**CONTENTS**

1. Create DB named ‘EMS’ (CREATE DATABASE statement)
2. Create tables ‘Employee’ and ‘Department’ (CREATE TABLE statement)
3. Add new columns to the table (ALTER TABLE statement)
4. Insert data to the table (INSERT statement)
5. Update the table (UPDATE statement)
6. Fetch all records (SELECT statement)
7. Fetch specific records (WHERE statement)
8. Fetch specific records (IN statement)
9. Sort the records (ORDER BY statement)
10. Relational Operators
11. DISTINCT
12. LIMIT
13. Aggregate Functions - SUM, AVG, MIN, MAX, COUNT
14. AND
15. DELETE
16. INNER JOIN
17. LEFT JOIN
18. RIGHT JOIN
19. CROSS JOIN
20. FOREIGN KEY
21. SELF JOIN
22. Alter datatype of a column in an existing table
23. Alter table name of an existing table
24. UNION
25. UNION ALL
26. GROUP BY
27. GROUP BY HAVING
28. LIKE
29. NOT LIKE
30. BETWEEN
31. ALIASES, CONCAT
32. EXISTS operator
33. ROW\_NUMBER()
34. ALL, ANY operator
35. CASE Expression
36. COALESCE() and IFNULL() functions
37. STORED PROCEDURE
38. CTE (Common Table Expression)
39. AUTO\_INCREMENT
40. TEMPORARY TABLE
41. GENERATED COLUMN
42. SQL CONSTRAINTS
43. INSERT ON DUPLICATE KEY UPDATE
44. Create a database named ‘EMS’ to store the employee and department details.

create database if not exists EMS;

1. To let the computer know the database we want to apply the code and operations, use the following command:

use EMS;

1. Create two tables namely ‘Employee’ and ‘Department’
2. For creating ‘Employee’ table used the following query:

create table Employee

(

EmpId integer not null primary key,

EmpName varchar(100),

DeptId integer,

JoiningDate date not null,

EmailId varchar(100) not null,

Address varchar(100)

);

1. For creating ‘Department’ table used the following query:

create table Department

(

DeptID integer not null primary key,

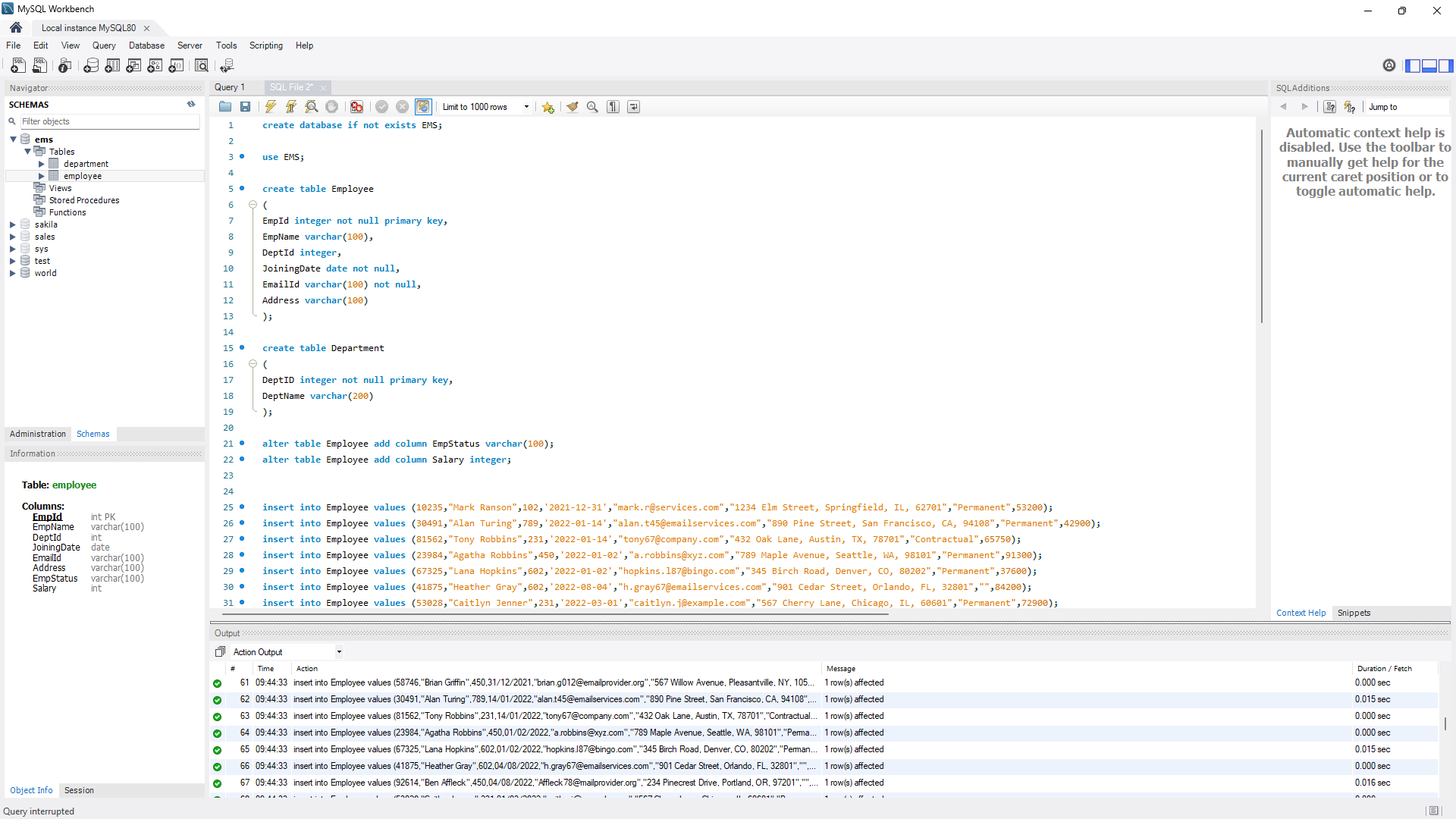
DeptName varchar(200)

);

1. To add new columns in the ‘Employee’ table:

alter table Employee add column EmpStatus varchar(100);

alter table Employee add column Salary integer;



1. Added few dataset for the ‘Employee’ table using the following INSERT commands:

insert into Employee values (10235,"Mark Ranson",102,'2021-12-31',"mark.r@services.com","1234 Elm Street, Springfield, IL, 62701","Permanent",53200);

insert into Employee values (30491,"Alan Turing",789,'2022-01-14',"alan.t45@emailservices.com","890 Pine Street, San Francisco, CA, 94108","Permanent",42900);

insert into Employee values (81562,"Tony Robbins",231,'2022-01-14',"tony67@company.com","432 Oak Lane, Austin, TX, 78701","Contractual",65750);

insert into Employee values (23984,"Agatha Robbins",450,'2022-01-02',"a.robbins@xyz.com","789 Maple Avenue, Seattle, WA, 98101","Permanent",91300);

insert into Employee values (67325,"Lana Hopkins",602,'2022-01-02',"hopkins.l87@bingo.com","345 Birch Road, Denver, CO, 80202","Permanent",37600);

insert into Employee values (41875,"Heather Gray",602,'2022-08-04',"h.gray67@emailservices.com","901 Cedar Street, Orlando, FL, 32801","",84200);

insert into Employee values (53028,"Caitlyn Jenner",231,'2022-03-01',"caitlyn.j@example.com","567 Cherry Lane, Chicago, IL, 60601","Permanent",72900);

insert into Employee values (14789,"Gibbs Duhem",231,'2022-03-01',"duhem.gibbs@company.com","789 Magnolia Boulevard, Miami, FL, 33101","Permanent",59400);

insert into Employee values (76293,"Anderson Cooper",450,'2022-03-01',"anderson.cooper@emailhosting.biz","123 Sycamore Street, Boston, MA, 02108","Contractual",81000);

insert into Employee values (31507,"Michael Douglas",231,'2022-03-01',"michael.d@provider.net","456 Cedar Avenue, Phoenix, AZ, 85001","Contractual",46800);

insert into Employee values (64823,"Sophia Anderson",450,'2022-03-15',"sophia.a56@company.com","789 Birch Lane, Atlanta, GA, 30301","",69250);

insert into Employee values (89016,"Benjamin Martinez",875,'2022-03-15',"benjamin.m@example.com","234 Oak Street, Dallas, TX, 75201","",56700);

insert into Employee values (42756,"Harper Johnson",333,'2022-04-01',"harper.j99@bingo.com","567 Elm Avenue, Los Angeles, CA, 90001","",88600);

insert into Employee values (86532,"Mason Thompson",333,'2022-06-01',"m.thompson@emailservices.net","456 Pinecrest Road, Philadelphia, PA, 19101","",43500);

insert into Employee values (29104,"Ava Nguyen",789,'2022-07-02',"ava.n23@company.com","789 Birch Avenue, Nashville, TN, 37201","",62150);

insert into Employee values (102,"Olivia Ramirez",21984,'2022-04-17',"123 Maple Lane, Houston, TX, 77001",'olivia.r@emailhosting.biz',"",null);

insert into Employee values (450,"Brian Griffin",58746,'2021-12-31',"567 Willow Avenue, Pleasantville, NY, 10570","brian.g012@emailprovider.org","Permanent",78500);

insert into Employee values (92614,"Ben Affleck",450,'2022-08-04',"","234 Pinecrest Drive, Portland, OR, 97201","",50150);

insert into Employee values (57389,"Elijah Brown",333,'2022-04-01',"","","",49300);

insert into Employee values (73659,"Logan Patel",null,'2022-07-15',"logan.p@xyz.com","234 Magnolia Street, Las Vegas, NV, 89101","",96400);

insert into Employee values (60238,"Mia Garcia",null,'2022-01-03',"mia.g77@services.com","567 Cedar Drive, Minneapolis, MN, 55401","Permanent",40000);

