, providing a detailed overview of the database schema and its components. The thorough exploration of the ER and relational models demonstrates a meticulous approach to database design, ensuring that the data is effectively structured, and relationships are accurately represented. The paper underscores the significance of a well-organized and comprehensive database in the context of space exploration. It emphasizes the potential for the database to facilitate efficient management of information about employees, scientists, spaceships, manufacturers, and various celestial objects. By providing a detailed overview of the database's entities, relationships, and attributes, the paper serves as a valuable resource for understanding the intricacies of organizing and managing data in the field of space exploration. The project involved the implementation of a comprehensive database for a space company using MySQL Workbench, encompassing tables for various aspects of the company's operations. The database was designed to address the complex data management needs of the space company, including tables for departments, managers, employees, branches, applicants, trainers, astronauts, scientists, missions, manufacturers, services, spaceships, spacecraft locations, and more. The implementation process involved creating the actual database for the model using MySQL Workbench. The tables were initialized using Data Definition Language (DDL) SQL commands, with careful consideration given to the relationships between different entities within the database. For example, the Department table had a one-to-one relationship with the Manager table, and the Employee table included foreign keys for the Department and Manager tables. The project also included the creation of advanced SQL solutions to address specific data management and integrity issues within the database. For instance, a trigger was

implemented to update the salary of a manager when their experience changed, ensuring that the salary remained accurate and up-to-date . Additionally, a function was developed to track the number of missions each scientist had initiated, providing valuable insights for database administrators. Furthermore, a procedure was created to calculate the cumulative number of managed departments, applicants, and employees for each manager, streamlining the process for database administrators and managers . These advanced SQL solutions demonstrated the effective use of SQL for database management within the context of a space company. The project also included the implementation of basic SQL queries to showcase the functionality of the database. These queries addressed various scenarios, such as identifying applicants managed by specific managers, finding managers based on department budget and capacity criteria, and retrieving information about trainers and astronauts based on specific criteria. Overall, the project showcased the successful implementation of a comprehensive database for a space company, addressing a wide range of data management and integrity challenges. The use of MySQL Workbench, SQL queries, triggers, functions, and procedures demonstrated the versatility and power of SQL for managing complex databases in a real-world business context. In conclusion, the project provided valuable insights into the design and implementation of a database for a space company, highlighting the importance of effective data management and integrity within such a specialized industry. The solutions presented in the project showcased the practical application of SQL for addressing various data management challenges, ultimately contributing to the efficient operation of the space company's database system. On the other hand, the project's findings highlight the importance of effective data management and integrity within a specialized industry such as space exploration. The methods employed, including the use of triggers to update data, assertions to limit tasks for managers, and procedures to track and manage data, have proven to be valuable tools for maintaining the integrity and accuracy of the database.

Moving forward, the project's methods can be further applied to improve the database implementation by continuously refining and optimizing the SQL queries, triggers, and procedures. Additionally, the project's findings can serve as a basis for future enhancements, such as the integration of advanced data analytics and reporting capabilities to provide valuable insights for decision-making within the space company. Overall, the project's results demonstrate the effectiveness of SQL-based solutions for database management and provide a solid foundation for further improvements and advancements in the implementation of the space company's database.