

# 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.

<b>Introduction</b>	<b>3</b>
<b>Data Model</b>	<b>4</b>
<b>FD and Normalization</b>	<b>8</b>
<b>DDL</b>	<b>Error! Bookmark not defined.</b>
<b>Triggers</b>	<b>17</b>
<b>SQL Queries</b>	<b>19</b>
<b>Conclusion</b>	<b>27</b>

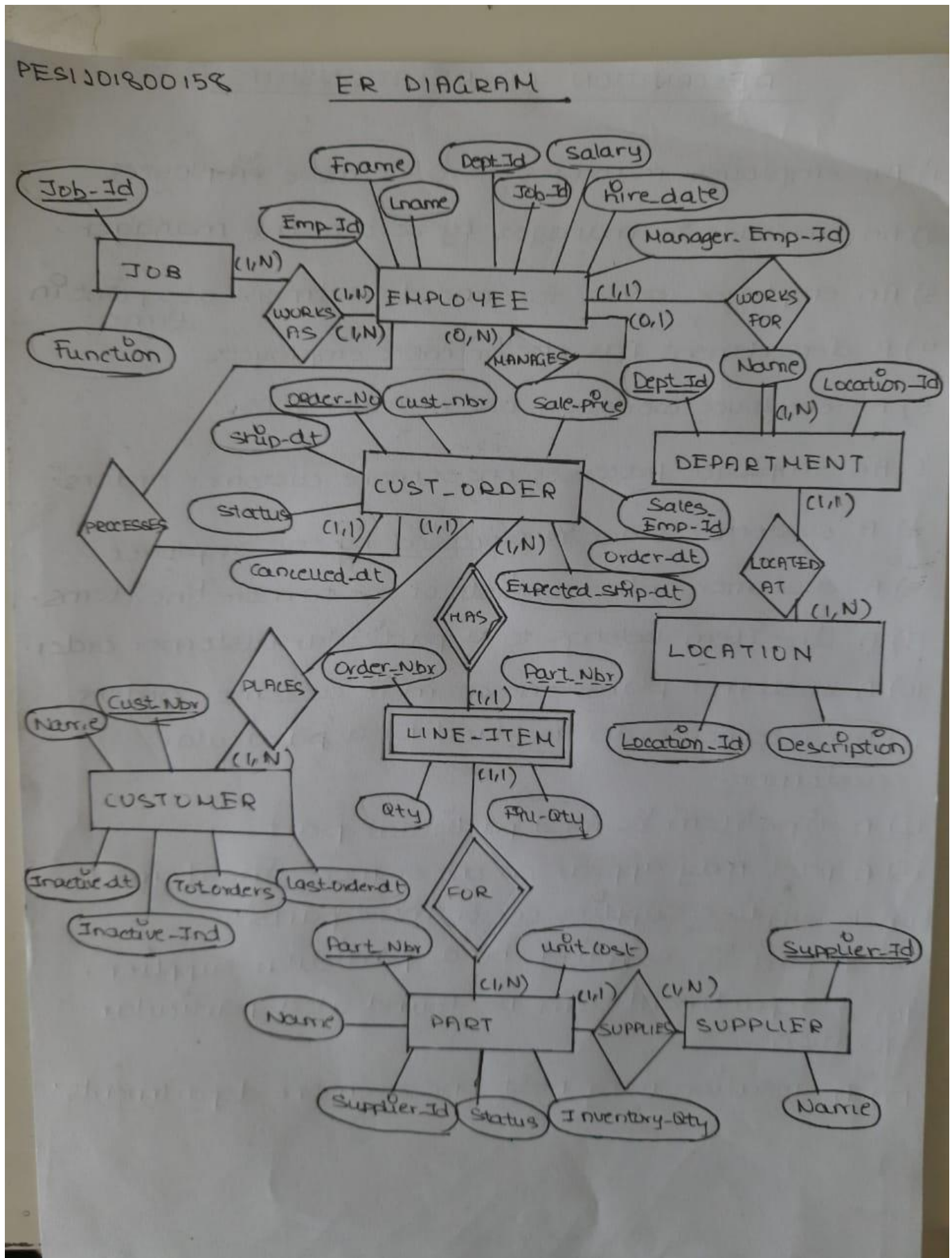
# Introduction

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

<u>job-id</u>	function
---------------	----------

### LOCATION

<u>location-id</u>	description
--------------------	-------------

### DEPARTMENT

<u>dept-id</u>	name	location-id
----------------	------	-------------

### EMPLOYEE

<u>emp-id</u>	fname	lname	dept-id	salary	hire-date	job-id	manager-emp-id
---------------	-------	-------	---------	--------	-----------	--------	----------------

### CUSTOMER

<u>cust-nbr</u>	name	inactive-dt	inactive-Ind	tot-orders	last-order-dt
-----------------	------	-------------	--------------	------------	---------------

