DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

<<PES1201800158>> <<Vibha Kurpad>>

The mini world I have chosen to represent is a parts distributor organization. More specifically the auto parts distributor model. While drawing the ER diagram, I have identified 8 strong entities and 1 weak entity. The strong entities in this model include the EMPLOYEE, CUSTOMER, CUSTOMER ORDER, DEPARTMENT, LOCATION, PART, SUPPLIER and JOB. The weak entity is derived from two strong entities, CUSTOMER ORDER and PART which is called LINE ITEM.

In this report the introduction details the various transactions which occur amongst the different entities. The data model section will include pictures of the ER diagram, description of the various relationships amongst the entities, and the data types I have chosen to represent each primary key.

The normalization section begins with drawing of the schema and identifying the various functional dependencies which exist amongst the attributes. This will help in identifying the primary and foreign keys in the relation.

The DDL section contains the scripts needed to create the tables. I have also included screenshots of the tables after inserting some data into them. This will be helpful in making sense of what the queries perform in the next section.

The trigger section contains the code for the trigger. To implement the trigger, I have used a procedure which gets called when the triggering event occurs. In this case whenever a new employee's data is entered into the employee table or an existing employee's salary is being updated, a record will be created in the mgr_alert table notifying which employees salary got updated and what the new amount is. It also mentions the manager's employee id.

The SQL queries section contains a few lines detailing what the query does and what result we should expect. For each query I have attached a screenshot having the code for the query as well as the result of executing it. Joins have been performed such as a self-join on the employee relation to get a list of employees and their managers. An update on a table with a check constraint is performed to verify whether the check violation error appears. Correlated and non – correlated queries have been performed.

To conclude this report, we discuss some limitations and further enhancements which can be done to the model. By making certain changes to the data model we can keep track of a salesperson's performance and can keep track of sales by their geographical location.

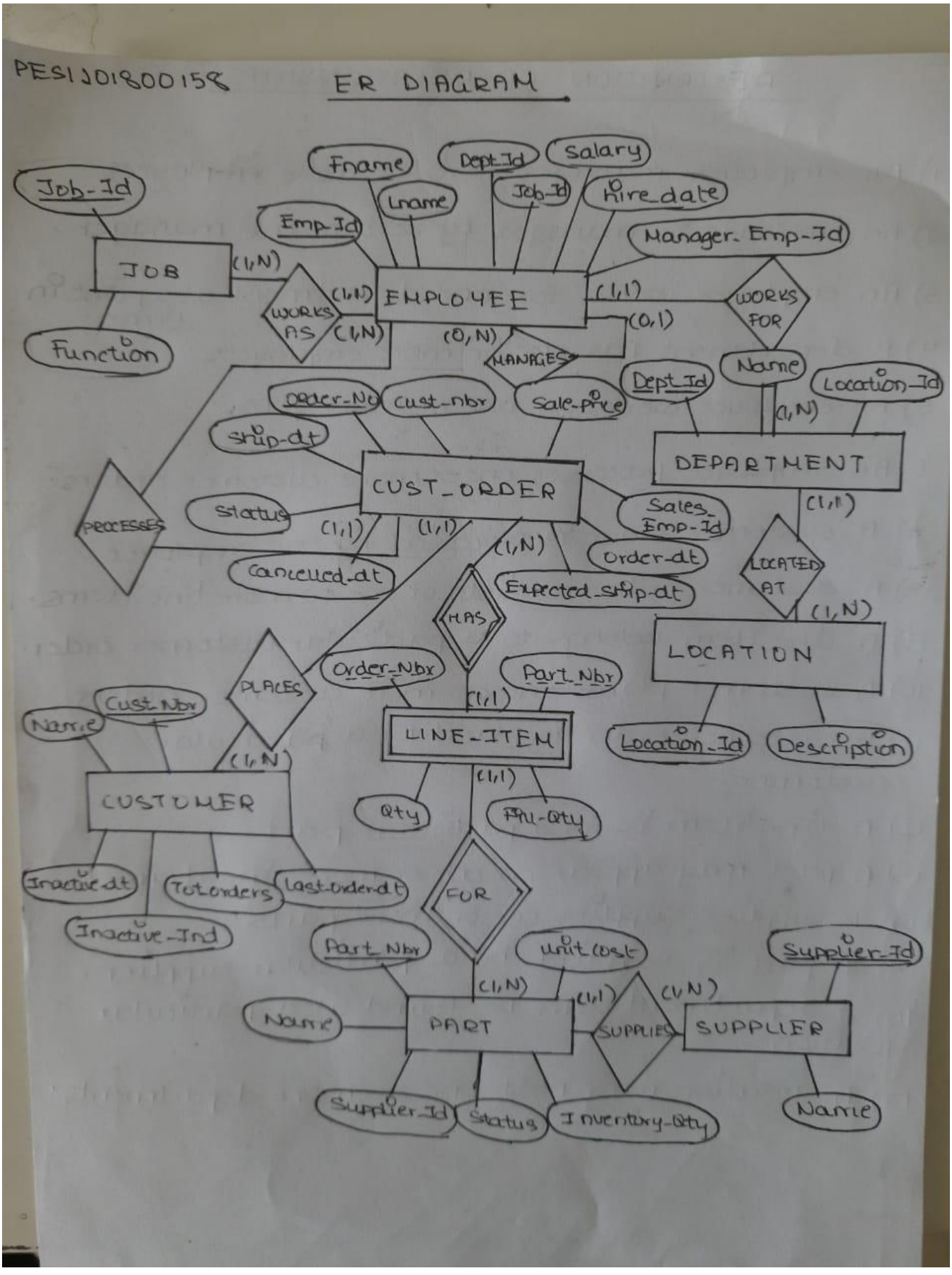
Introduction

The miniworld I have chosen to represent is a parts distributor organization. More specifically the auto parts distributor model. In this model we have identified a few important entities. First are the employees of the auto part distributor firm. There are various jobs which have been taken on. These jobs fall under four different departments. They are Accounting, Sales, Operations and Research. These departments are situated in different locations.

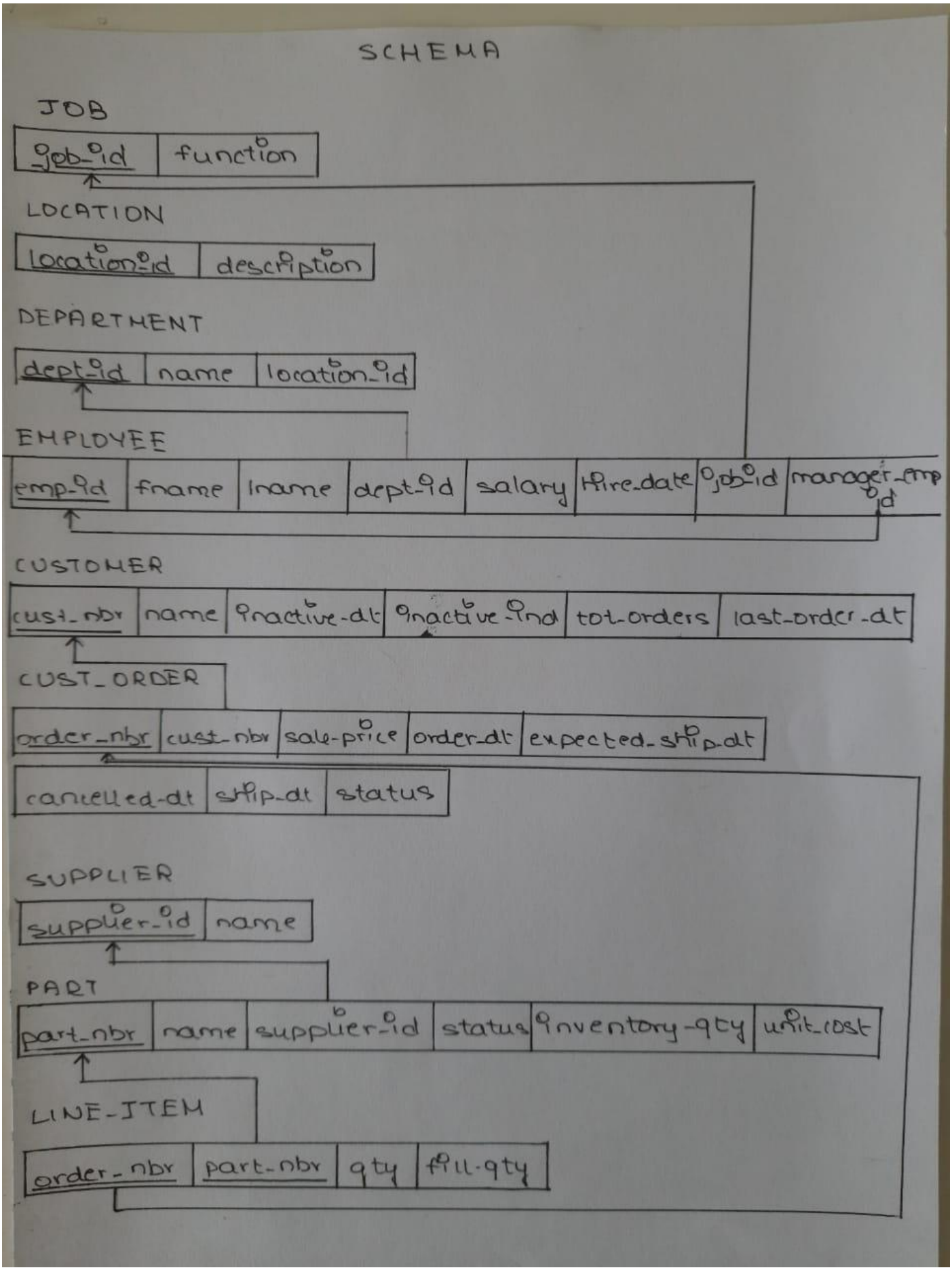
This auto distributor firm has partnered up with 6 suppliers to distribute these parts to various customers. The parts the distributor has an inventory may either be frozen, active or obsolete. The distributor engages with many customers. Some of these customers have been inactive and most of them remain active. The ones who remain active place orders with the firm to have certain parts of a particular quantity delivered. These orders maybe shipped on time or might be slightly delayed. Customers also have the option to cancel orders previously placed.

Data Model

ER Diagram:



Schema:


SCHEMA
JOB
job_id
function
LOCATION
location_id
description
DEPARTMENT
dept_id
name
location_id
EMPLOYEE
emp_id
fname
lname
dept_id
salary
hire_date
job_id
manager_emp_id
CUSTOMER
cust_nbr
name
reactive_dt
inactive_ind
tot_orders
last_order_dt
CUST_ORDER
order_nbr
cust_nbr
sale_price
order_dt
expected_ship_dt
cancelled_dt
ship_dt
status
SUPPLIER
supplier_id
name
PART
part_nbr
name
supplier_id
status
inventory_qty
unit_cost
LINE_ITEM
order_nbr
part_nbr
qty
fill_qty

JOB: For this relation the primary key is the job_id as it uniquely identifies each row in this relation. The datatype chosen for this key is number(3) to accommodate for the creation of new jobs in the future.

LOCATION: For this relation the primary key is location_id as it uniquely identifies each row in this relation. The datatype chosen for this key is varchar2(3). This will allow for the inclusion of more locations in the future. The values of these keys begin with the letter L to distinguish the values from other primary keys.

DEPARTMENT: For this relation the primary key is department_id as it uniquely identifies each row in this relation. The datatype chosen for this key is varchar2(4). This will allow for the creation of new departments in the future if required. The values of these keys begin with the letter D to distinguish the values from other primary keys.

EMPLOYEE: For this relation the primary key is emp_id as it uniquely identifies each row in this relation. The datatype chosen for this key is varchar2(6). This will allow for the insertion of new employee data in the future if required. The values of these keys begin with the letters EMP to distinguish the values from other primary keys.

CUSTOMER: For this relation the primary key is cust_nbr as it uniquely identifies each row in this relation. The datatype chosen for this key is varchar2(4). This will allow for the inclusion of more customers in the future if required. The values of these keys begin with the letter C to distinguish the values from other primary keys.

CUST_ORDER: For this relation the primary key is order_nbr as it uniquely identifies each row in this relation. The datatype chosen for this key is varchar2(4). This will allow for the creation of new orders as and when customers place them in the future if required. The values of these keys begin with the letter O to distinguish the values from other primary keys.

SUPPLIER: For this relation the primary key is supplier_id as it uniquely identifies each row in this relation. The datatype chosen for this key is number(2). This will allow for the inclusion of new suppliers in the future if required.