1. Added few dataset for the ‘Department’ table using the following INSERT commands:

insert into Department values (102,"Human Resources");

insert into Department values (450,"Sales");

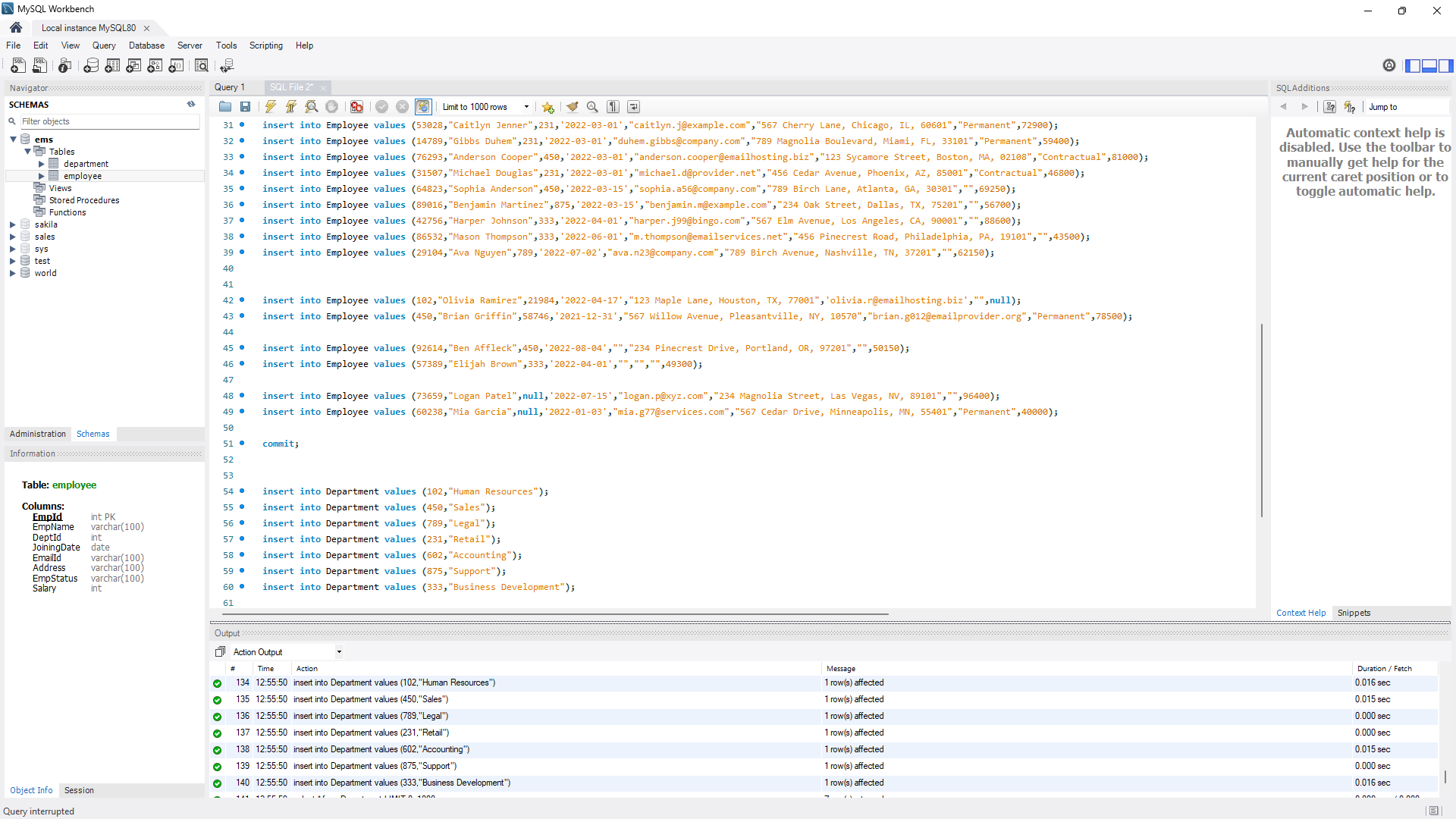
insert into Department values (789,"Legal");

insert into Department values (231,"Retail");

insert into Department values (602,"Accounting");

insert into Department values (875,"Support");

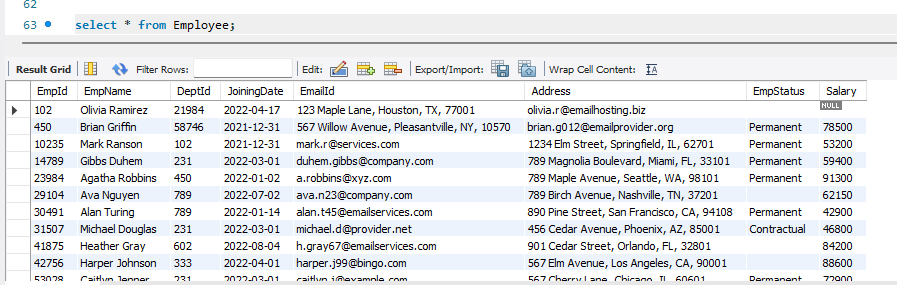
insert into Department values (333,"Business Development");



1. For the saving the changes ran the following command:

commit;

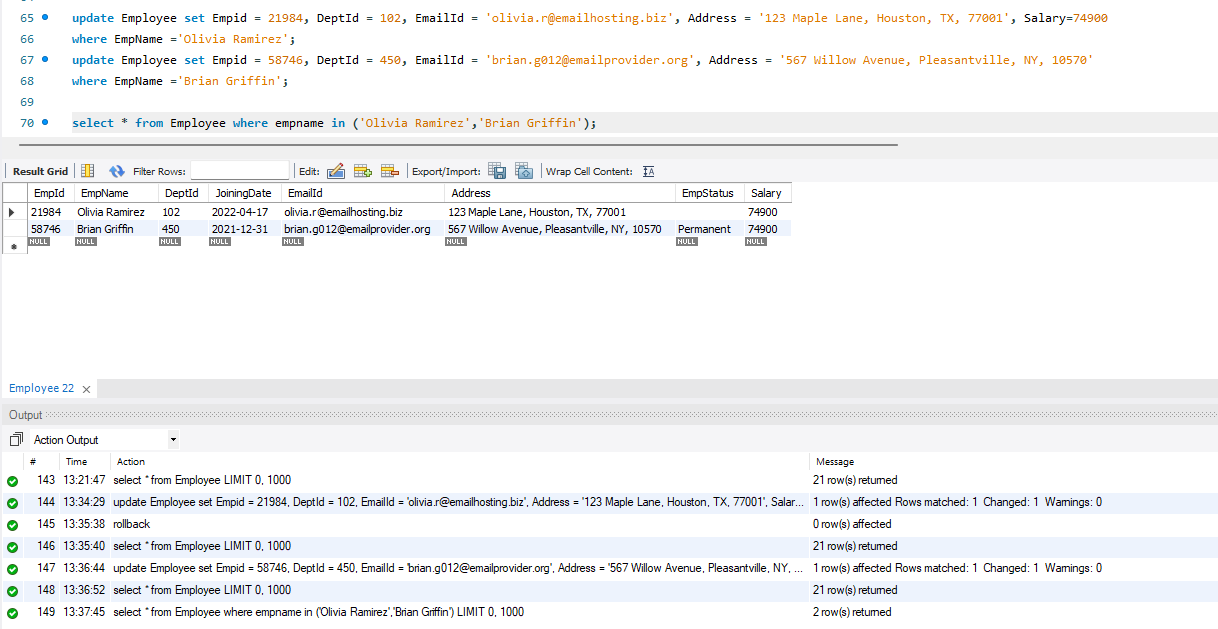
1. As we can see the ‘Employee’ table has some value mismatch