### CUST\_ORDER

<u>order-nbr</u>	cust-nbr	sale-price	order-dt	expected-ship-dt
------------------	----------	------------	----------	------------------

cancelled-dt	ship-dt	status
--------------	---------	--------

### SUPPLIER

<u>supplier-id</u>	name
--------------------	------

### PART

<u>part-nbr</u>	name	supplier-id	status	inventory-qty	unit-cost
-----------------	------	-------------	--------	---------------	-----------

### LINE-ITEM

<u>order-nbr</u>	<u>part-nbr</u>	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 <sup>o</sup>is managed by zero or one manager.
- 3) An employee works for one department at a point <sup>o</sup>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 <sup>o</sup>is processed by one employee.
- 8) A customer order consists of one or more line <sup>o</sup>items.
- 9) A line <sup>o</sup>item belongs to a particular customer order.
- 10) A customer places one or more customer orders.
- 11) A customer order <sup>o</sup>is placed by a particular customer.
- 12) A line <sup>o</sup>item <sup>o</sup>is for a particular part.
- 13) A part may appear on one <sup>o</sup>or more line <sup>o</sup>items.
- 14) A supplier <sup>o</sup>supplies one or more parts.
- 15) A part <sup>o</sup>is supplied by a particular supplier.
- 16) A department can be found at a particular <sup>o</sup>location.
- 17) A location may host one or more departments.



# FD and Normalization

## FUNCTIONAL DEPENDENCIES

JOB

<u>job-id</u>	function
---------------	----------

LOCATION

<u>location-id</u>	description
--------------------	-------------

DEPARTMENT

<u>dept-id</u>	name	location-id
----------------	------	-------------

EMPLOYEE

<u>emp-id</u>	frame	lname	dept-id	salary	hire-date	job-id	manager-emp-id
---------------	-------	-------	---------	--------	-----------	--------	----------------

CUSTOMER

<u>cust-nbr</u>	name	inactive-dt	inactive-9nd	tot-orders	last-order-dt
-----------------	------	-------------	--------------	------------	---------------

CUST-ORDER

<u>order-nbr</u>	cust-nbr	sale-price	order-dt	expected-ship-dt	cancelled-dt	ship-dt	status
------------------	----------	------------	----------	------------------	--------------	---------	--------

SUPPLIER

<u>supplier-id</u>	name
--------------------	------

PART

<u>part-nbr</u>	name	supplier-id	status	inventory-qty	unit-cost
-----------------	------	-------------	--------	---------------	-----------

LINE-ITEM

<u>order-nbr</u>	<u>part-nbr</u>	qty	fill-qty
------------------	-----------------	-----	----------



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.

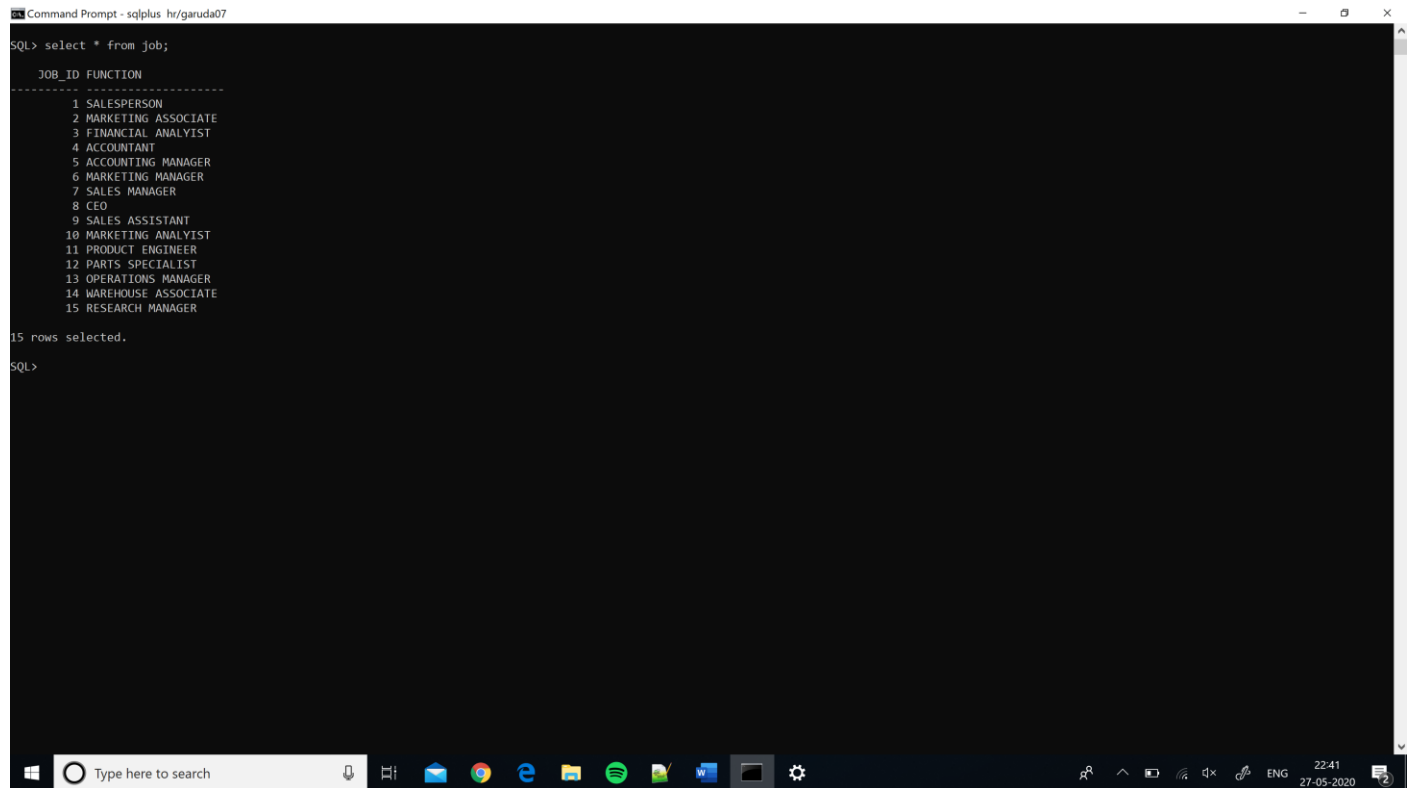
2nf is violated if the part name is added to the line\_item relation. This is because a non-key attribute i.e., the part name is only dependent on a subset of the primary key (part name is only dependent on the part\_nbr and not order\_nbr). To be in 2nf the non key attributes must depend on the entire 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  
                )  
;
```