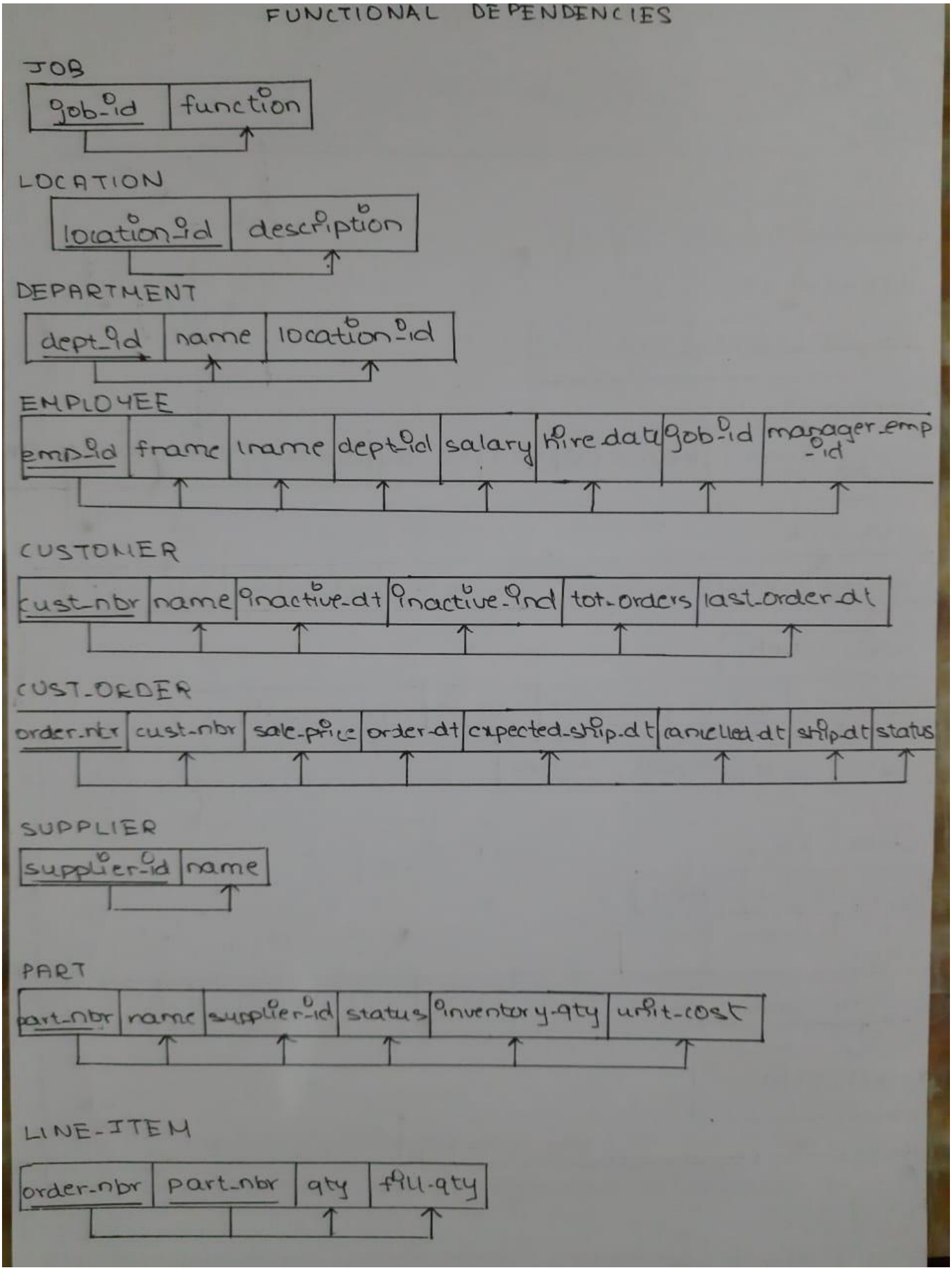
PART: For this relation the primary key is part_nbr as it uniquely identifies each row in this relation. The datatype chosen for this key is varchar2(3). This will allow for the inclusion of more parts in the future if suppliers decide to supply new parts. The values of these keys begin with the letter P to distinguish the values from other primary keys.

LINE_ITEM: For this relation the primary key is a combination of two attributes, order_nbr and part_nbr. This is so because the combination of these two columns values uniquely identifies a row in this relation. Order_nbr and part_nbr are both foreign keys which derive their values from the relations cust_order and part respectively. The data types for the two key attributes are mentioned above.

DESCRIPTION OF RELATIONSHIPS

1) An employee manages zero or more employees.
2) An employee is managed by zero or one manager.
3) An employee works for one department at a point in time.
4) A department has one or more employees.
5) An employee works on one ~~or more~~ jobs.
6) An employee processes one or more customer orders.
7) A customer order is processed by one employee.
8) A customer order consists of one or more line items.
9) A line item belongs to a particular customer order.
10) A customer places one or more customer orders.
11) A customer order is placed by a particular customer.
12) A line item is for a particular part.
13) A part may appear on one or more line items.
14) A supplier supplies one or more parts.
15) A part is supplied by a particular supplier.
16) A department can be found at a particular location.
17) A location may host one or more departments.

FD and Normalization



For the relation JOB; job_id -> function, hence the primary key is job_id.

For the relation LOCATION; location_id -> description, hence the primary key is location_id.

For the relation DEPARTMENT; dept_id -> name, location_id, hence the primary key is dept_id.

For the relation EMPLOYEE; emp_id -> fname, lname, dept_id, salary, hire_date, job_id, manager_emp_id, hence the primary key is emp_id.

For the relation CUSTOMER; cust_nbr -> name, inactive_dt, inactive_ind, tot_orders, last_order_dt, hence the primary key is cust_nbr.

For the relation CUST_ORDER; order_nbr -> cust_nbr, sale_price,order_dt, ship_dt, expected_ship_dt, cancelled_dt, status, hence the primary key is order_nbr.

For the relation SUPPLIER; supplier_id -> name, hence the primary key is supplier_id.

For the relation PART; part_nbr -> name, supplier_id , status, inventory_qty, unit_cost, hence the primary key is part_nbr.

For the relation LINE_ITEM; {order_nbr, part_nbr} -> qty, fill_qty, hence the primary key is order_nbr and part_nbr.

NORMALIZATION:

Since I followed the approach of developing an ER Model and converting it into a schema, the resulting relations are in third normal form.

1nf is violated if the attributes qty and fill_qty of the line_item table are a part of the cust_order table. This results in repeating groups which is a violation of the first normal form.

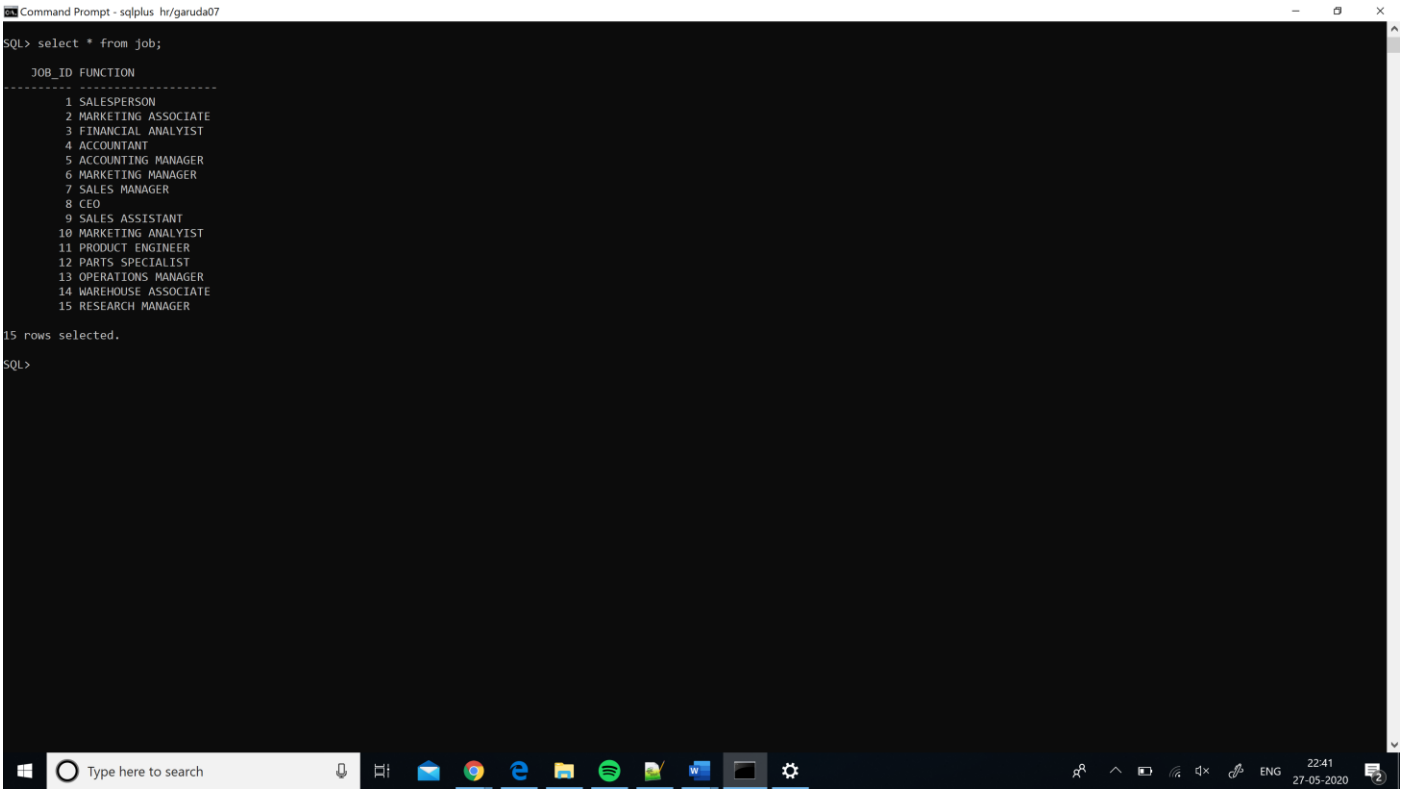
2nf is violated if the part name is added to the line_item relation. This is because a non-key attribute i.e., the part name is only dependent on a subset of the primary key (part name is only dependent on the part_nbr and not order_nbr). To be in 2nf the non key attributes must depend on the entire primary key.

3nf is violated if you add the department name along with department number in employee table because this would introduce transitive dependencies. The attribute department name (name) is dependent on the department id (dept_id) which is a non-key attribute. Dept_id is determined by employee id (emp_id).

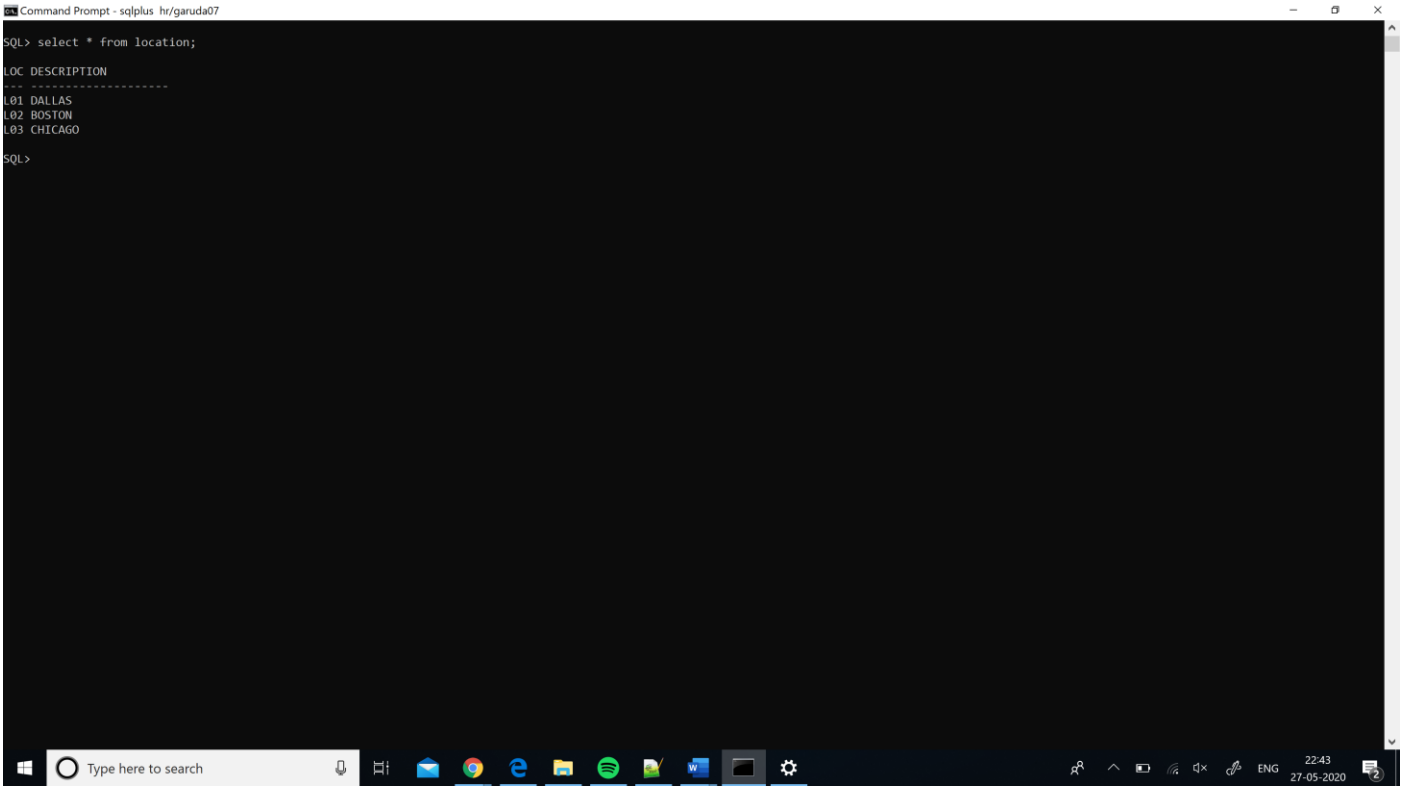
DDL

Below are the create table scripts:

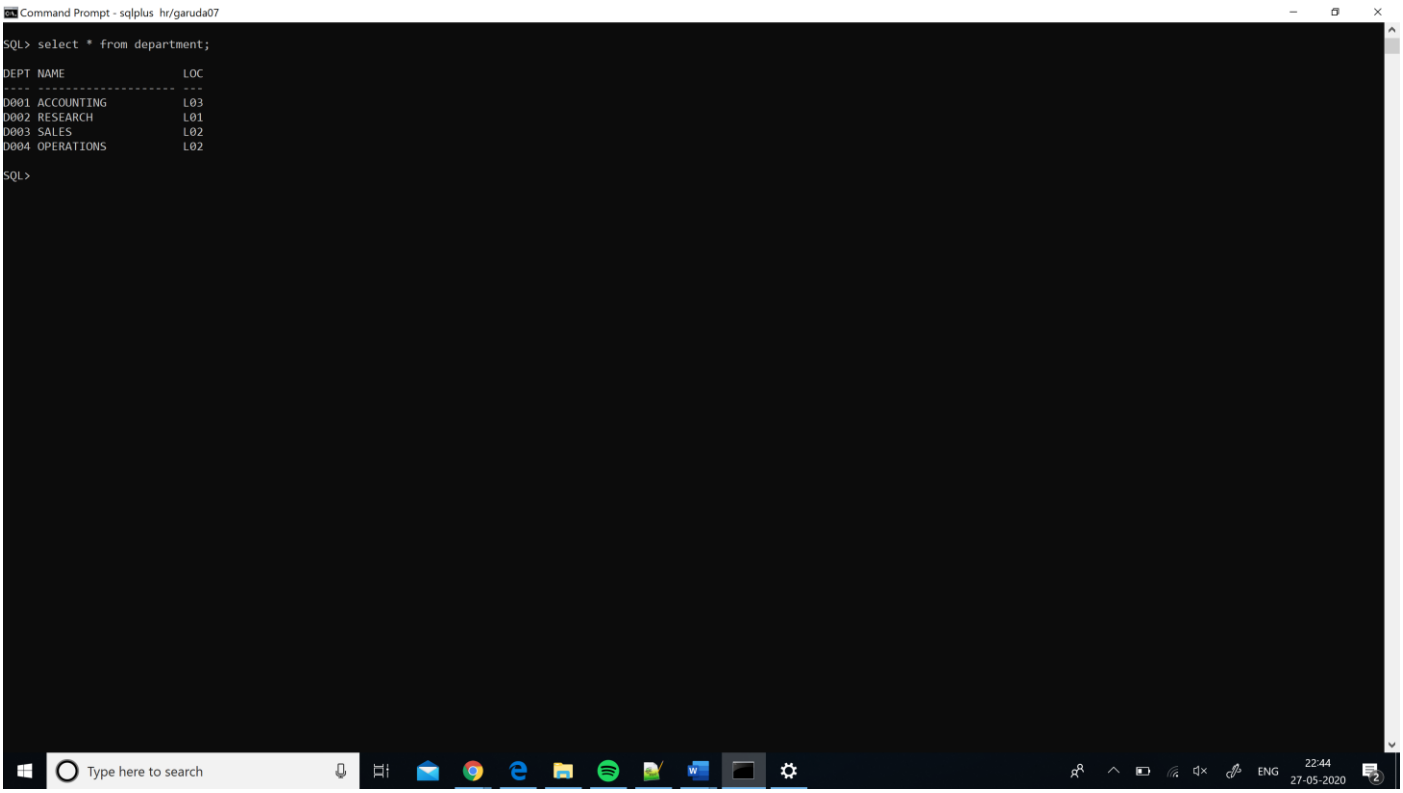
```
create table job(job_id number(3) primary key,
        function varchar2(20) not null
  )
;
```




```
create table location(location_id varchar2(3) primary key,
        description varchar2(20) not null
  )
;
```

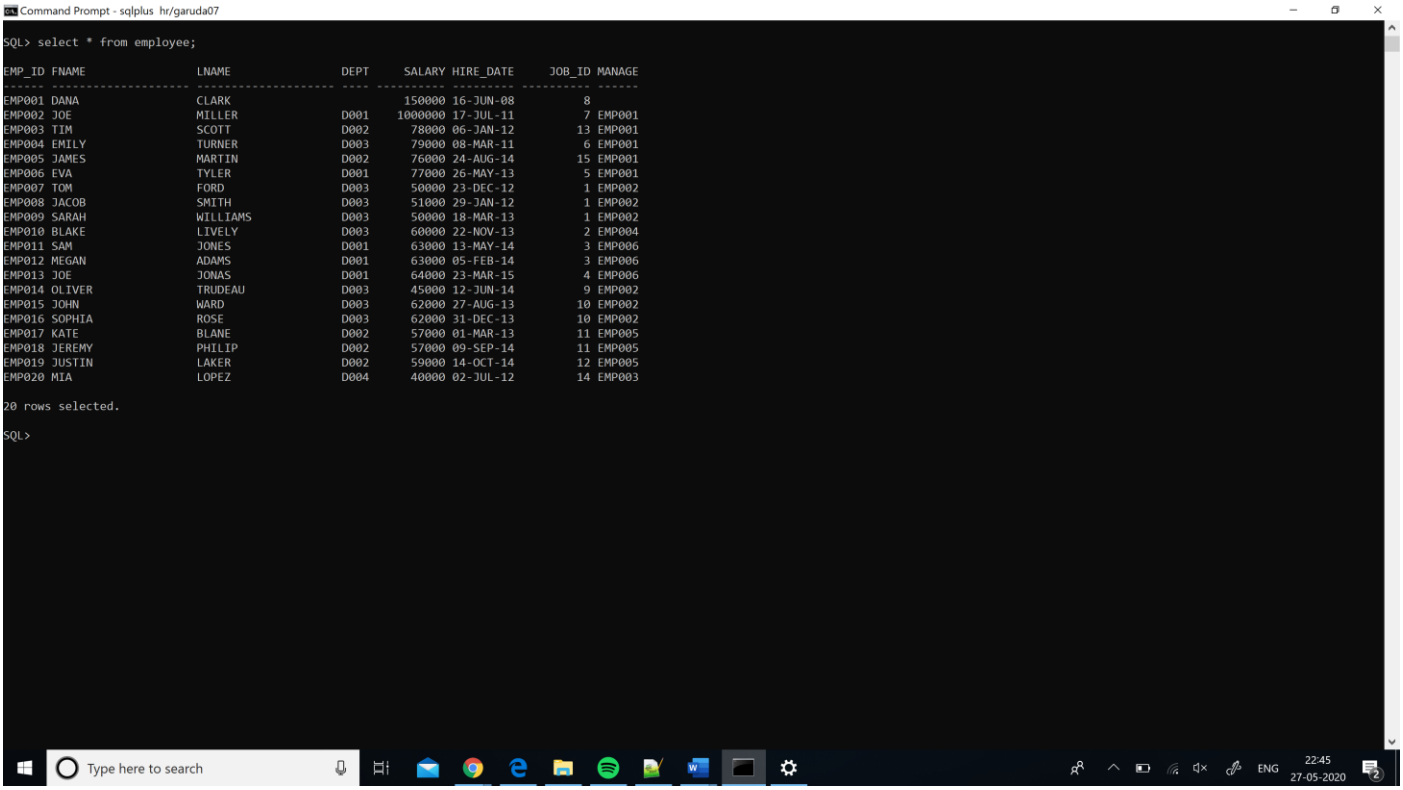



```
create table department(dept_id varchar2(4) primary key,
        name varchar2(20) not null,
        location_id varchar2(3) not null,
        constraint loc_id_fk foreign key(location_id)
        references location(location_id)
   )
;
```

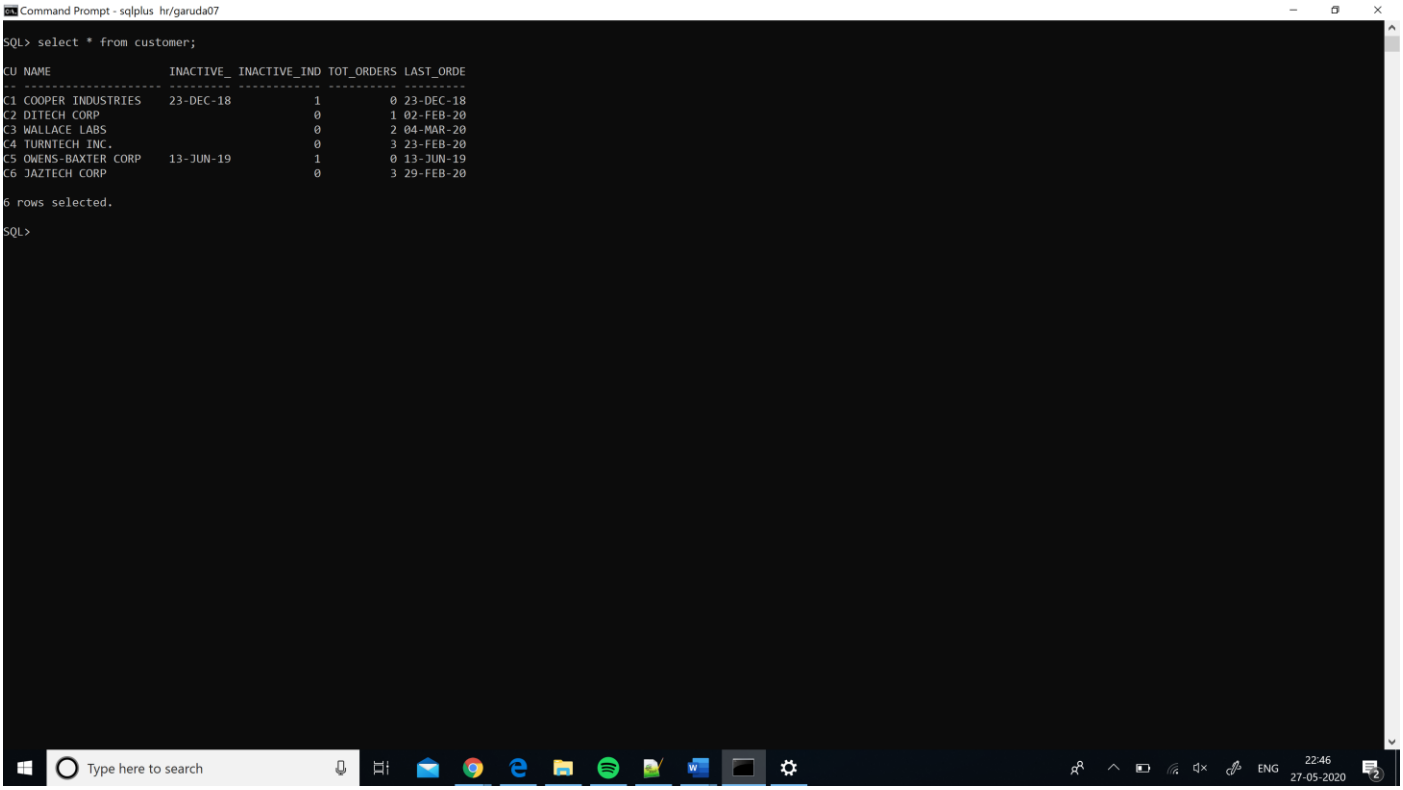



```
create table employee(emp_id varchar2(6) primary key,
        fname varchar2(20) not null,
        lname varchar2(20) not null,
```