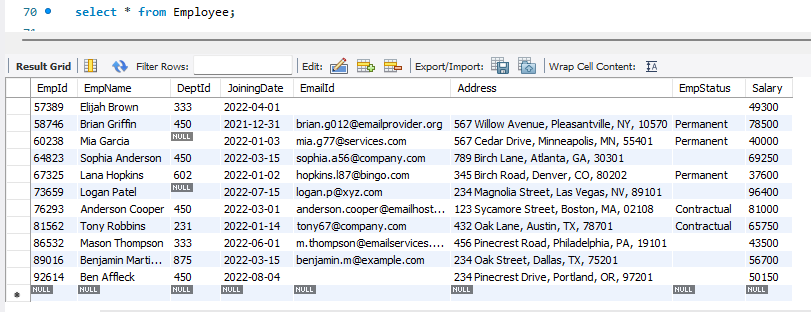
1. Olivia and Brian have its EmpId, DeptId, emailId and addresses mismatched. Even Olivia’s salary is not mentioned. To correct it we ran the following commands:

update Employee set Empid = 21984, DeptId = 102, EmailId = 'olivia.r@emailhosting.biz', Address = '123 Maple Lane, Houston, TX, 77001', Salary=74900 where EmpName ='Olivia Ramirez';

update Employee set Empid = 58746, DeptId = 450, EmailId = 'brian.g012@emailprovider.org', Address = '567 Willow Avenue, Pleasantville, NY, 10570' where EmpName ='Brian Griffin';

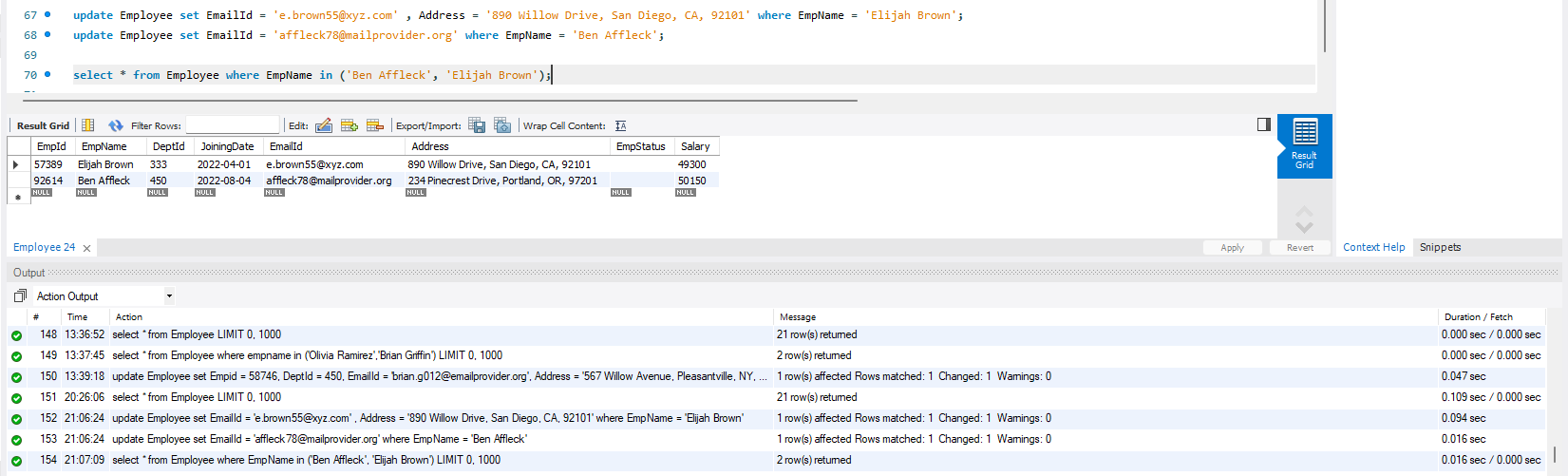


1. Elijah have its EmailId and Address empty, Ben have its EmailId empty. To populate it we ran the following queries:



update Employee set EmailId = 'e.brown55@xyz.com' , Address = '890 Willow Drive, San Diego, CA, 92101' where EmpName = 'Elijah Brown';

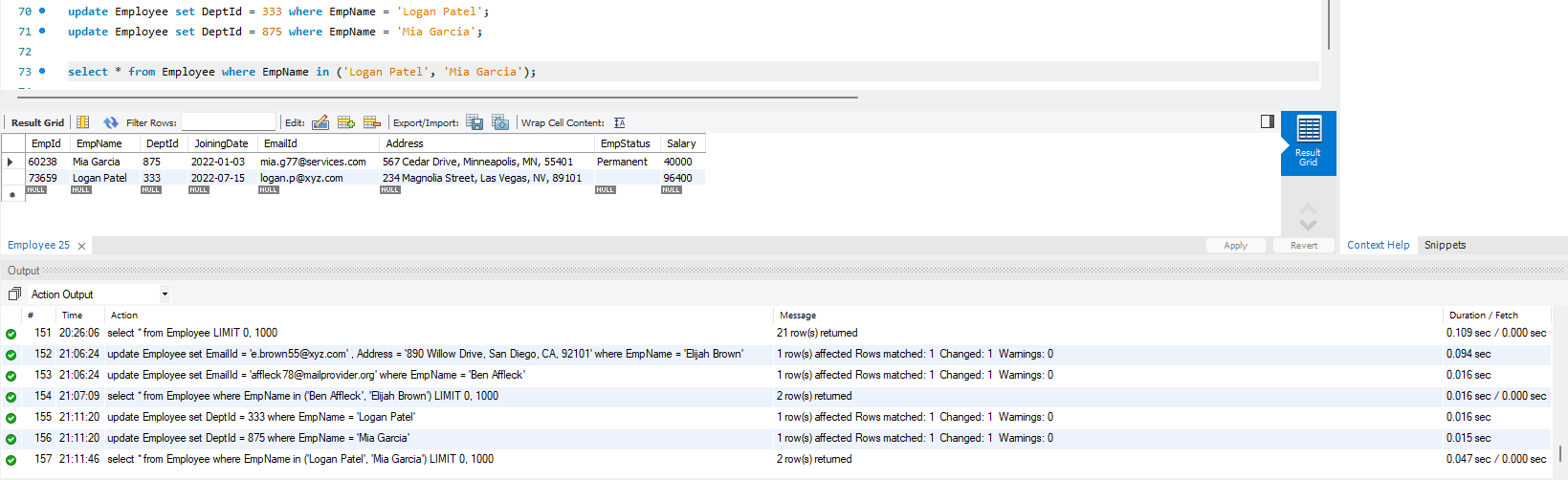
update Employee set EmailId = 'affleck78@mailprovider.org' where EmpName = 'Ben Affleck';



1. Mia and Logan have its DeptId empty. To populate it ran the following queries:

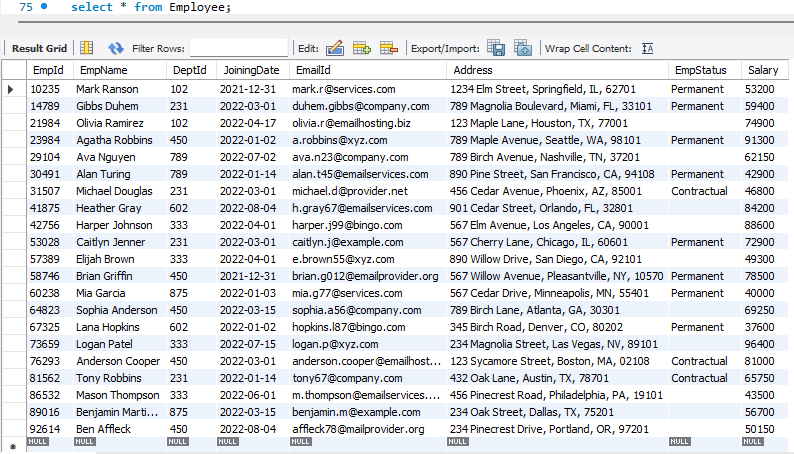
update Employee set DeptId = 333 where EmpName = 'Logan Patel';

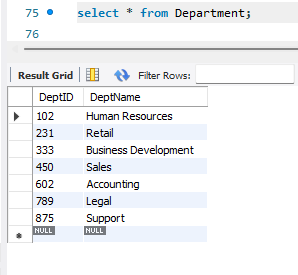
update Employee set DeptId = 875 where EmpName = 'Mia Garcia';



1. To view all the records of the table we can run the following query:

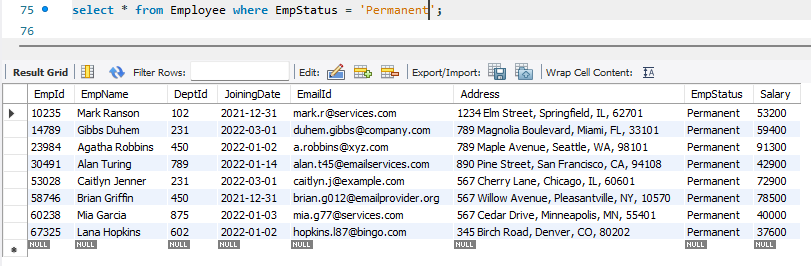
select \* from Employee;