```
Command Prompt - sqlplus hr/garuda07  
SQL> select * from job;  
  
JOB_ID FUNCTION  
-----  
1 SALESPERSON  
2 MARKETING ASSOCIATE  
3 FINANCIAL ANALYST  
4 ACCOUNTANT  
5 ACCOUNTING MANAGER  
6 MARKETING MANAGER  
7 SALES MANAGER  
8 CEO  
9 SALES ASSISTANT  
10 MARKETING ANALYST  
11 PRODUCT ENGINEER  
12 PARTS SPECIALIST  
13 OPERATIONS MANAGER  
14 WAREHOUSE ASSOCIATE  
15 RESEARCH MANAGER  
  
15 rows selected.  
SQL>
```

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

```
Command Prompt - sqlplus hr/garuda07

SQL> select * from location;

LOC DESCRIPTION
-----
L01 DALLAS
L02 BOSTON
L03 CHICAGO

SQL>
```

```
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)
                        )
;
```

```
Command Prompt - sqlplus hr/garuda07

SQL> select * from department;

DEPT NAME          LOC
-----
D001 ACCOUNTING    L03
D002 RESEARCH      L01
D003 SALES         L02
D004 OPERATIONS    L02

SQL>
```

```
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)
)
;

```

```

Command Prompt - sqlplus hr/garuda07

SQL> select * from employee;

EMP_ID FNAME          LNAME          DEPT     SALARY HIRE_DATE      JOB_ID MANAGE
-----
EMP001 DANA            CLARK           D001     150000 16-JUN-08      8
EMP002 JOE            MILLER          D001     100000 17-JUL-11      7 EMP001
EMP003 TIM            SCOTT           D002     78000  06-JAN-12     13 EMP001
EMP004 EMILY          TURNER          D003     79000  08-MAR-11      6 EMP001
EMP005 JAMES          MARTIN          D002     76000  24-AUG-14     15 EMP001
EMP006 EVA            TYLER           D001     77000  26-MAY-13      5 EMP001
EMP007 TOM            FORD            D003     50000  23-DEC-12      1 EMP002
EMP008 JACOB          SMITH           D003     51000  29-JAN-12      1 EMP002
EMP009 SARAH          WILLIAMS        D003     50000  18-MAR-13      1 EMP002
EMP010 BLAKE          LIVERLY         D003     60000  22-NOV-13      2 EMP004
EMP011 SAM            JONES           D001     63000  13-MAY-14      3 EMP006
EMP012 MEGAN          ADAMS           D001     63000  05-FEB-14      3 EMP006
EMP013 JOE            JONAS           D001     64000  23-MAR-15      4 EMP006
EMP014 OLIVER         TRUDEAU         D003     45000  12-JUN-14      9 EMP002
EMP015 JOHN          WARD            D003     62000  27-AUG-13     10 EMP002
EMP016 SOPHIA         ROSE            D003     62000  31-DEC-13     10 EMP002
EMP017 KATE          BLANE           D002     57000  01-MAR-13     11 EMP005
EMP018 JEREMY         PHILIP          D002     57000  09-SEP-14     11 EMP005
EMP019 JUSTIN        LAKER           D002     59000  14-OCT-14     12 EMP005
EMP020 MIA            LOPEZ           D004     40000  02-JUL-12     14 EMP003

20 rows selected.