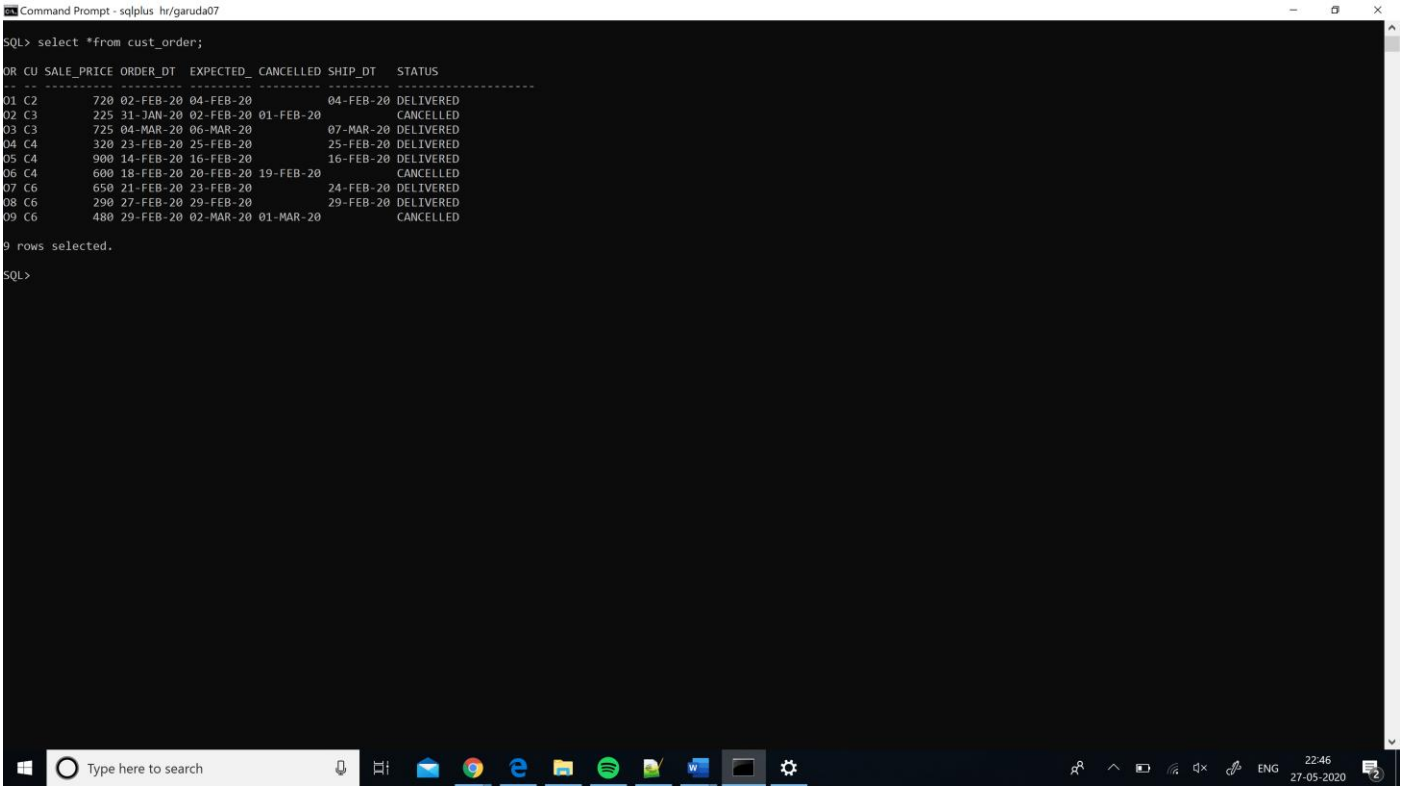
```
        dept_id varchar2(4) null,
        salary number(7) not null,
        hire_date date not null,
        job_id number(3) not null,
        manager_emp_id varchar2(6),
        constraint dept_id_fk foreign key(dept_id)
        references department(dept_id),
        constraint job_id_fk foreign key(job_id)
        references job(job_id),
        constraint mngr_id_fk foreign key(manager_emp_id)
        references employee(emp_id)
   )
;
```



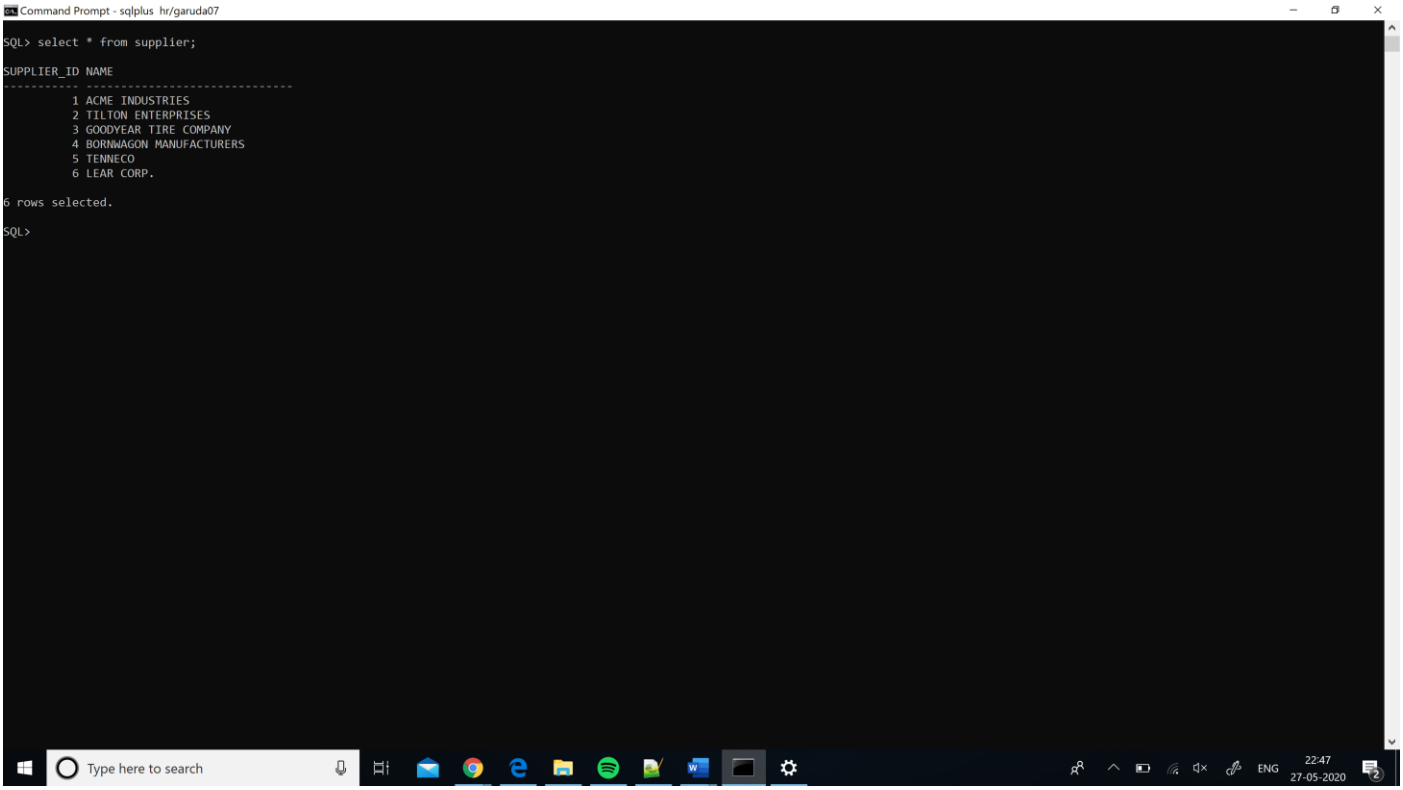

```
create table customer(cust_nbr varchar2(4) primary key,
        name varchar2(20) not null,
        inactive_dt date,
        inactive_ind number(1) not null,
        tot_orders number(5),
        last_order_dt date
  )
;
```




```
create table cust_order(order_nbr varchar2(4) primary key,
        cust_nbr varchar2(2) not null,
        sale_price number(9,2) not null,
        order_dt date not null,
        expected_ship_dt date,
        cancelled_dt date,
        ship_dt date,
        status varchar2(20) not null,
        constraint cust_nbr_fk foreign key(cust_nbr)
        references customer(cust_nbr),
        constraint date_check check(order_dt < cancelled_dt)
        )
;
```

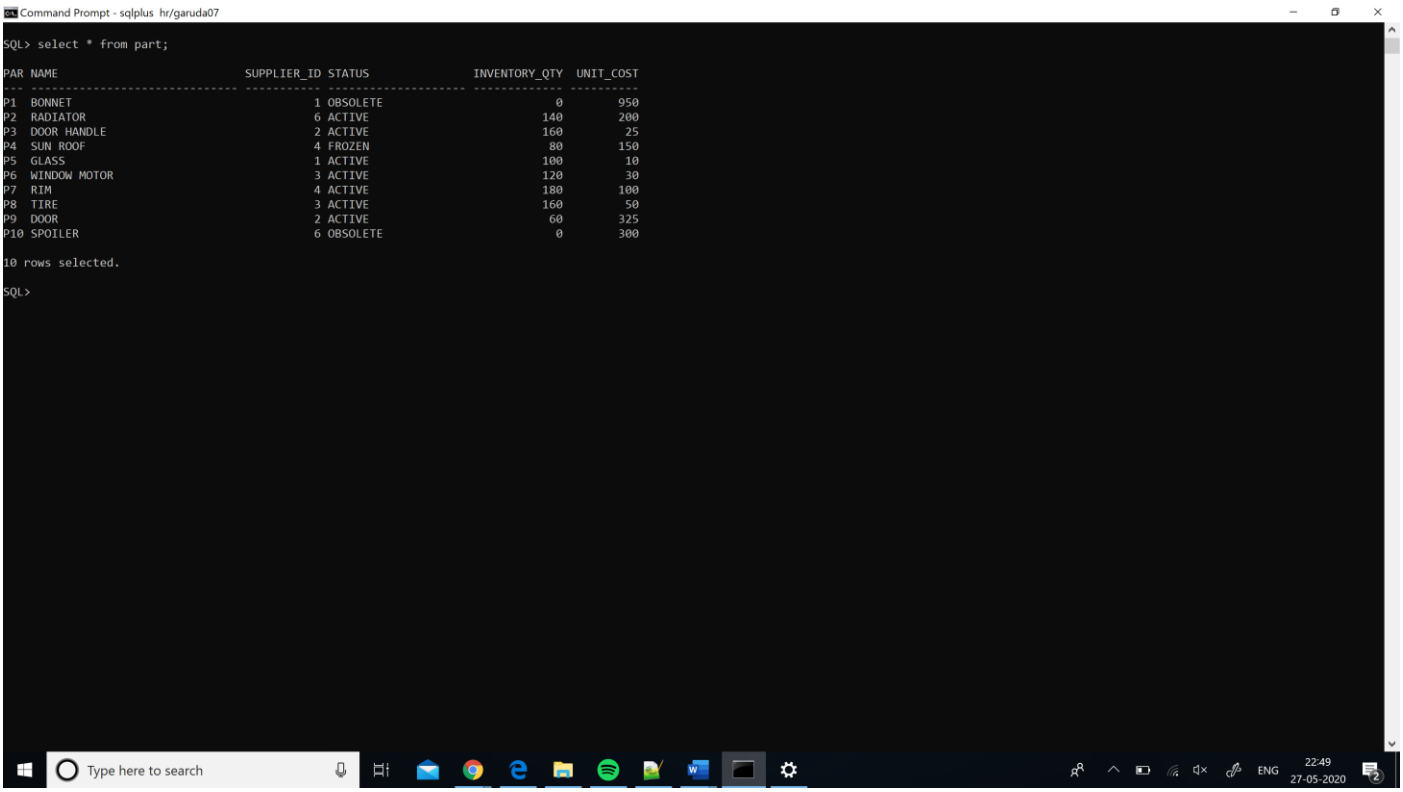



```
create table supplier(supplier_id number(2) primary key,
        name varchar2(30) not null,
        constraint valid_sup check(supplier_id between 1 and 99)
        )
;
```