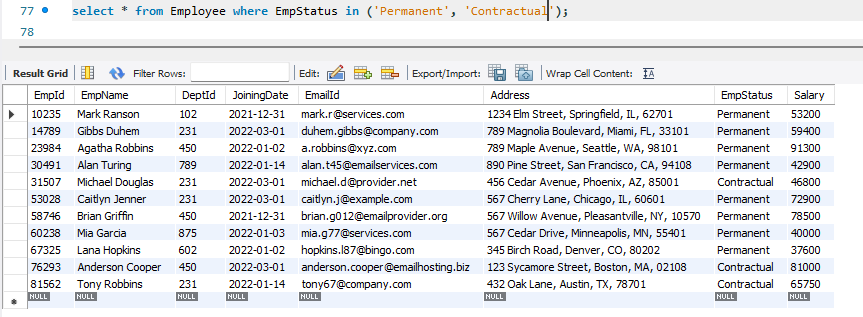
1. To select a particular record like if we want to view the record of permanent employees :

select \* from Employee where EmpStatus = 'Permanent';



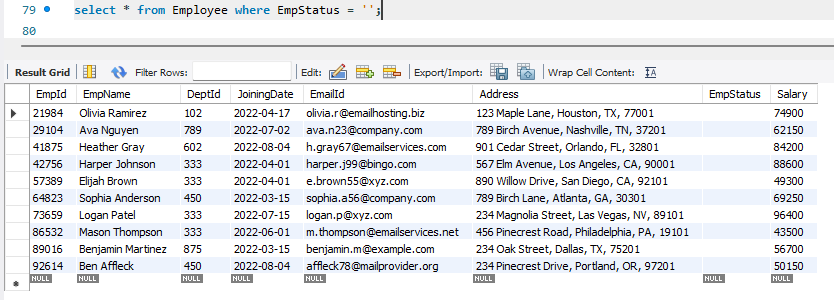
1. To view records of employees whose employee status is either permanent or contractual:

select \* from Employee where EmpStatus in ('Permanent', 'Contractual');



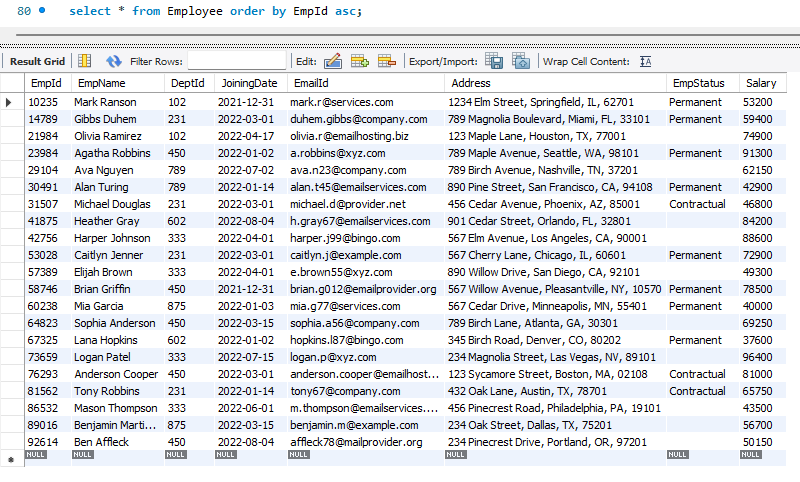
1. To view records of employees whose employee status are not populated on the table:

select \* from Employee where EmpStatus = '';



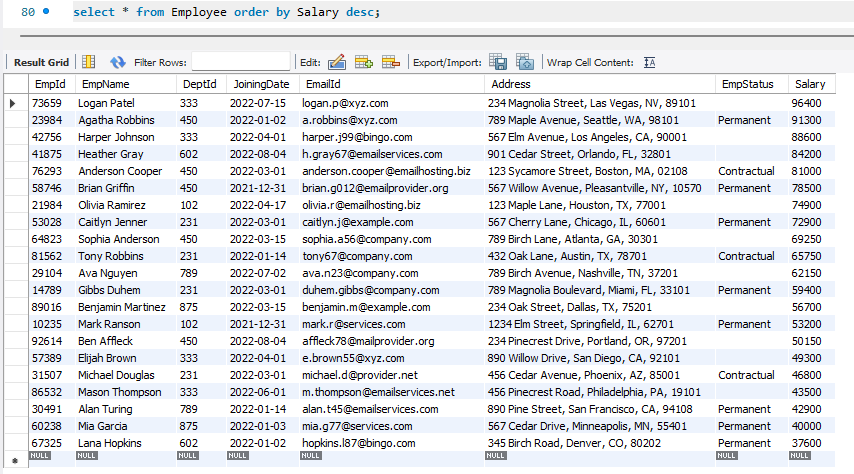
1. To view records of employees sorted in ascending order by their employee id:

select \* from Employee order by EmpId asc;



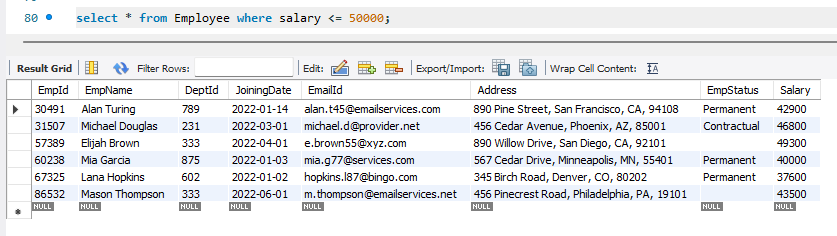
1. To view records of employees sorted in descending order by their salary:

select \* from Employee order by Salary desc;



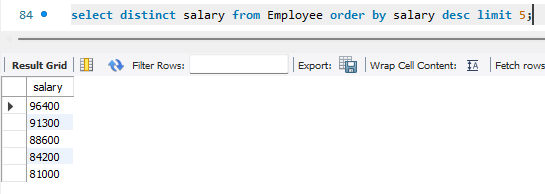
1. To view records of employees whose salary is less than equal to 50000

select \* from Employee where salary <= 50000;



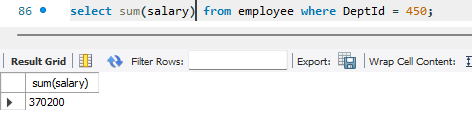
1. To view the top 5 records of employees earning the highest salary:

select distinct salary from Employee order by salary desc limit 5;



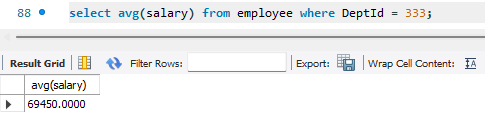
1. To view the sum of salary of a particular department id:

select sum(salary) from employee where DeptId = 450;



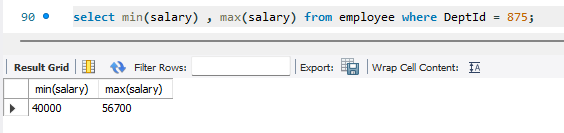
1. To view the average salary of a particular department id:

select avg(salary) from employee where DeptId = 333;



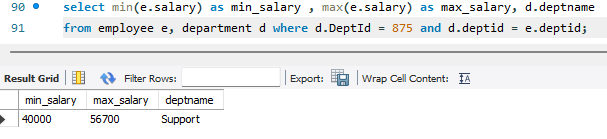
1. To view the minimum and maximum salary of a particular department:

select min(salary) , max(salary) from employee where DeptId = 875;



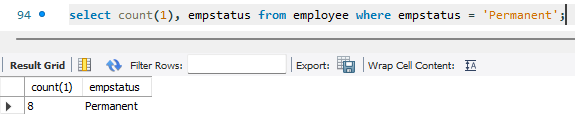
1. To view the department name along with the minimum and maximum salary :

select min(e.salary) as min\_salary , max(e.salary) as max\_salary, d.deptname from employee e, department d where d.DeptId = 875 and d.deptid = e.deptid;



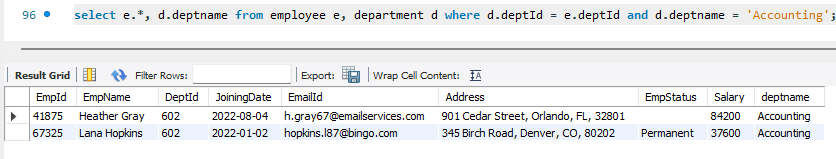
1. To view the count of employees who have a permanent job status :

select count(1), empstatus from employee where empstatus = 'Permanent';