SQL>

```

```

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
)
;

```

```
Command Prompt - sqlplus hr/garuda07
SQL> select * from customer;

CU NAME          INACTIVE_ INACTIVE_IND TOT_ORDERS LAST_ORDE
-----
C1 COOPER INDUSTRIES 23-DEC-18      1          0 23-DEC-18
C2 DIITECH CORP      0              0          1 02-FEB-20
C3 WALLACE LABS      0              2 04-MAR-20
C4 TURMTECH INC.     0              3 23-FEB-20
C5 OWENS-BAXTER CORP 13-JUN-19      1          0 13-JUN-19
C6 JAZTECH CORP      0              3 29-FEB-20

6 rows selected.

SQL>
```

```
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)
                        )
;
```

```
Command Prompt - sqlplus hr/garuda07

SQL> select *from cust_order;

OR CU SALE_PRICE ORDER_DT EXPECTED_ CANCELLED SHIP_DT STATUS
-----
01 C2 720 02-FEB-20 04-FEB-20 04-FEB-20 DELIVERED
02 C3 225 31-JAN-20 02-FEB-20 01-FEB-20 CANCELLED
03 C3 725 04-MAR-20 06-MAR-20 07-MAR-20 DELIVERED
04 C4 320 23-FEB-20 25-FEB-20 25-FEB-20 DELIVERED
05 C4 900 14-FEB-20 16-FEB-20 16-FEB-20 DELIVERED
06 C4 600 18-FEB-20 20-FEB-20 19-FEB-20 CANCELLED
07 C6 650 21-FEB-20 23-FEB-20 24-FEB-20 DELIVERED
08 C6 290 27-FEB-20 29-FEB-20 29-FEB-20 DELIVERED
09 C6 480 29-FEB-20 02-MAR-20 01-MAR-20 CANCELLED

9 rows selected.

SQL>
```

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

```
Command Prompt - sqlplus hr/garuda07

SQL> select * from supplier;

SUPPLIER_ID NAME
-----
1 ACME INDUSTRIES
2 TILTON ENTERPRISES
3 GOODYEAR TIRE COMPANY
4 BORNWAGON MANUFACTURERS
5 TENNECO
6 LEAR CORP.

6 rows selected.

SQL>
```

```
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)
)

```

```

Command Prompt - sqlplus hr/garuda07

SQL> select * from part;

PAR NAME          SUPPLIER_ID STATUS          INVENTORY_QTY  UNIT_COST
-----
P1 BONNET          1 OBSOLETE          0             950
P2 RADIATOR        6 ACTIVE            140            200
P3 DOOR HANDLE     2 ACTIVE            160             25
P4 SUN ROOF        4 FROZEN             80             150
P5 GLASS           1 ACTIVE            100             10
P6 WINDOW MOTOR    3 ACTIVE            120             30
P7 RIM             4 ACTIVE            180            100
P8 TIRE            3 ACTIVE            160             50
P9 DOOR            2 ACTIVE             60            325
P10 SPOILER        6 OBSOLETE           0             300

10 rows selected.

SQL>

```

```

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)
)
;

```



```
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>
```

# 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:

```
Command Prompt - sqlplus hr/garuda07

SQL> @C:\Users\vibha\Sem4\DBMS\MINI_PROJECT\procedureForTrigger.sql
Procedure created.

SQL> @C:\Users\vibha\Sem4\DBMS\MINI_PROJECT\trigger.sql
Trigger created.

SQL> update employee set salary = 150000 where emp_id = 'EMP003';
1 row updated.

SQL> select * from mgr_alert;

MGR_ID ALERT                                EMP_ID  MESSAGE2
-----
EMP001 salary update for employee_id        EMP003   150000