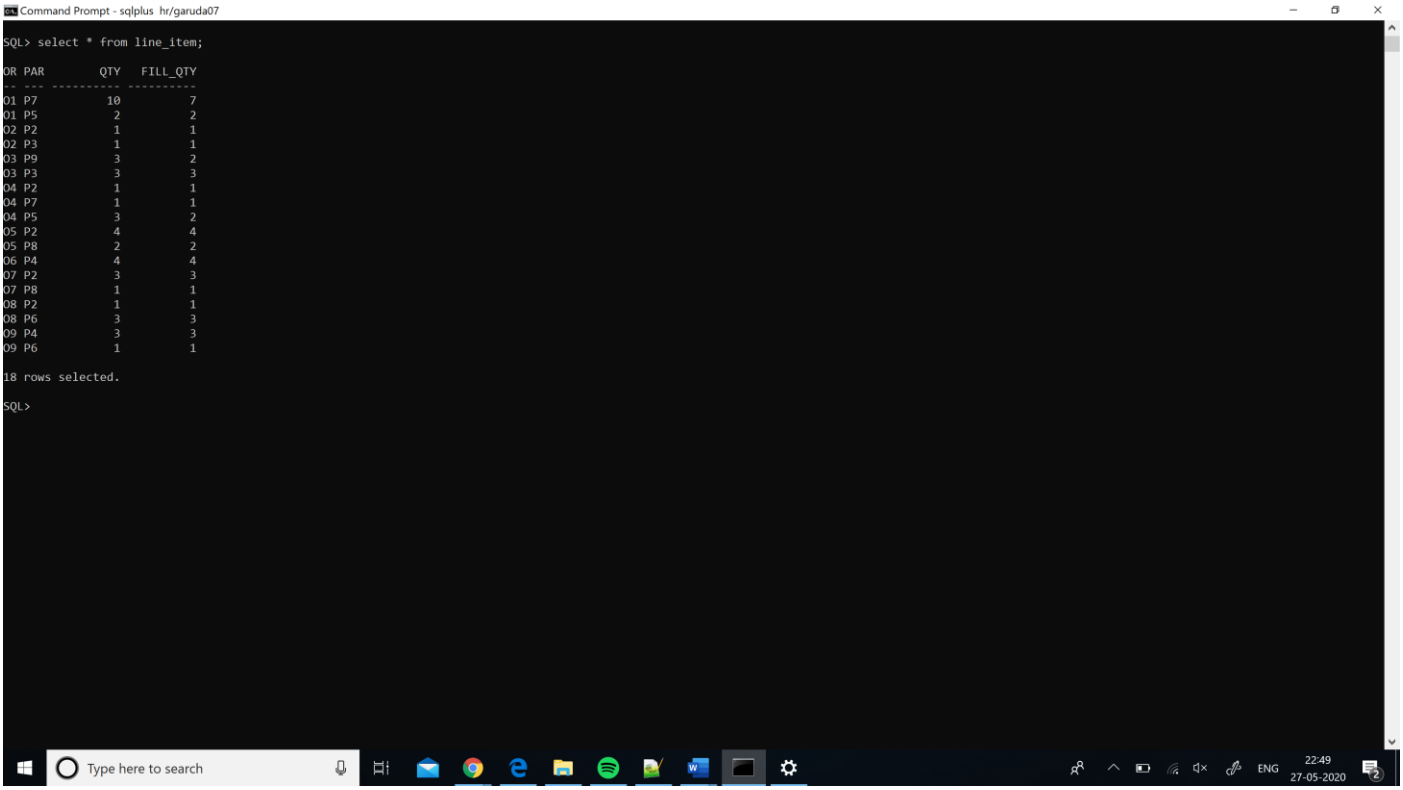



```
create table part(part_nbr varchar2(3)primary key,
        name varchar2(30) not null,
        supplier_id number(2) not null,
        status varchar2(20) not null,
```

```
        inventory_qty number(5),
        unit_cost number(8,2) not null,
        constraint supplier_id_fk foreign key(supplier_id)
        references supplier(supplier_id)
        )
;
```




```
create table line_item(order_nbr varchar2(2),
        part_nbr varchar2(3),
        qty number(5) not null,
        fill_qty number(5) not null,
        constraint line_item_pk primary key (order_nbr,part_nbr),
        constraint order_nbr_fk foreign key(order_nbr)
        references cust_order(order_nbr),
        constraint part_nbr_fk foreign key(part_nbr)
        references part(part_nbr),
        constraint valid_qty check(fill_qty <= qty)
        )
;
```


Command Prompt - sqlplus hr/garuda07
SQL> select * from line_item;
OR PAR QTY FILL_QTY
-- --- ---------- ----------
01 P7 10 7
01 P5 2 2
02 P2 1 1
02 P3 1 1
03 P9 3 2
03 P3 3 3
04 P2 1 1
04 P7 1 1
04 P5 3 2
05 P2 4 4
05 P8 2 2
06 P4 4 4
07 P2 3 3
07 P8 1 1
08 P2 1 1
08 P6 3 3
09 P4 3 3
09 P6 1 1
18 rows selected.
SQL>
Type here to search
ENG
22:49
27-05-2020

Triggers

In any business model, it is important to keep track of new employees' salary, or if an existing employee's salary has been updated. This is because the manager must have a record of all the employees' salary subordinate to him. He can easily review their salaries with the help of this trigger and procedure.

In this trigger, whenever a new employee's data is being inserted, or an existing employee's salary is being updated, an alert will be created and to keep track of this a record is added to the mgr_alert table. This table contains the information of the employee's manager id, the employee's id and the updated amount.

To carry out this trigger a procedure called logerr2 is first created. This takes care of inserting the data into the mgr_alert table. Then the trigger is defined.

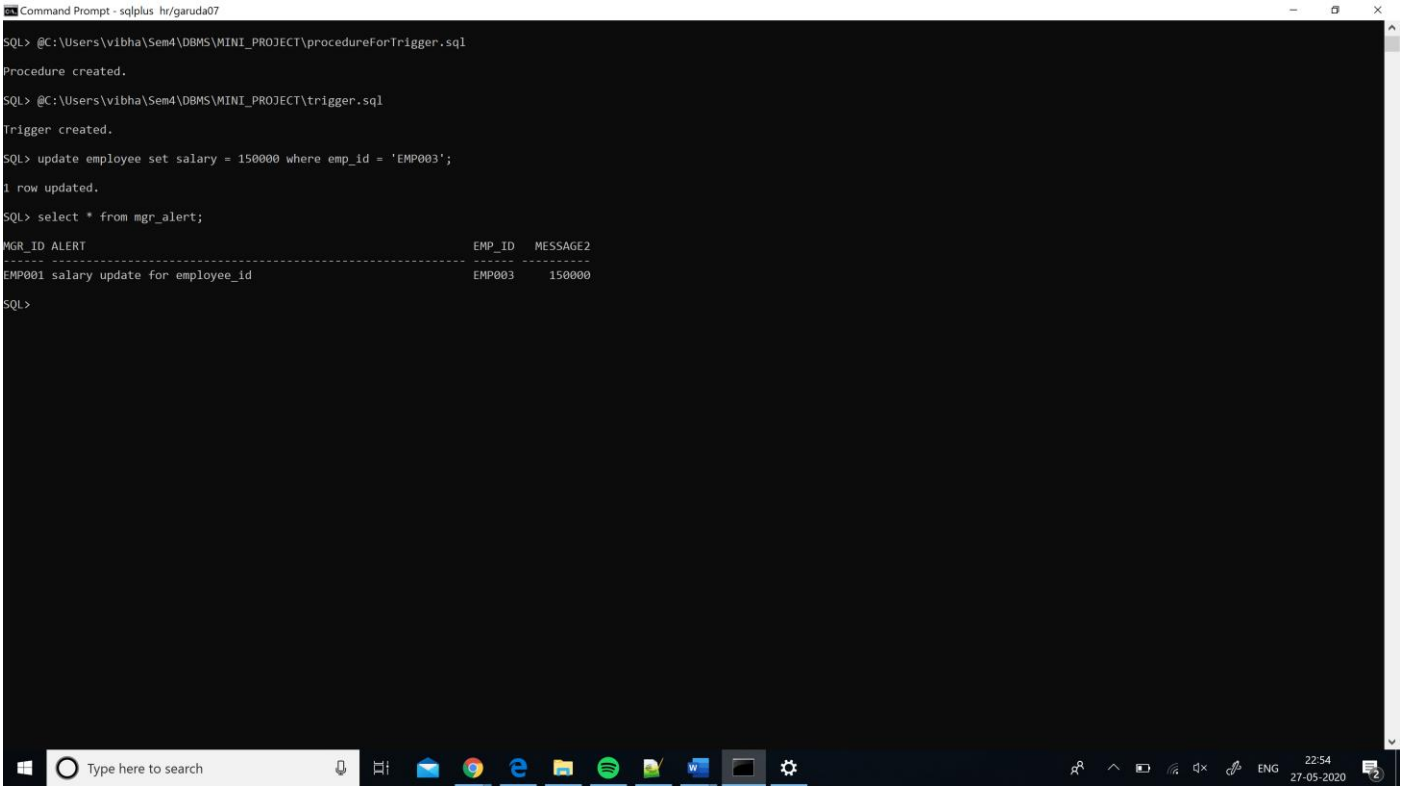
Code for Procedure:

```
create or replace procedure log_err2(mgr_id in varchar2,emp_id in varchar2,emp_sal in number)
as pragma autonomous_transaction;
begin
  insert into mgr_alert values(mgr_id, 'salary update for employee_id', emp_id, emp_sal);
  commit;
exception
  when others then rollback;
  raise;
end;
```

Code for Trigger:

```
create or replace trigger salary_error
before insert or update of salary
on employee
for each row
declare
do_nothing exception;
begin
if((INSERTING OR UPDATING)) then
        log_err2(:new.manager_emp_id,:new.emp_id,:new.salary);
end if;
end;
```

Screenshot of execution:

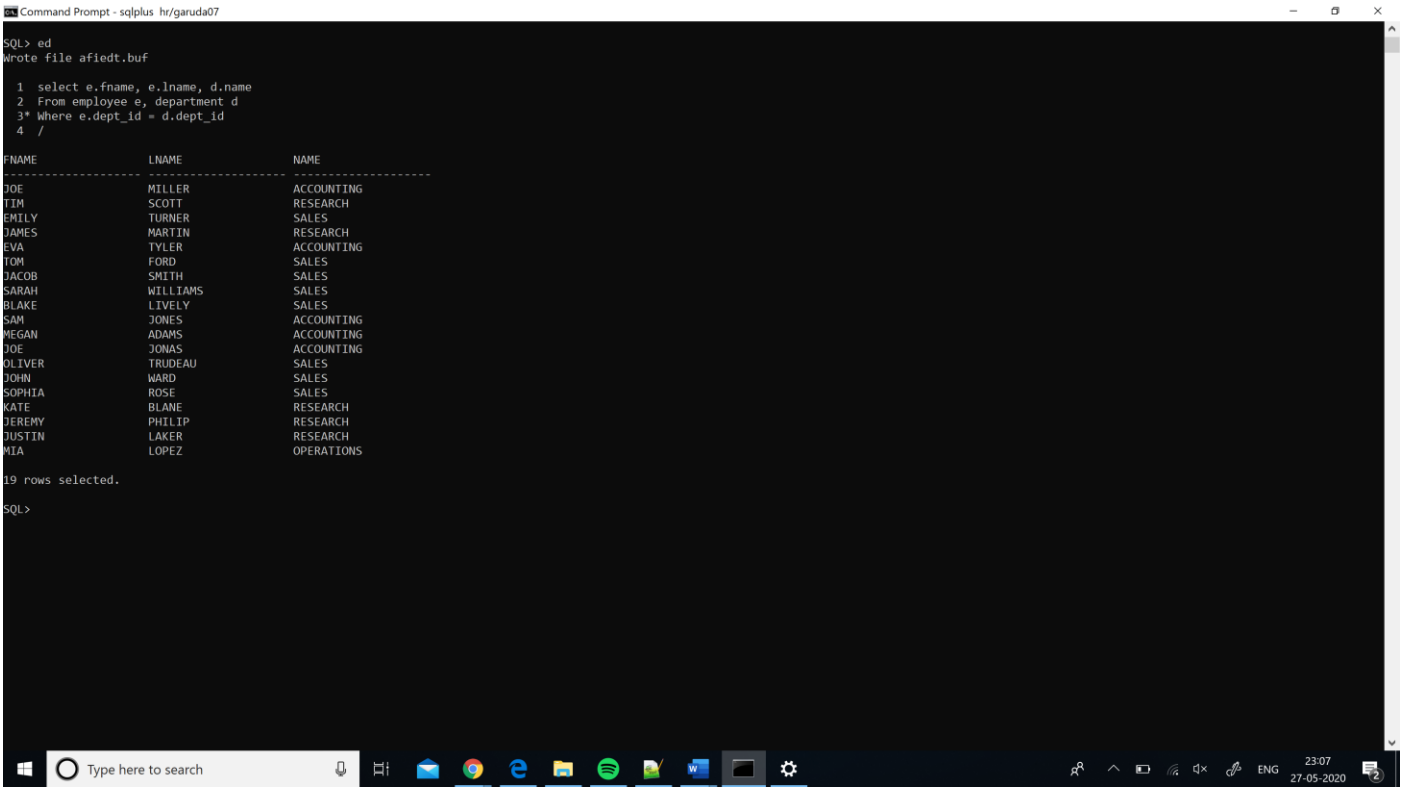


The employee with employee id EMP002 had their salary updated to 1,000,000 which is reflected in the mgr_alert table.