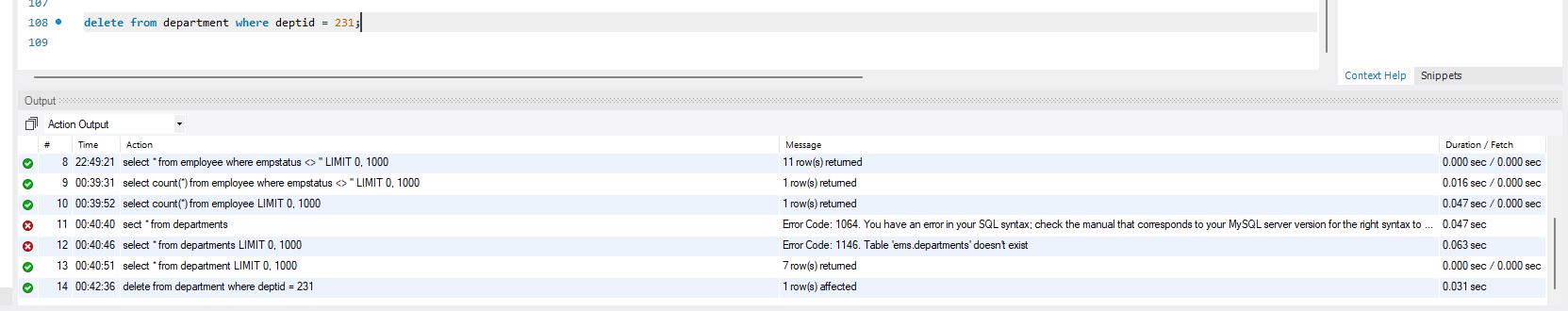
1. To view the employee details of Accounting department:

select e.\*, d.deptname from employee e, department d where d.deptId = e.deptId and d.deptname = 'Accounting';



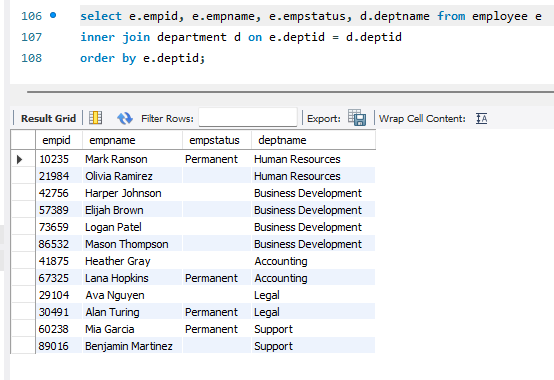
1. To delete any record from a table:

delete from department where deptid = 231;



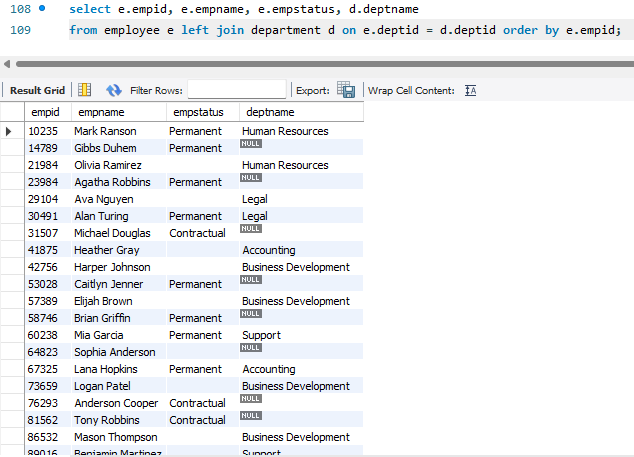
1. To view the records of employees along with their department names :

select e.empid, e.empname, e.empstatus, d.deptname from employee e inner join department d on e.deptid = d.deptid order by e.deptid;



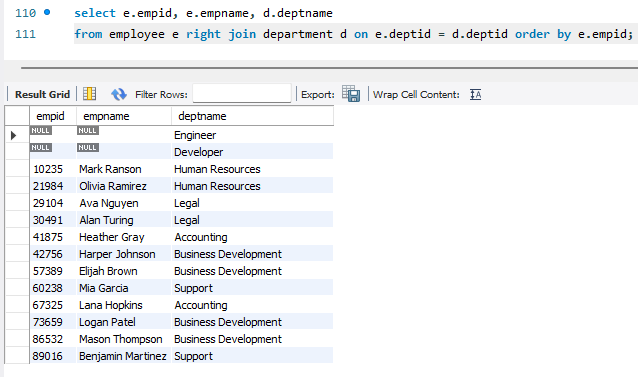
1. To generate a report of all employees, including those who are not assigned to any department yet:

select e.empid, e.empname, e.empstatus, d.deptname from employee e left join department d on e.deptid = d.deptid order by e.empid;



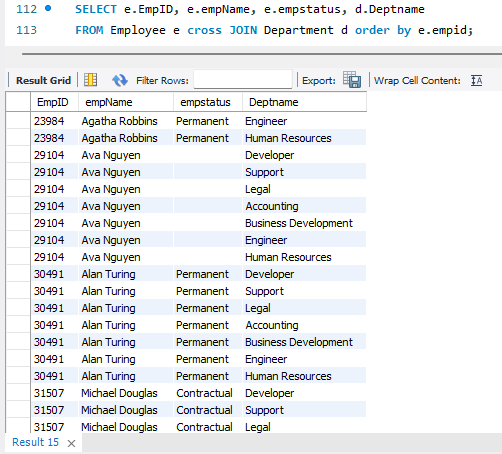
1. To generate a report listing all departments, including those without any assigned employees:

select e.empid, e.empname, d.deptname from employee e right join department d on e.deptid = d.deptid order by e.empid;



1. A **CROSS JOIN** produces a result set that is a combination of every row from the first table with every row from the second table.

SELECT e.EmpID, e.empName, e.empstatus, d.Deptname from Employee e cross join Department d order by e.empid;



NOTE: If you add a **WHERE** clause (if table1 and table2 has a relationship), the **CROSS JOIN** will produce the same result as the **INNER JOIN** clause.

1. To add a **foreign key constraint**:

alter table employee add constraint fk\_em foreign key (ManagerId) references employee(empid);

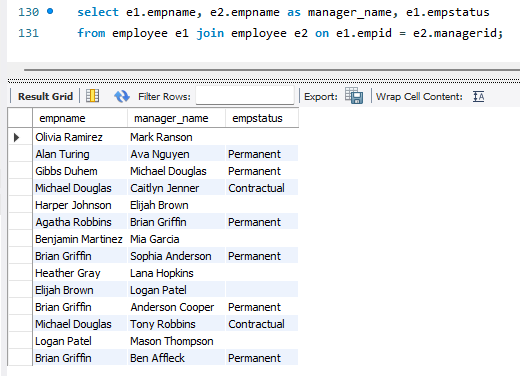
1. A **SELF JOIN** is used when a table needs to be joined with itself, often to compare rows within the same table. In this scenario, let's say the Employees table contains a hierarchical relationship where each employee reports to another employee (who is also an employee in the same table).

Employees Table:

Columns: EmployeeID (Primary Key), Name, ManagerID (Foreign Key referencing EmployeeID in the same table)

SQL Query Using Self Join: We'll use a self join to retrieve the names of employees along with the names of their respective managers.

select e1.empname, e2.empname as manager\_name, e1.empstatus from employee e1 join employee e2 on e1.empid = e2.managerid;



1. To change the datatype of a column in an existing table:

alter table Employee\_add modify column PHONENO varchar(20);

1. To change table name of an existing table:

alter table employee\_add rename employee\_client;

alter table employee rename employee\_parent\_company;

1. Created an employee additional table :

create table Employee\_add(

empid int not null primary key,

empname varchar(100) not null,

deptid int,

ISD\_Code varchar(10),

PHONENO varchar(10),

BONUS int

);

alter table Employee\_add modify column PHONENO varchar(20);