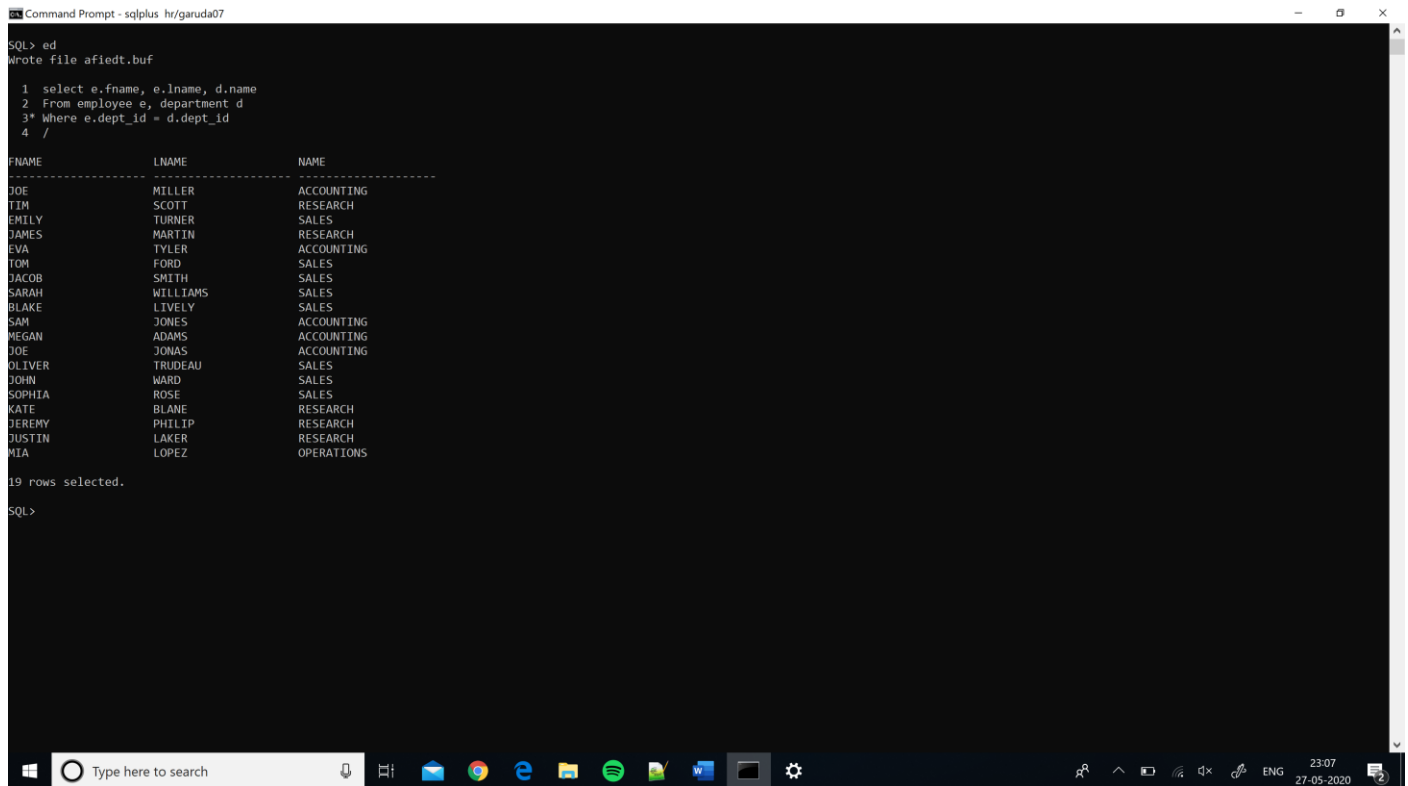
SQL>
```

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
```



```
SQL> ed
Wrote file afiledt.buf

1 select e.fname, e.lname, d.name
2 From employee e, department d
3* Where e.dept_id = d.dept_id
4 /

FNAME          LNAME          NAME
-----
DOE            MILLER         ACCOUNTING
TIM            SCOTT          RESEARCH
EMILY          TURNER         SALES
JAMES          MARTIN         RESEARCH
EVA            TYLER          ACCOUNTING
TOM            FORD           SALES
JACOB          SMITH          SALES
SARAH          WILLIAMS       SALES
BLAKE          LIVERLY        SALES
SAM            JONES          ACCOUNTING
MEGAN          ADAMS          ACCOUNTING
DOE            JONAS          ACCOUNTING
OLIVER         TRUDEAU        SALES
JOHN           WARD           SALES
SOPHIA         ROSE           SALES
KATE           BLANE          RESEARCH
JEREMY         PHILIP         RESEARCH
DUSTIN         LAKER          RESEARCH
MIA            LOPEZ          OPERATIONS

19 rows selected.

SQL>
```

- 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
```

```
Command Prompt - sqlplus hr/garuda07

SQL> ed
Wrote file afiedt.buf

 1 Select s.supplier_id, s.name, p.part_nbr, p.name
 2 From supplier s LEFT OUTER JOIN part p
 3* ON s.supplier_id = p.supplier_id
 4 /

SUPPLIER_ID NAME                PAR NAME
-----
1 ACME INDUSTRIES              P1 BONNET
6 LEAR CORP.                  P2 RADIATOR
2 TILTON ENTERPRISES          P3 DOOR HANDLE
4 BORNWAGON MANUFACTURERS     P4 SUN ROOF
1 ACME INDUSTRIES              P5 GLASS
3 GOODYEAR TIRE COMPANY       P6 WINDOW MOTOR
4 BORNWAGON MANUFACTURERS     P7 RIM
3 GOODYEAR TIRE COMPANY       P8 TIRE
2 TILTON ENTERPRISES          P9 DOOR
6 LEAR CORP.                  P10 SPOILER
5 TENNECO

11 rows selected.