SQL Queries

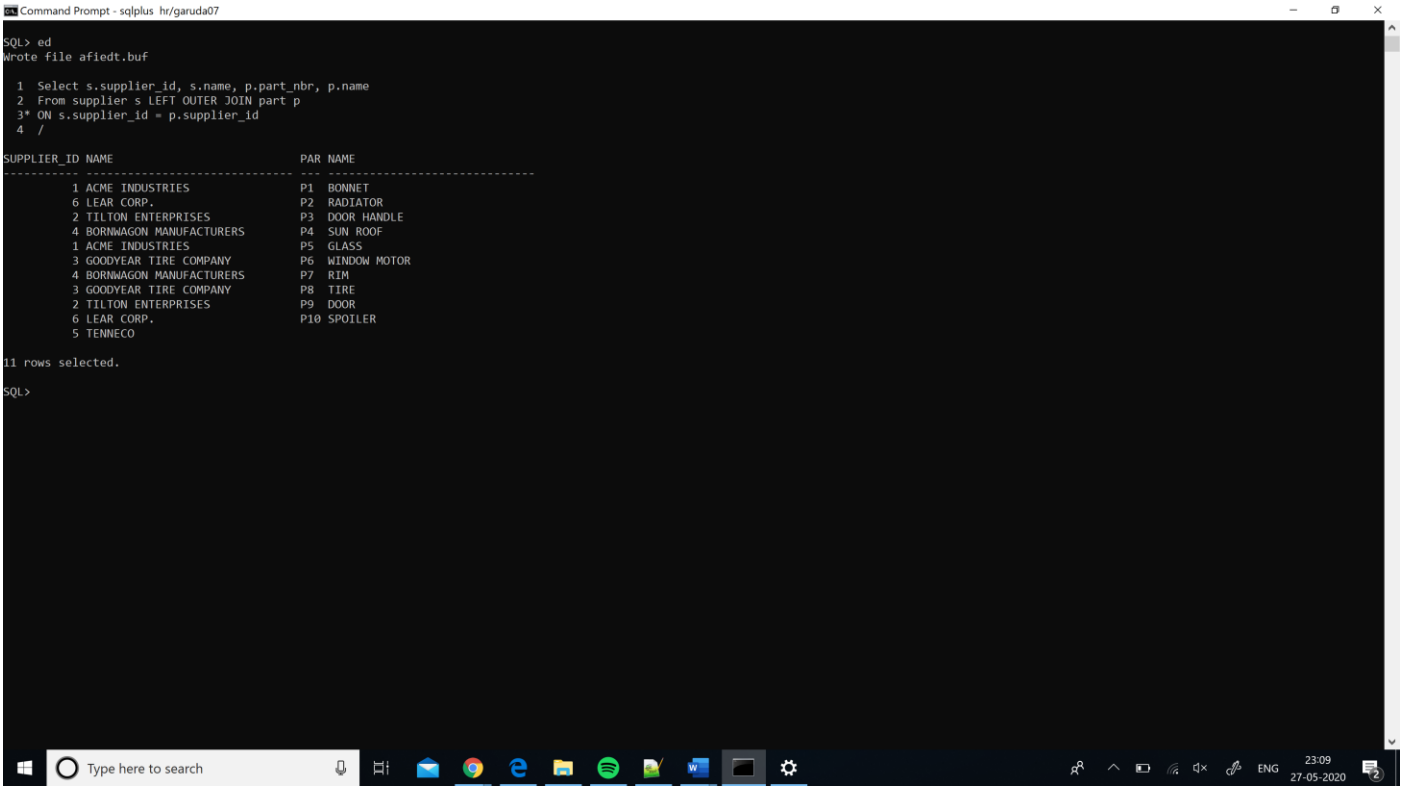
1) Inner Join: Gives a list of all the employees and the department they work for.

```
select e.fname, e.lname, d.name
From employee e, department d
Where e.dept_id = d.dept_id
```



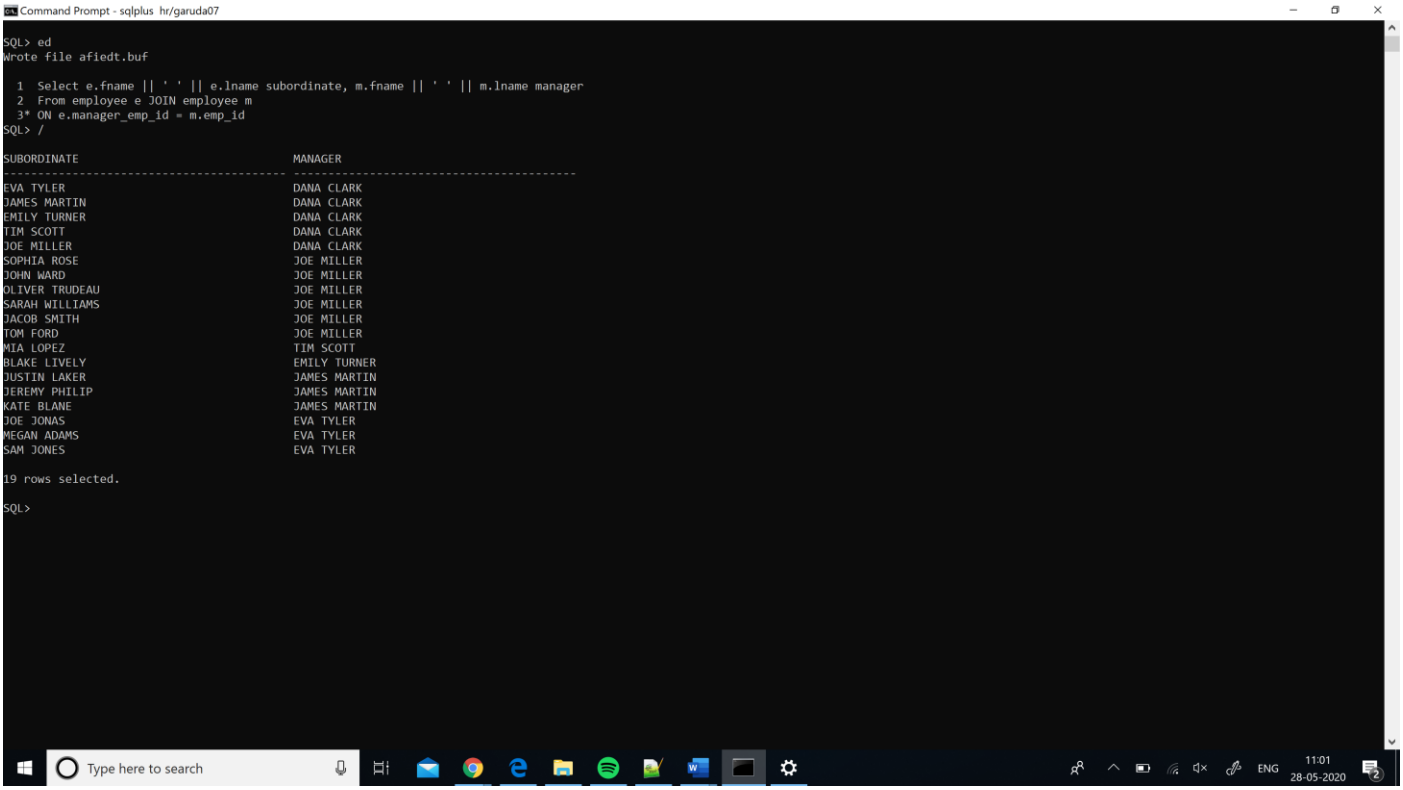
2) Outer Join: Gives a list of suppliers who supply and don't supply parts. If the supplier supplies parts, give the name and part_id of the parts which are supplied.

```
Select s.supplier_id, s.name, p.part_nbr, p.name
From supplier s LEFT OUTER JOIN part p
ON s.supplier_id = p.supplier_id
```



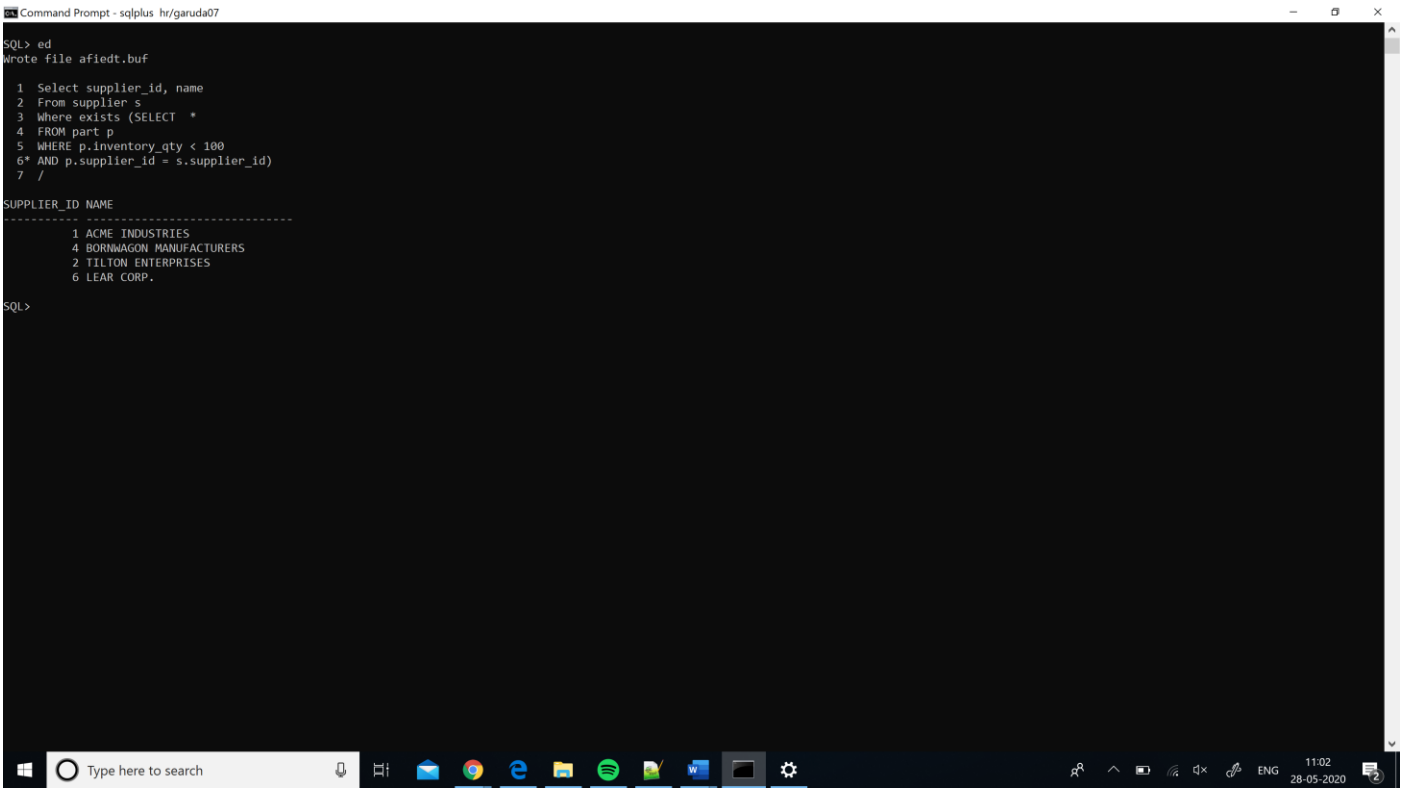
3) Self-Join: Gives a list of all the employees and their manager their manager's name.