1. Populated with the following values:

insert into employee\_add values(10014,"Emily Johnson",102,"+1 217","555-123-4567",2500);

insert into employee\_add values(21561,"Benjamin Carter",249,"+1 305","987-654-3210",3000);

insert into employee\_add values(37896,"Samantha Adams",875,"+1 713","123-456-7890",8521);

insert into employee\_add values(45217,"Lucas Thompson",789,"+1 206","555-789-123",4576);

insert into employee\_add values(56323,"Natalie Walker",102,"+1 615","321-555-9876",6275);

insert into employee\_add values(62109,"Elijah Parker",902,"+1 415","999-888-7777",3735);

insert into employee\_add values(78902,"Olivia Rodriguez",875,"+1 480","444-333-2222",7052);

insert into employee\_add values(83454,"Ethan Campbell",789,"+1 407","777-666-5555",5587);

insert into employee\_add values(91235,"Sophia Mitchell",249,"+1 213","234-567-8901",1497);

insert into employee\_add values(10221,"Jacob Wright",102,"+1 312","876-543-2109",4429);

insert into employee\_add values(13456,"Ava Anderson",901,"+1 619","111-222-3333",5198);

insert into employee\_add values(14568,"Matthew Turner",875,"+1 914","555-444-3333",3379);

insert into employee\_add values(17891,"Grace Flores",901,"+1 612","888-777-6666",2879);

insert into employee\_add values(18907,"William King",789,"+1 404","666-555-4444",4355);

insert into employee\_add values(20013,"Harper Cooper",249,"+1 303","987-654-3210",1987);

insert into employee\_add values(22346,"Alexander Hill",102,"+1 702","123-456-7890",7364);

insert into employee\_add values(23560,"Chloe Brooks",789,"+1 617","222-333-4444",9073);

insert into employee\_add values(24572,"James Bennett",875,"+1 512","777-888-9999",8647);

insert into employee\_add values(25671,"Lily Rivera",902,"+1 215","555-222-3333",6790);

insert into employee\_add values(28907,"Michael Gray",249,"+1 214","444-555-6666",7891);

insert into employee\_add values(29873,"Charlotte Reed",902,"+1 503","987-654-3210",6687);

insert into employee\_client values(10235,"Mark Ranson",102,"+1 217","555-649-888",6510);

insert into employee\_client values(14789,"Gibbs Duhem",231,"+1 305","649-888-214",5107);

insert into employee\_client values(23984,"Agatha Robbins",450,"+1 206","888-214-987",7510);

insert into employee\_client values(29104,"Ava Nguyen",789,"+1 615","214-789-259",8543);

insert into employee\_client values(30491,"Alan Turing",789,"+1 415","789-259-864",4387);

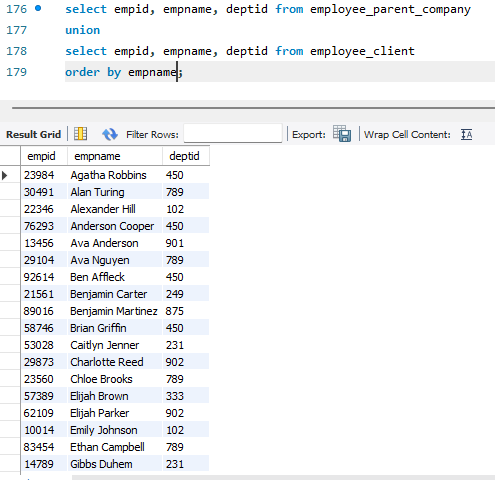
1. To view the employee details of the parent company and the client company

select empid, empname, deptid from employee\_parent\_company

union

select empid, empname, deptid from employee\_client

order by empname;



NOTE: A **UNION** operation allows you to combine the results of two or more SELECT statements. It eliminates duplicate rows by default. If you want to include duplicates, you can use **UNION ALL**.

The **UNION** operation is helpful when you have similar data structures in different tables and you want to combine their results into a single list.

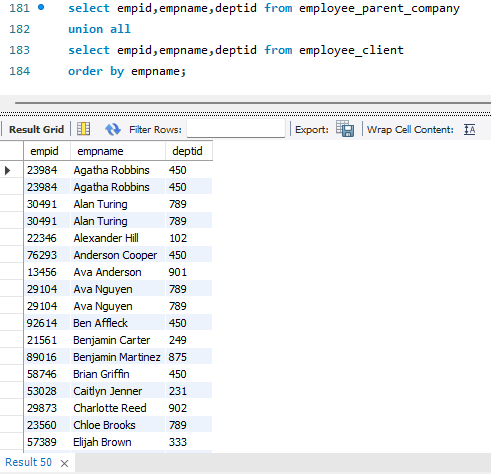
1. To view the duplicate employee records in both the table

select empid,empname,deptid from employee\_parent\_company

union all

select empid,empname,deptid from employee\_client

order by empname;

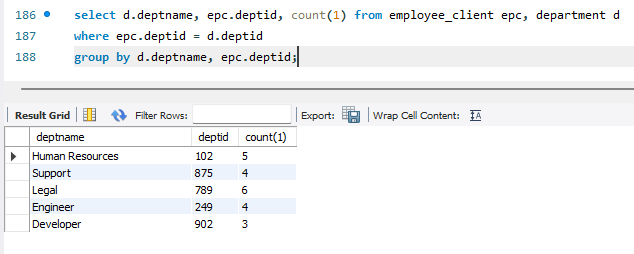


1. To view the number of employees in client company in each department

select d.deptname, epc.deptid, count(1) from employee\_client epc, department d

where epc.deptid = d.deptid

group by d.deptname, epc.deptid;

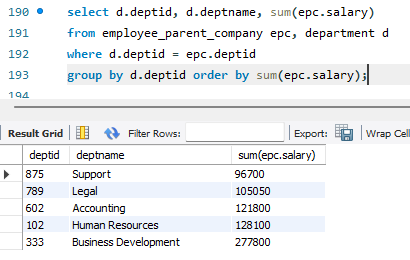


select d.deptid, d.deptname, sum(epc.salary)

from employee\_parent\_company epc, department d

where d.deptid = epc.deptid

group by d.deptid order by sum(epc.salary);



1. To view the result of

This scenario demonstrates the use of HAVING to filter aggregated data based on conditions after performing a GROUP BY operation. It's useful when you want to filter grouped results by an aggregate function (such as SUM, COUNT, AVG, etc.) in MySQL.

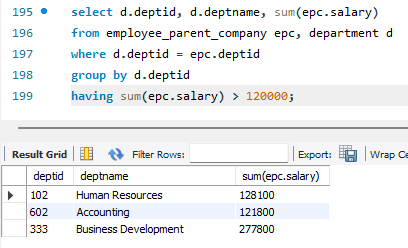
select d.deptid, d.deptname, sum(epc.salary)

from employee\_parent\_company epc, department d

where d.deptid = epc.deptid

group by d.deptid

having sum(epc.salary) > 120000;



1. To view the top 5 high paying employees in the parent company

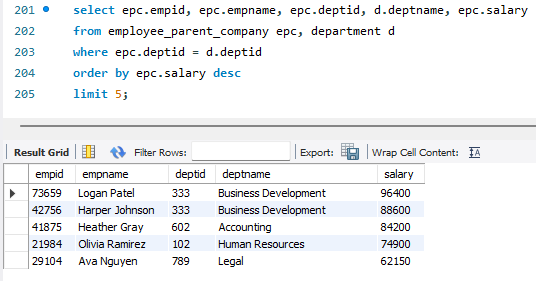
select epc.empid, epc.empname, epc.deptid, d.deptname, epc.salary

from employee\_parent\_company epc, department d

where epc.deptid = d.deptid

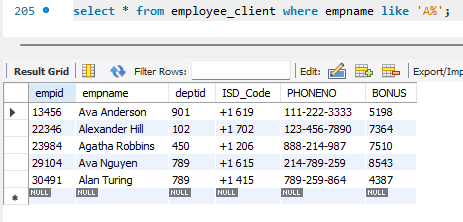
order by epc.salary desc

limit 5;



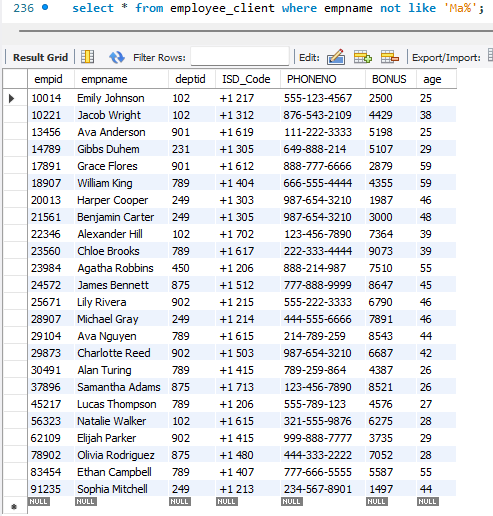
1. To fetch the client employee details whose name starts with ‘A’:

select \* from employee\_client where empname like 'A%';



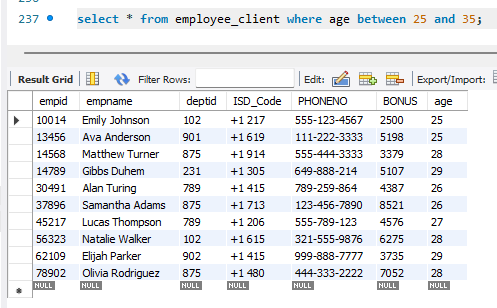
1. To fetch the client employee details whose name starts with ‘Ma:

select \* from employee\_client where empname not like 'Ma%';



1. To view the client employee details whose age is between 25 and 35:

select \* from employee\_client where age between 25 and 35;



1. Combine the country code and phone number of the employees:

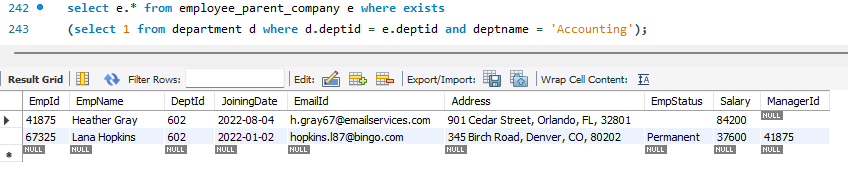
select empid, empname, deptid, concat(ISD\_Code,' ', PHONENO) as mobile\_no from employee\_client;