SQL>
```

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;
```

```
Command Prompt - sqlplus hr/garuda07

SQL> ed
Wrote file afiedt.buf

 1 Select e.fname || ' ' || e.lname subordinate, m.fname || ' ' || m.lname manager
 2 From employee e JOIN employee m
 3* ON e.manager_emp_id = m.emp_id
SQL> /

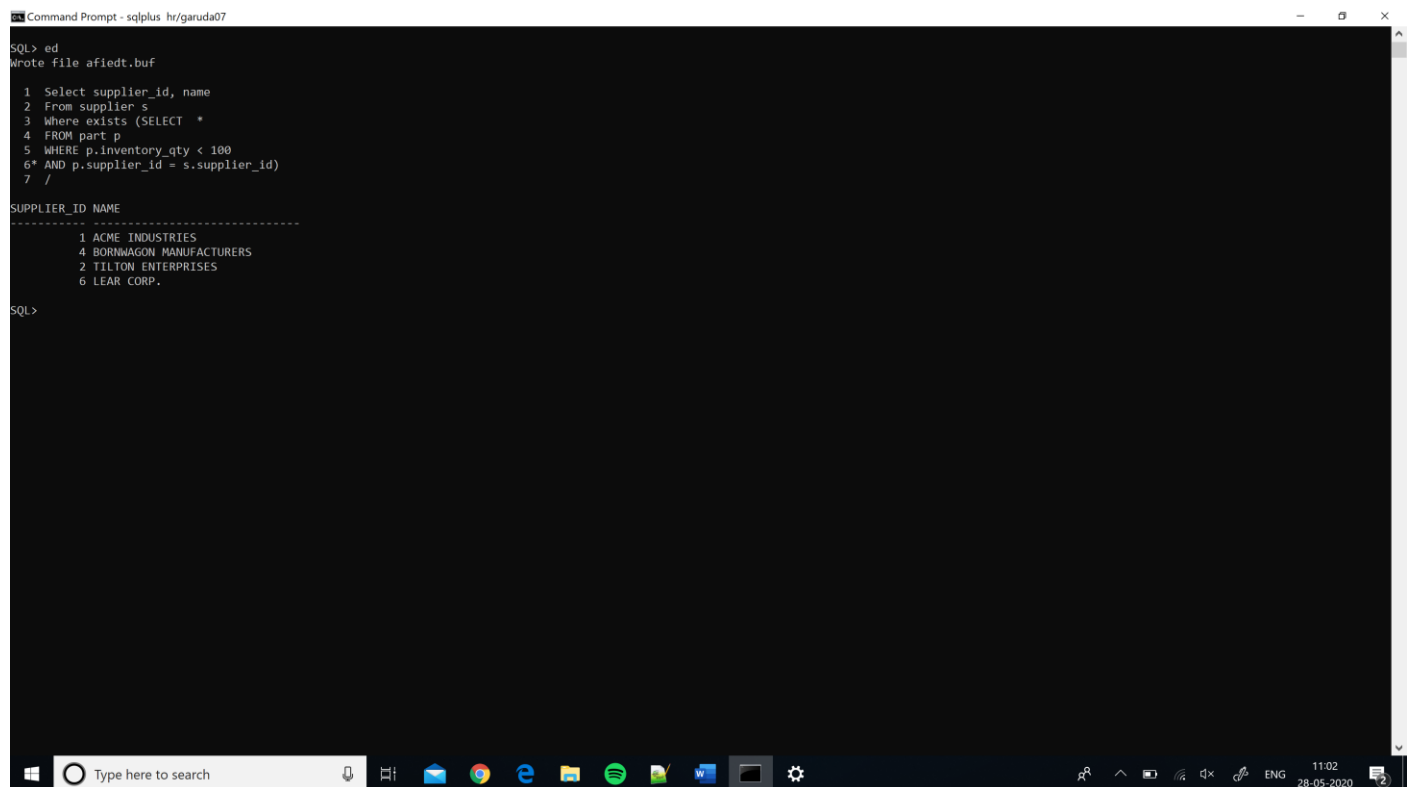
SUBORDINATE                MANAGER
-----
EVA TYLER                  DANA CLARK
JAMES MARTIN               DANA CLARK
EMILY TURNER               DANA CLARK
TIM SCOTT                  DANA CLARK
JOE MILLER                 DANA CLARK
SOPHIA ROSE                JOE MILLER
JOHN WARD                  JOE MILLER
OLIVER TRUDEAU             JOE MILLER
SARAH WILLIAMS             JOE MILLER
JACOB SMITH                JOE MILLER
TOM FORD                   JOE MILLER
MIA LOPEZ                  TIM SCOTT
BLAKE LIVELY               EMILY TURNER
DUSTIN LAKER               JAMES MARTIN
JEREMY PHILIP              JAMES MARTIN
KATE BLANE                 JAMES MARTIN
JOE JONAS                  EVA TYLER
MEGAN ADAMS                EVA TYLER
SAM JONES                  EVA TYLER

19 rows selected.

SQL>
```