Select e.fname || ' ' || e.lname subordinate, m.fname || ' ' || m.lname manager
From employee e JOIN employee m
ON e.manager_emp_id = m.emp_id;



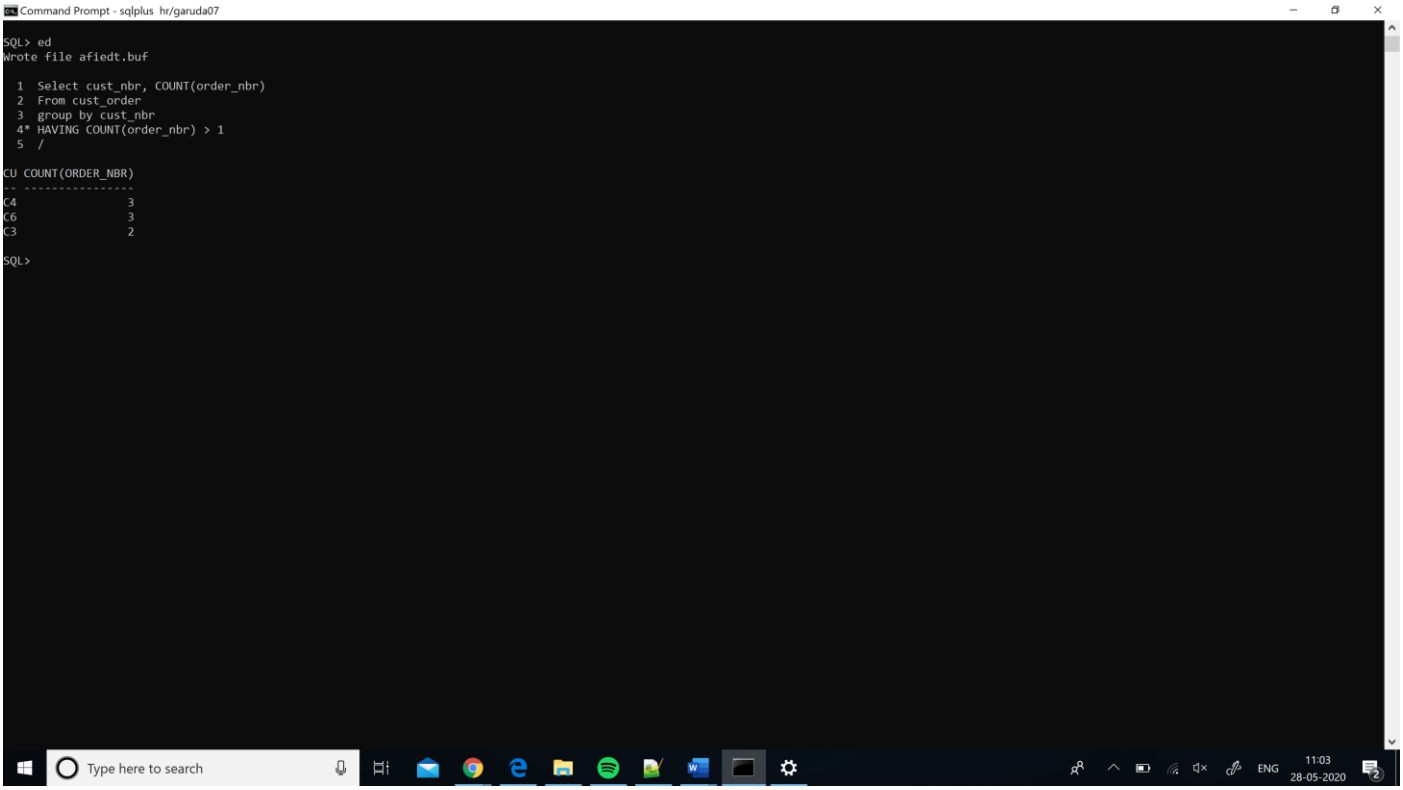
4) Join and subquery: Gives a list of suppliers who supply parts for which the inventory quantity is less than 100 units.

```
Select supplier_id, name
From supplier s
Where exists (SELECT  *
FROM part p
WHERE p.inventory_qty < 100
AND p.supplier_id = s.supplier_id);
```



5) Having Clause using Aggregate Function: Gives count of order numbers for each customer which have total number of orders greater than 1.

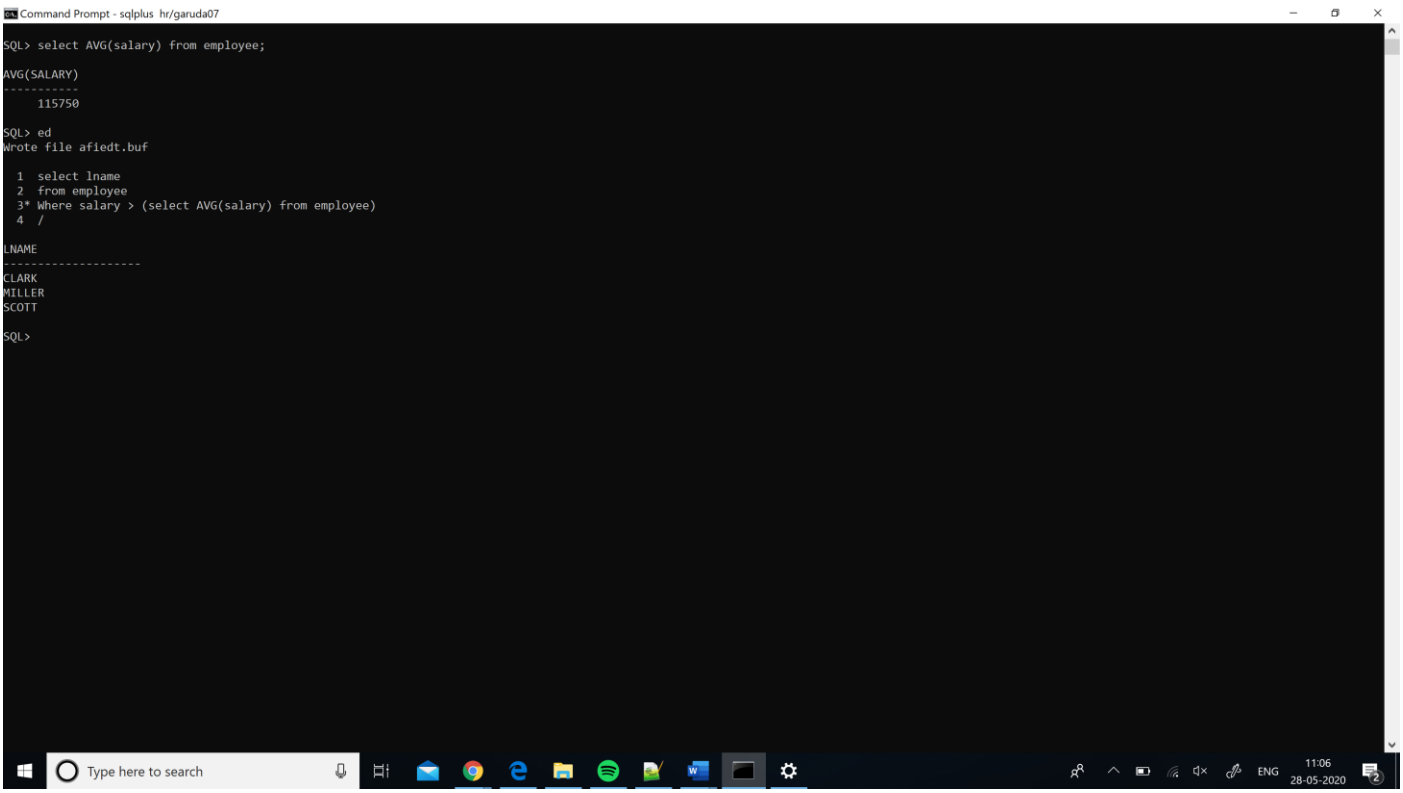
```
Select cust_nbr, COUNT(order_nbr)
From cust_order
group by cust_nbr
HAVING COUNT(order_nbr) > 1;
```



6) Non Correlated subquery : List of all employee last names whose salary is greater than the average salary.

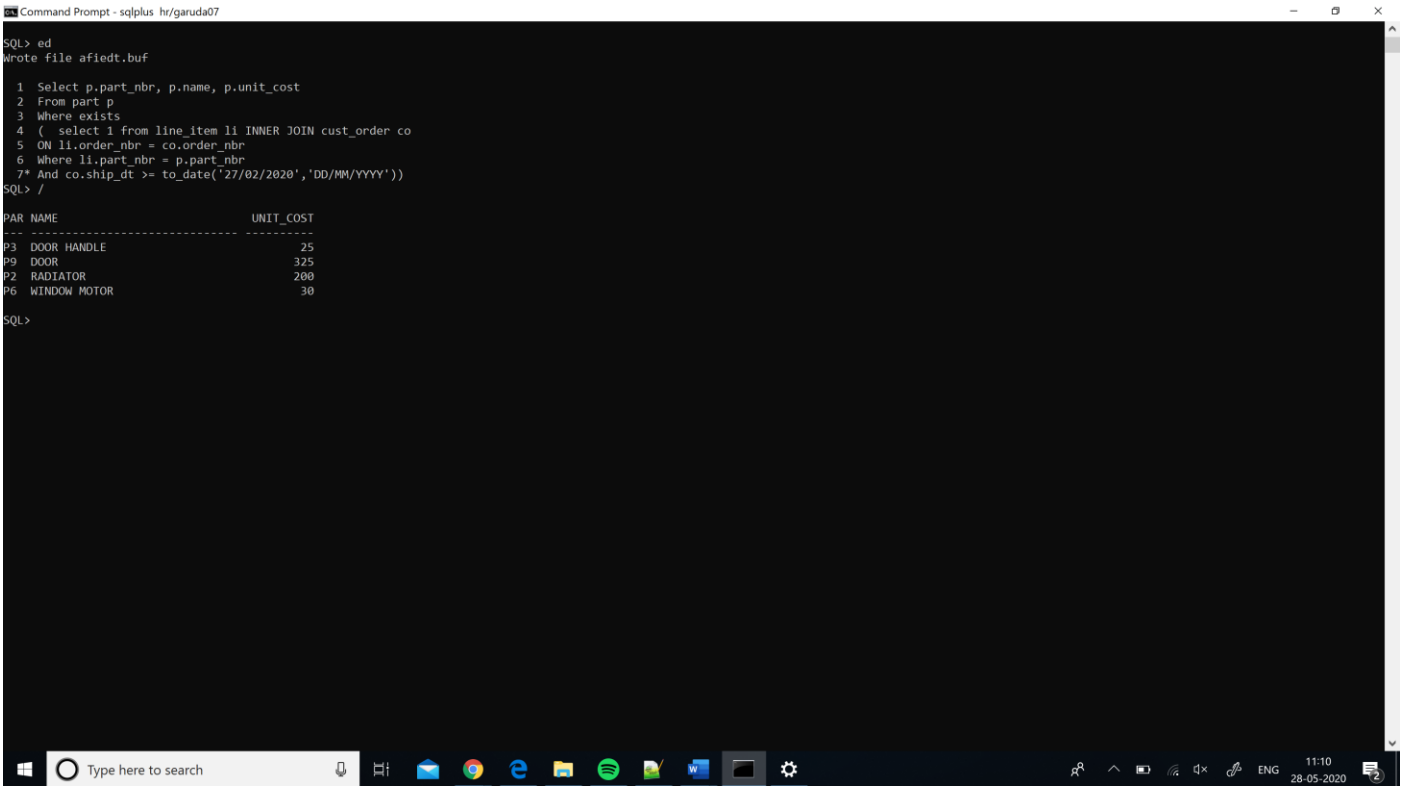
Select AVG(salary) from employee;

Select lname
From employee
Where salary > (select AVG(salary) from employee);