1. Fetch all accounting employee details using EXISTS operator

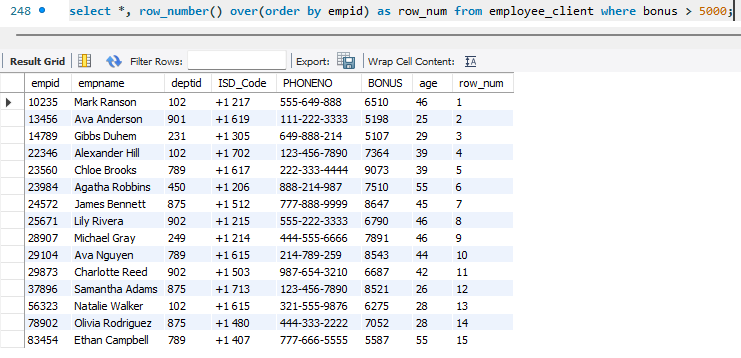
select e.\* from employee\_parent\_company e where exists

(select 1 from department d where d.deptid = e.deptid and deptname = 'Accounting');



1. Add row number to the result:

select \*, row\_number() over(order by empid) as row\_num from employee\_client where bonus > 5000;

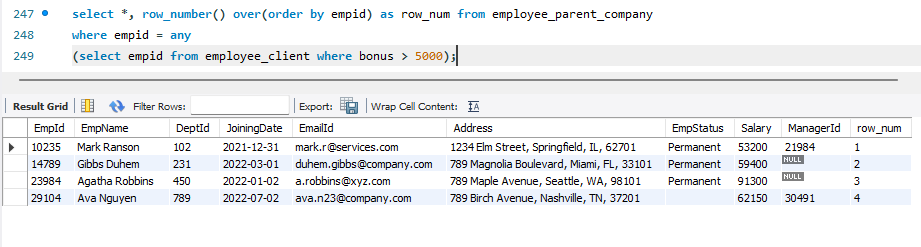


1. Fetch all employee records from parent table whose bonus > 5000

select \*, row\_number() over(order by empid) as row\_num from employee\_parent\_company

where empid = any

(select empid from employee\_client where bonus > 5000);

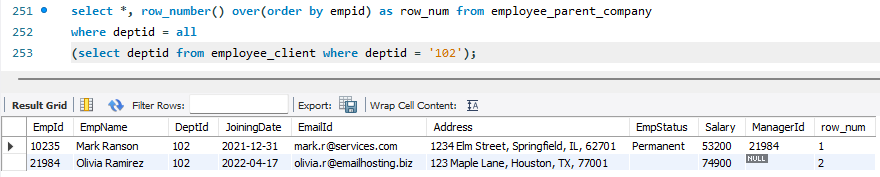


1. Fetch employee records from parent company whose dept id is 102:

select \*, row\_number() over(order by empid) as row\_num from employee\_parent\_company

where deptid = all

(select deptid from employee\_client where deptid = '102');



1. Map the retirement period of the client employees:

select Empid, empname, age,

case

when age >= 55 then "About to retire in the next 5 years"

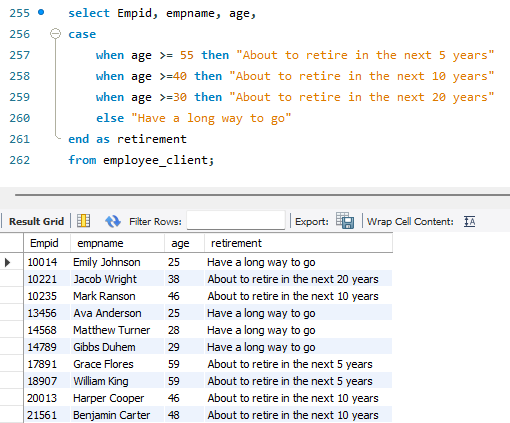
when age >=40 then "About to retire in the next 10 years"

when age >=30 then "About to retire in the next 20 years"

else "Have a long way to go"

end as retirement

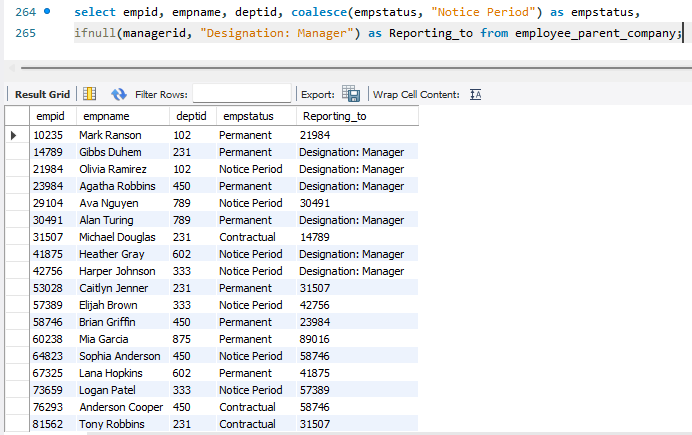
from employee\_client;



1. Display another text if null using COALESCE() and IFNULL() functions:

select empid, empname, deptid, coalesce(empstatus, "Notice Period") as empstatus,

ifnull(managerid, "Designation: Manager") as Reporting\_to from employee\_parent\_company;



1. STORED PROCEDURE:

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. So, if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it. You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

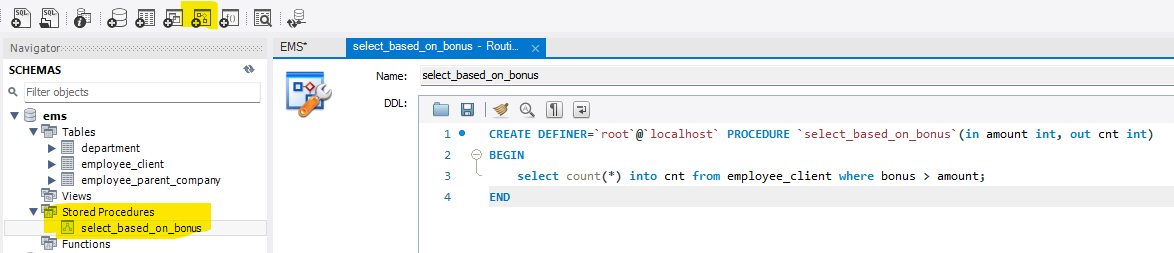
CREATE PROCEDURE `select\_based\_on\_bonus`(in amount int, out cnt int)

BEGIN

select count(\*) into cnt from employee\_client where bonus > amount;

END

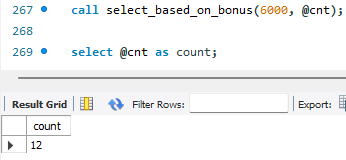
Click on ‘Apply’ below the ‘Routine’ window to apply the procedure to the database



To select the procedure go back to worksheet and use :

call select\_based\_on\_bonus(6000, @cnt);

select @cnt as count;



1. CTE (Common Table Expression):

A common table expression is a named temporary result set that exists solely within the execution scope of a single SQL statement, such as SELECT, INSERT, UPDATE, or DELETE.

Similar to a derived table, a common table expression (CTE) is not stored as an object and lasts only during the query execution.

Unlike a derived table, a common table expression (CTE) can be self-referencing (in the case of a recursive CTE) or referenced multiple times within the same query. Moreover, a CTE offers enhanced readability and performance compared to a derived table.

**MySQL CTE syntax**

The structure of a CTE includes the name, an optional column list, and a query that defines the CTE. After you define a CTE, you can use like a view in the SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement.