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);
```



```
Command Prompt - sqlplus hr/garuda07

SQL> ed
Wrote file afiedt.buf

 1 Select supplier_id, name
 2 From supplier s
 3 Where exists (SELECT *
 4 FROM part p
 5 WHERE p.inventory_qty < 100
 6* AND p.supplier_id = s.supplier_id)
 7 /

SUPPLIER_ID NAME
-----
 1 ACME INDUSTRIES
 4 BORNWAGON MANUFACTURERS
 2 TILTON ENTERPRISES
 6 LEAR CORP.

SQL>
```

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;
```

```
Command Prompt - sqlplus hr/garuda07

SQL> ed
Wrote file afiedt.buf

 1 Select cust_nbr, COUNT(order_nbr)
 2 From cust_order
 3 group by cust_nbr
 4* HAVING COUNT(order_nbr) > 1
 5 /

CU COUNT(ORDER_NBR)
--
C4          3
C6          3
C3          2

SQL>
```

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);

```
Command Prompt - sqlplus hr/garuda07

SQL> select AVG(salary) from employee;

AVG(SALARY)
-----
      115750

SQL> ed
Wrote file afiedt.buf

 1 select lname
 2 from employee
 3* where salary > (select AVG(salary) from employee)
 4 /

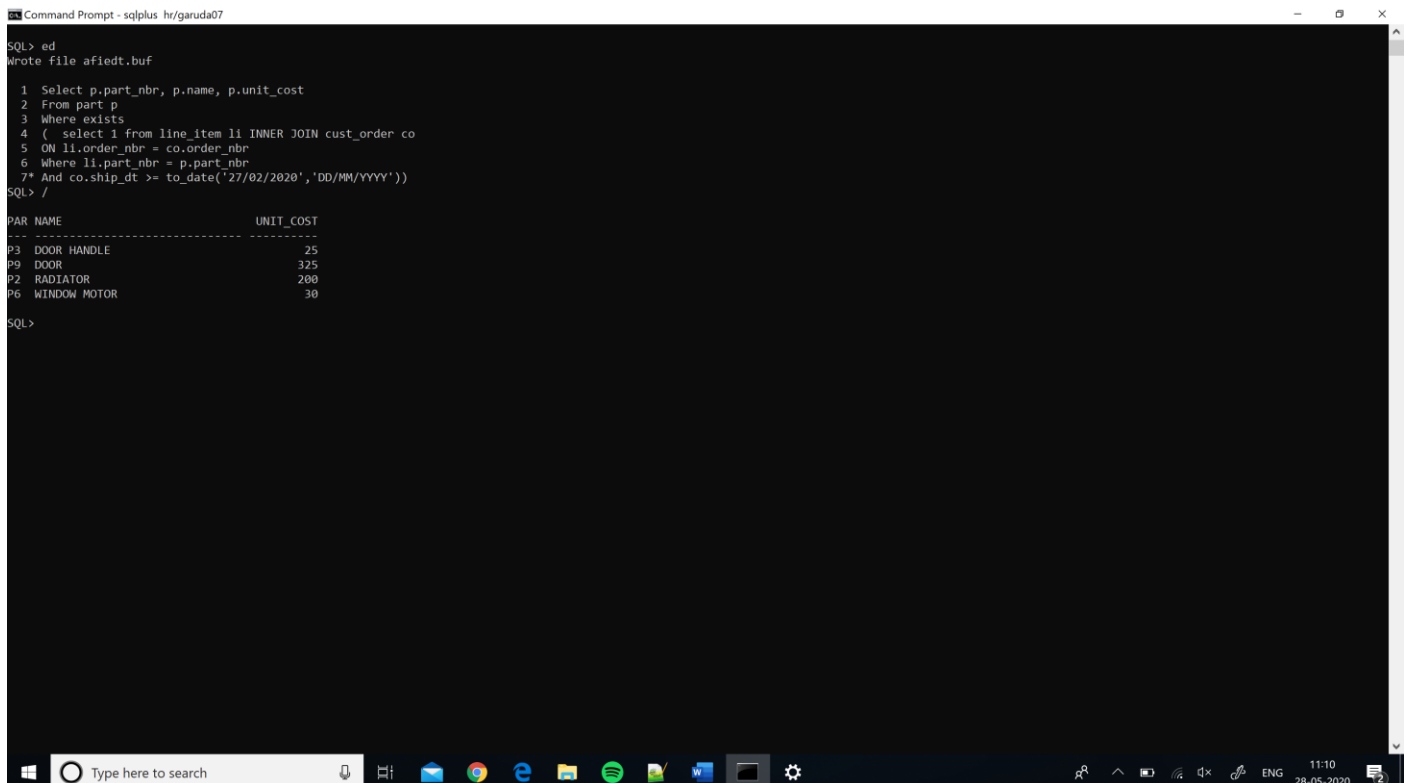
LNAME
-----
CLARK
MILLER
SCOTT

SQL>
```