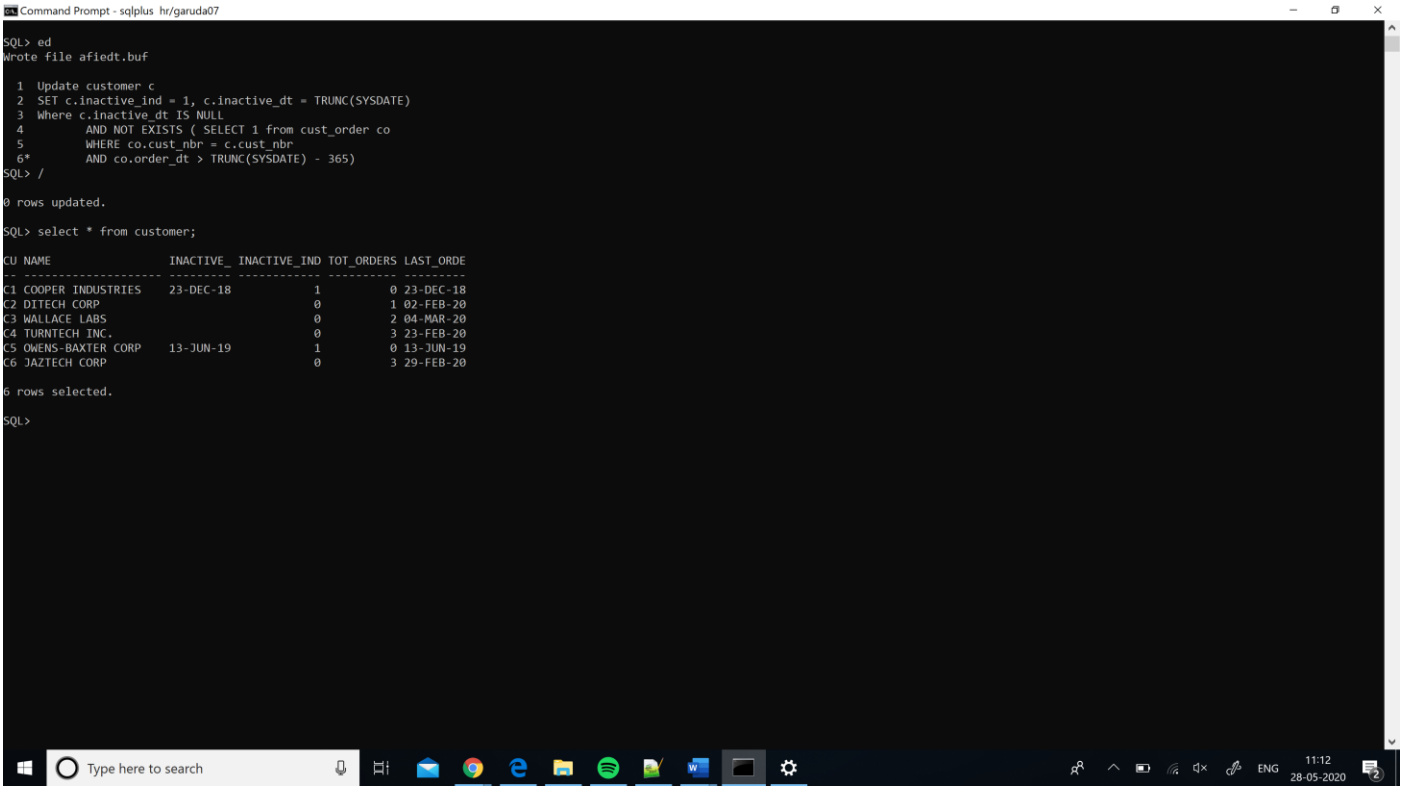
7) Correlated subquery: Gives a list of the names of the parts shipped after 27th Feb 2020.

```
Select p.part_nbr, p.name, p.unit_cost
From part p
Where exists
(  select 1 from ine_item li INNER JOIN cust_order co
ON li.order_nbr = co.order_nbr
Where li.part_nbr = p.part_nbr
And co.ship_dp >= to_date('27/02/2020','DD/MM/YYYY'))
```



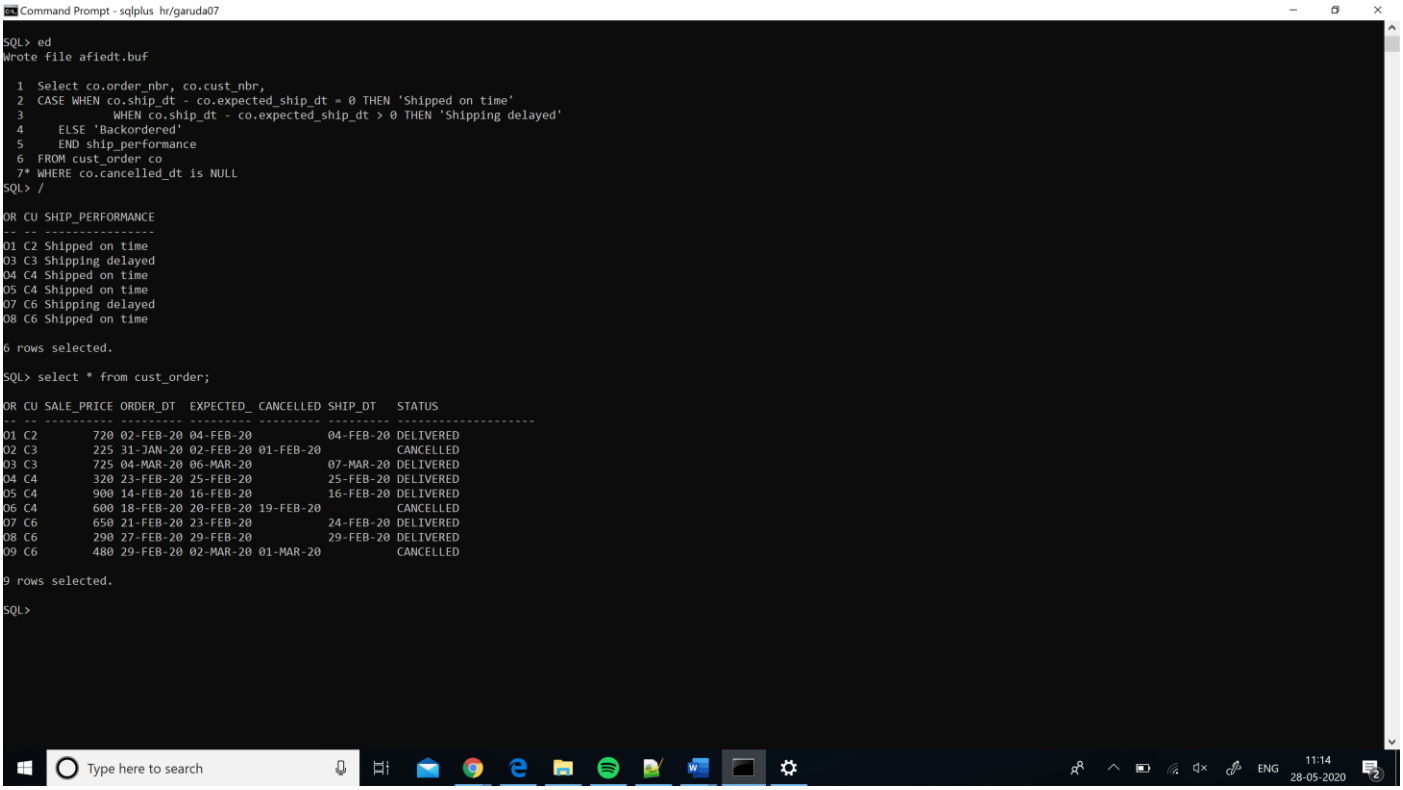
8) Correlated Query 2: Makes all customer records inactive for those customers who haven't placed an order in the past year. Such type of queries are commonly used in maintenance routines.

```
Update customer c
SET c.inactive_ind = 1, c.inactive_dt = TRUNC(SYSDATE)
Where c.inactive_dt IS NULL
    AND NOT EXISTS ( SELECT 1 from cust_order co
    WHERE co.cust_nbr = c.cust_nbr
    AND co.order_dt > TRUNC(SYSDATE) – 365);
```



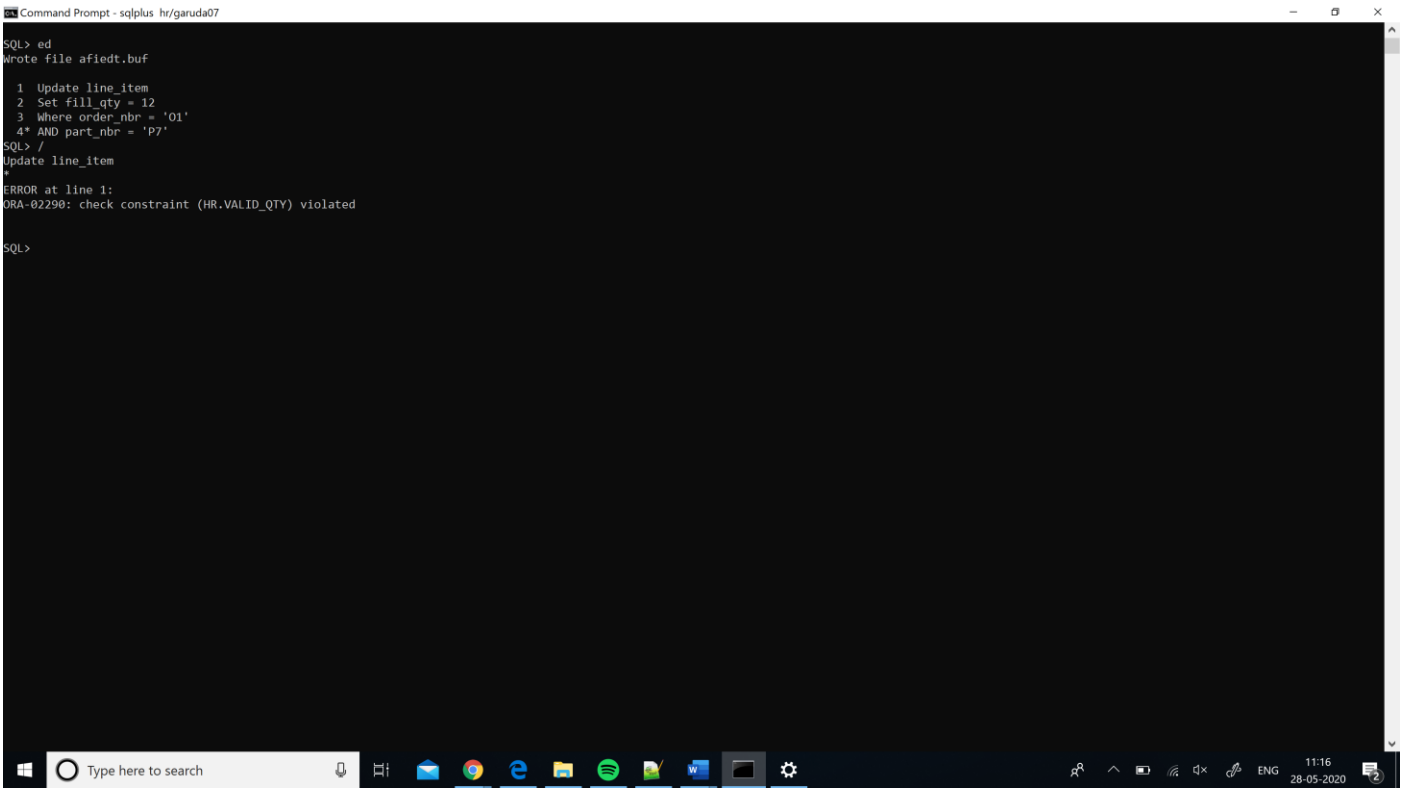
9) Use of case in select statement: Gives a list of the order number, customer number and the message “Shipped on time” or “Shipping delayed” depending on the difference between the expected ship date and the ship date. If the difference is 0 then the message “Shipped on time” will be displayed else if the difference is greater than 0 the message “Shipping delayed” is displayed.

```
Select co.order_nbr, co.cust_nbr,
        CASE WHEN co.ship_dt – co.expected_ship_dt = 0 THEN ‘Shipped on time’
                WHEN co.ship_dt – co.expected_ship_dt > 0 THEN ‘Shipping delayed’
                 ELSE ‘Backordered’
        END ship_performance
FROM cust_order co
WHERE co.cancelled_dt is NULL;
```



10) Check constraint violation example: While defining the tables we introduced a check constraint on the table line_item, specifying that the fill_qty must always be less than or equal to the qty. If we try to update it with a value greater than the qty, an error will appear.

Update line_item
Set fill_qy = 12
Where order_nbr = ‘01’
AND part_nbr = ‘P7’



Test for Lossless Join: Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not be lost from the relation when decomposed. The join would result in the same original relation. This is the lossless join property.

Suppose our Employee relation initially had this structure

Emp_id	Fname	Lname	Dept_id	Location
EMP002	Joe	Miller	D001	L03
EMP003	Tim	Scott	D002	L01
EMP004	Emily	Turner	D003	L02

When we decompose this relation into two relations as we have in our original schema, we get

Employee

Emp_id	Fname	Lname	Dept_id
EMP002	Joe	Miller	D001
EMP003	Tim	Scott	D002
EMP004	Emily	Turner	D003

Department

Dept_id	Location
D001	L03
D002	L01
D003	L02

If we perform a natural join on these two tables, the resulting table will be the original table

Emp_id	Fname	Lname	Dept_id	Location
EMP002	Joe	Miller	D001	L03
EMP003	Tim	Scott	D002	L01
EMP004	Emily	Turner	D003	L02

Therefore, the above relation had lossless decomposition i.e. no loss of information.

Conclusion

Capabilities:

- This system allows us to store master data about employees, customers, suppliers, parts, locations, jobs and departments.
- It also allows us to store transactional data of the customer orders as well as line items.

Limitations:

- As seen in the ER diagram and the relation definition, we have chosen to leave out the sales_emp_id attribute of the customer order table. We cannot track the performance of a particular salesperson in case the performance evaluation needs to be done.
- We cannot track sales by geographical regions.
- We cannot track the amount of time an employee has worked a particular job.

Future Enhancements:

- Allow a part to be supplied by more than one supplier.
- Allow an employee to take on multiple jobs during his employment with the distributor.
- Enhance the data model to allow tracking of sales by geographical regions.