The following illustrates the basic syntax of a CTE:

WITH cte\_name (column\_list) AS ( query)

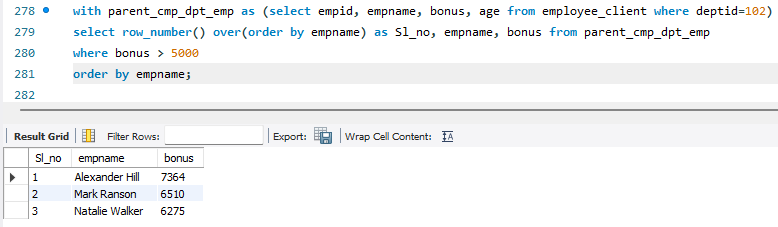
SELECT \* FROM cte\_name;

1. Basic CTE:

with parent\_cmp\_dpt\_emp as (select empid, empname, bonus, age from employee\_client where deptid=102)

select row\_number() over(order by empname) as Sl\_no, empname, bonus from parent\_cmp\_dpt\_emp

where bonus > 5000 order by empname;



Here:

1. First, define a CTE with the name parent\_cmp\_dpt\_emp that stores the employee id, name, bonus and age of employees in the 102 department. The defining query retrieves data from the employee\_client table.
2. Second, select the employee names whose bonus is greater than 5000 from the CTE.
3. Multiple CTE:

with emp\_2022 as (select empid, empname, deptid, salary

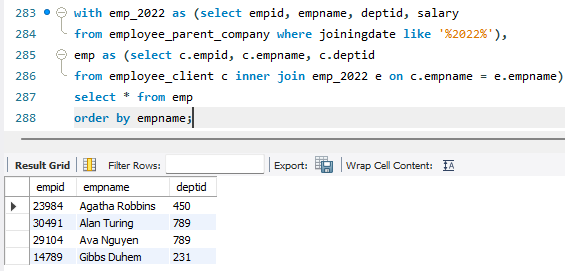
from employee\_parent\_company where joiningdate like '%2022%'),

emp as (select c.empid, c.empname, c.deptid

from employee\_client c inner join emp\_2022 e on c.empname = e.empname)

select \* from emp

order by empname;



Here:

1. CTE emp\_2022: Select empid, empname, deptid, salary columns, and include only employees who has joined in year 2022.
2. CTE emp: selects empid, empname, deptid by joining the employee\_parent\_company table with the emp\_2022 CTE based on the common column empname.
3. Main query: Select all columns from the emp CTE.
4. AUTO\_INCREMENT:

alter table department add column sl\_no int auto\_increment primary key;

1. Temporary Table:

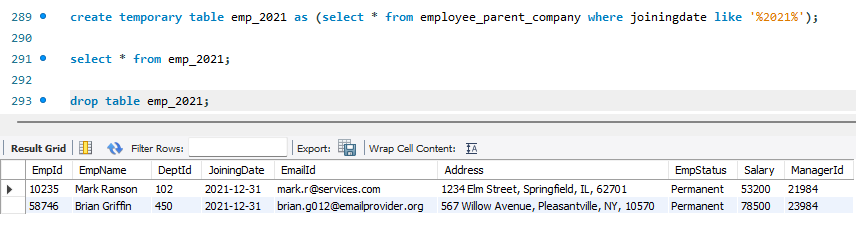
A MySQL temporary table has the following features:

* A temporary table is created by using CREATE TEMPORARY TABLE statement.
* MySQL removes the temporary table automatically when the session ends or the connection is terminated. Also, you can use the DROP TABLE statement to remove a temporary table explicitly when you are no longer using it.
* A temporary table is only available and accessible to the client that creates it. Different clients can create temporary tables with the same name without causing errors because only the client that creates the temporary table can see it. However, in the same session, two temporary tables cannot share the same name.
* It’s of best practice to avoid naming the temporary table with the same name as the permanent table.

create temporary table emp\_2021 as (select \* from employee\_parent\_company where joiningdate like '%2021%');

select \* from emp\_2021;

drop table emp\_2021;

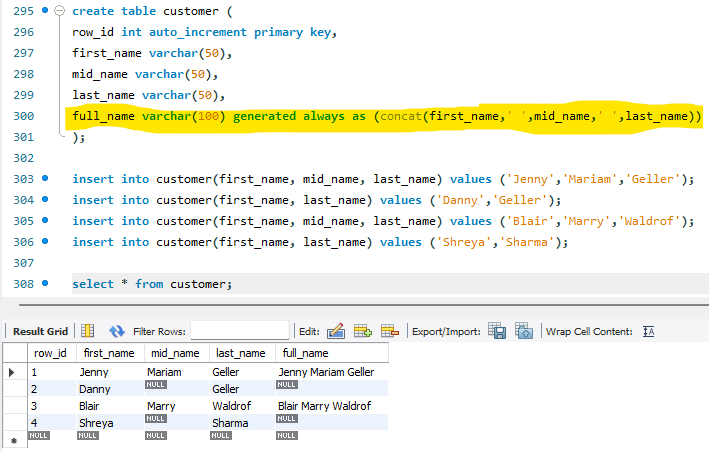


1. GENERATED COLUMN:

The values in the fullname column are computed on the fly when you query data from the contacts table.

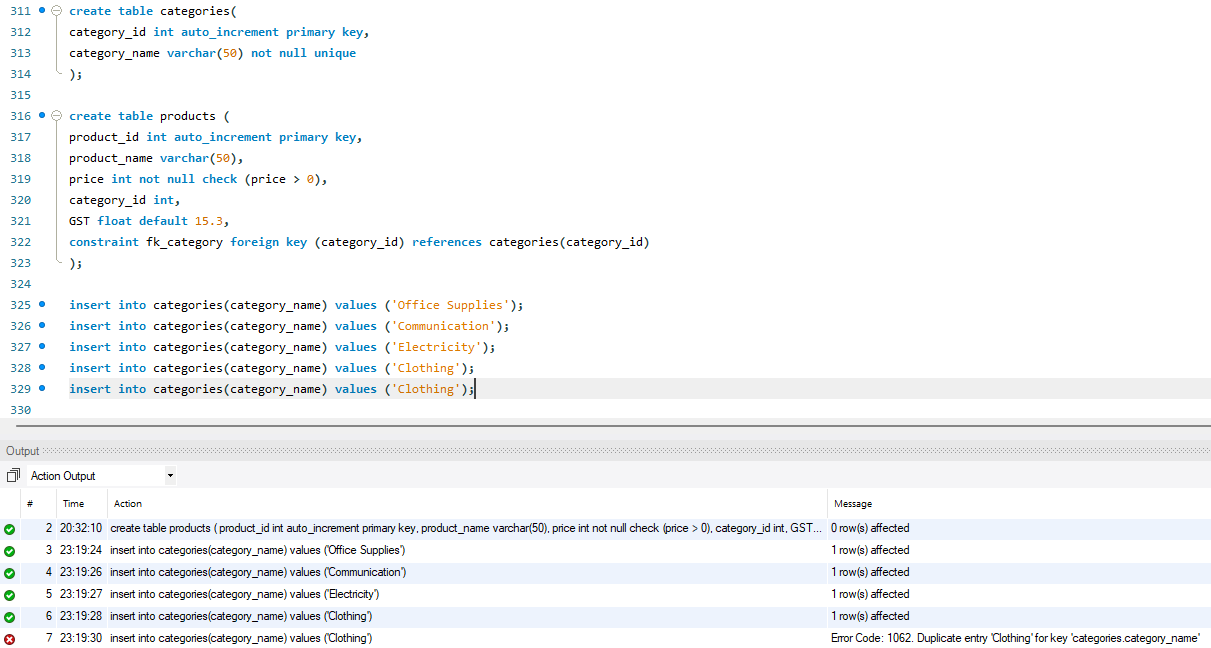
MySQL provides two types of generated columns: stored and virtual. The virtual columns are calculated on the fly each time data is read whereas the stored columns are calculated and stored physically when the data is updated.

Based on this definition, the fullname column that in the example is a virtual column.



1. SQL CONSTRAINTS:

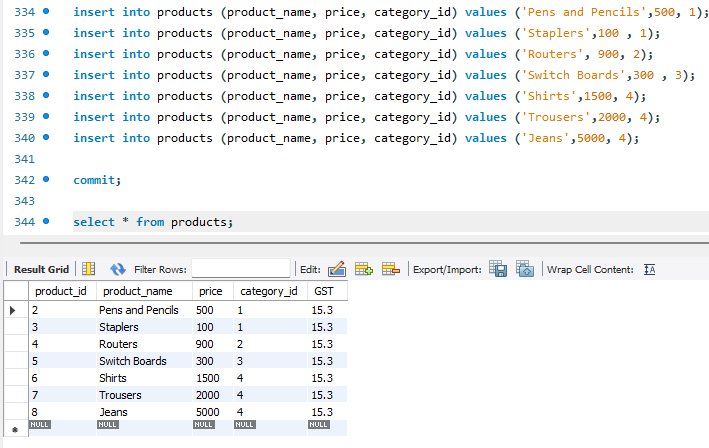
Here, I have created two tables with PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE, CHECK, DEFAULT constraints.



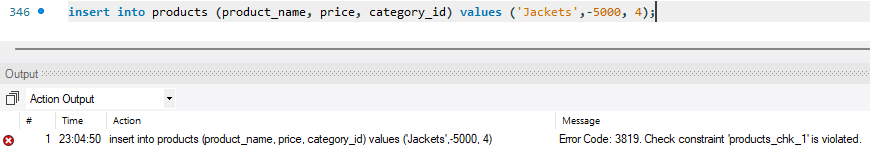
Here the insert fails as category\_name columns is marked to be UNIQUE



Here the insert on product table fails as category\_id is a foreign key to category table and category\_id 100 is not present on category table.



Here is an example of DEFAULT constraint on GST column



Here is an example of CHECK constraint where price should be > 0, otherwise the INSERT statement would fail.

1. INSERT ON DUPLICATE KEY UPDATE:

It updates an existing unique / primary key constraint column without throwing any error