7) Correlated subquery: Gives a list of the names of the parts shipped after 27<sup>th</sup> 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'))
```



```
Command Prompt - sqlplus hr/garuda07
SQL> ed
Write file afiedt.buf

1 Select p.part_nbr, p.name, p.unit_cost
2 From part p
3 Where exists
4 ( select 1 from line_item li INNER JOIN cust_order co
5 ON li.order_nbr = co.order_nbr
6 Where li.part_nbr = p.part_nbr
7* And co.ship_dt >= to_date('27/02/2020','DD/MM/YYYY'))
SQL> /

PAR NAME                                UNIT_COST
-----
P3 DOOR HANDLE                           25
P9 DOOR                                  325
P2 RADIATOR                              200
P6 WINDOW MOTOR                          30
SQL>
```

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);
```

```
Command Prompt - sqlplus hr/garuda07
SQL> ed
Wrote file afiedt.buf

 1 Update customer c
 2 SET c.inactive_ind = 1, c.inactive_dt = TRUNC(SYSDATE)
 3 Where c.inactive_dt IS NULL
 4 AND NOT EXISTS ( SELECT 1 from cust_order co
 5 WHERE co.cust_nbr = c.cust_nbr
 6* AND co.order_dt > TRUNC(SYSDATE) - 365)
SQL> /

0 rows updated.

SQL> select * from customer;

CU NAME          INACTIVE_ INACTIVE_IND TOT_ORDERS LAST_ORDE
-----
C1 COOPER INDUSTRIES 23-DEC-18          1          0 23-DEC-18
C2 DITECH CORP          0          0          1 02-FEB-20
C3 WALLACE LABS          0          0          2 04-MAR-20
C4 TURNTECH INC.          0          0          3 23-FEB-20
C5 OWENS-BAXTER CORP 13-JUN-19          1          0 13-JUN-19
C6 JAZTECH CORP          0          0          3 29-FEB-20

6 rows selected.

SQL>
```

- 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;
```

```
Command Prompt - sqlplus hr/garuda07

SQL> ed
Wrote file afiedt.buf

 1 Select co.order_nbr, co.cust_nbr,
 2 CASE WHEN co.ship_dt - co.expected_ship_dt = 0 THEN 'Shipped on time'
 3       WHEN co.ship_dt - co.expected_ship_dt > 0 THEN 'Shipping delayed'
 4       ELSE 'Backordered'
 5 END ship_performance
 6 FROM cust_order co
 7* WHERE co.cancelled_dt is NULL
SQL> /

OR CU SHIP_PERFORMANCE
-----
01 C2 Shipped on time
03 C3 Shipping delayed
04 C4 Shipped on time
05 C4 Shipped on time
07 C6 Shipping delayed
08 C6 Shipped on time

6 rows selected.

SQL> select * from cust_order;

OR CU SALE_PRICE ORDER_DT EXPECTED_ CANCELLED SHIP_DT STATUS
-----
01 C2 720 02-FEB-20 04-FEB-20 01-FEB-20 04-FEB-20 DELIVERED
02 C3 225 31-JAN-20 02-FEB-20 01-FEB-20 CANCELLED
03 C3 725 04-MAR-20 06-MAR-20 07-MAR-20 DELIVERED
04 C4 320 23-FEB-20 25-FEB-20 25-FEB-20 DELIVERED
05 C4 900 14-FEB-20 16-FEB-20 16-FEB-20 DELIVERED
06 C4 600 18-FEB-20 20-FEB-20 19-FEB-20 CANCELLED
07 C6 650 21-FEB-20 23-FEB-20 24-FEB-20 DELIVERED
08 C6 290 27-FEB-20 29-FEB-20 29-FEB-20 DELIVERED
09 C6 480 29-FEB-20 02-MAR-20 01-MAR-20 CANCELLED

9 rows selected.

SQL>
```

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\_qty = 12  
Where order\_nbr = '01'  
AND part\_nbr = 'P7'

```
Command Prompt - sqlplus hr/garuda07

SQL> ed
Wrote file afiedt.buf

 1 Update line_item
 2 Set fill_qty = 12
 3 Where order_nbr = '01'
 4* AND part_nbr = 'P7'
SQL> /
Update line_item
*
ERROR at line 1:
ORA-02290: check constraint (HR.VALID_QTY) violated

SQL>
```

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.