HEAVEN'S

DATABASE DESIGN PROPOSAL

Payal Sharma

25 April 2014

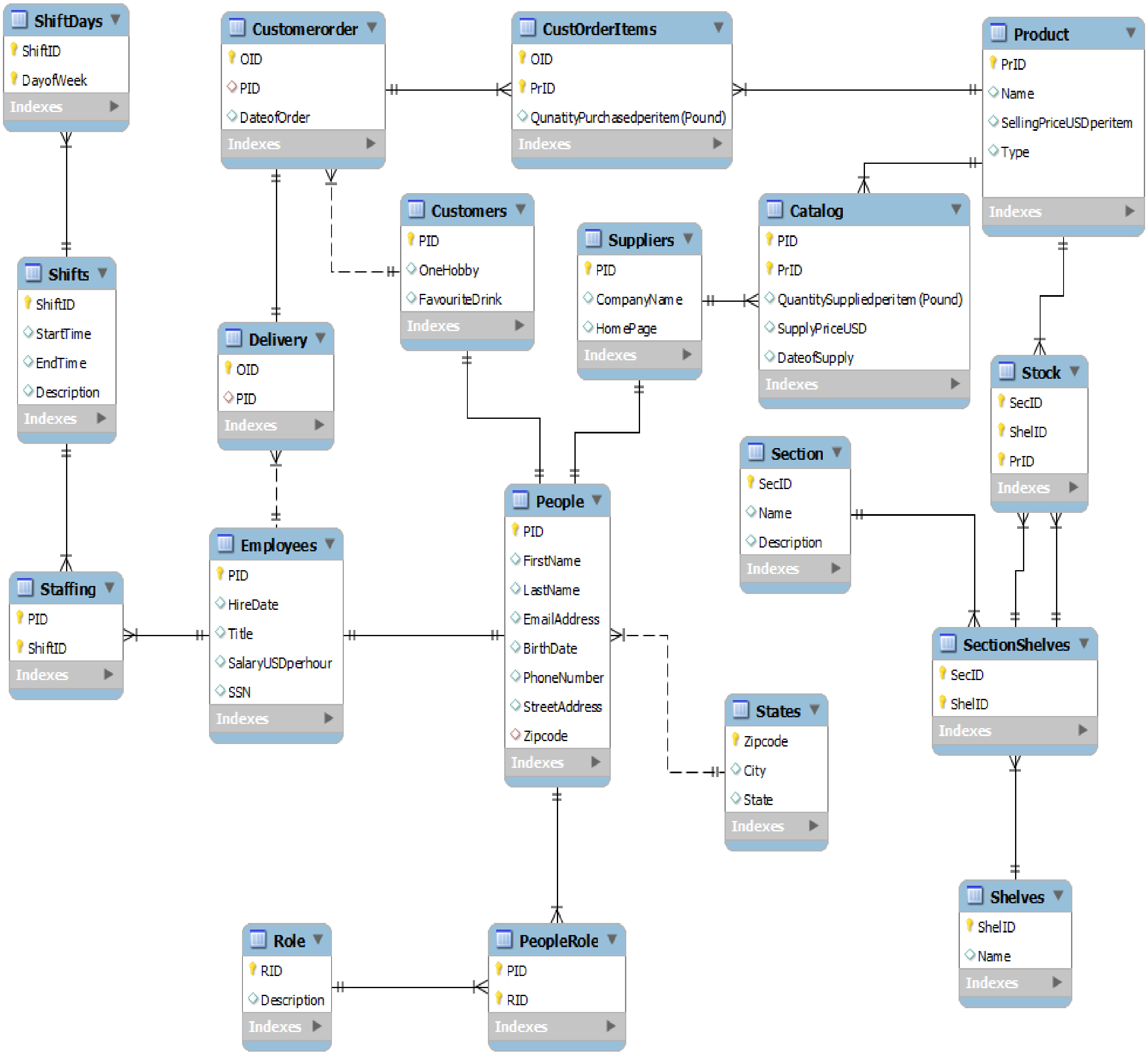
TABLE OF CONTENTS

EXECUTIVE SUMMARY

A grocery store is a retail store that sells food and other non-food items. Large grocery stores that stock products other than food, such as clothing or household items are called supermarkets. Small grocery stores that mainly sell fruits and vegetables are known as produce markets in America. There are over 100 grocery stores in the USA and over 75 large well-known grocery retailers and 50 small chain grocery stores located all through the United States. On average 18000 people go to grocery store every day.

This database proposal demonstrates a grocery store. In order to keep track of the information about the employees salary, product stock, product sales, daily profit and customer information, this database has been created. Its overview has been presented, followed by the details of how each of the database tables will be created. Each table will be followed with the functional dependencies and a table of sample data. Few database roles have been created and their usage has been explained. Furthermore, few triggers, stored procedures and security have been designed to ensure the data integrity. Sample reports have been created to show the information useful for people associated with store. Implementation details, known problems and future enhancements have also been included at the end of the document. This design was targeted for and tested on PostgreSQL 9.3

ENTITY RELATIONSHIP DIAGRAM

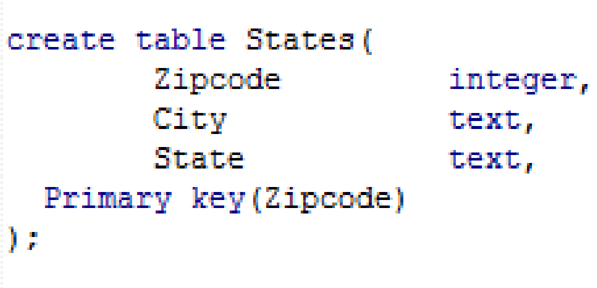


CREATE TABLE STATEMENTS

STATES TABLE:

This table will store valid zip codes with their associated city and state. This table will refer to People table.

CREATE STATEMENT:

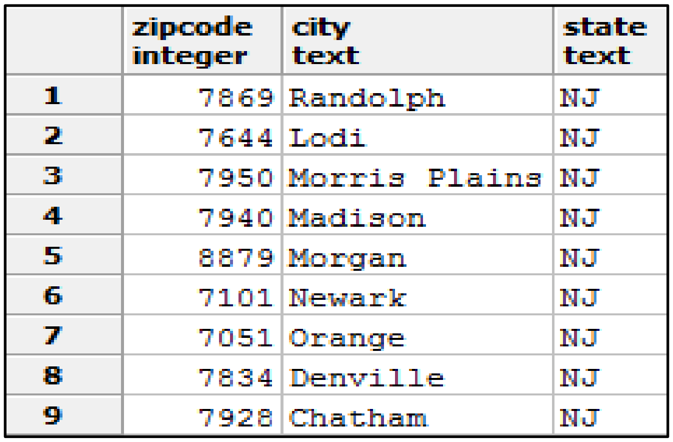


```
create table States(
        Zipcode          integer,
        City             text,
        State            text,
  Primary key(Zipcode)
);
```

FUNCTIONAL DEPENDENCIES:

Zipcode → City, State

SAMPLE DATA:

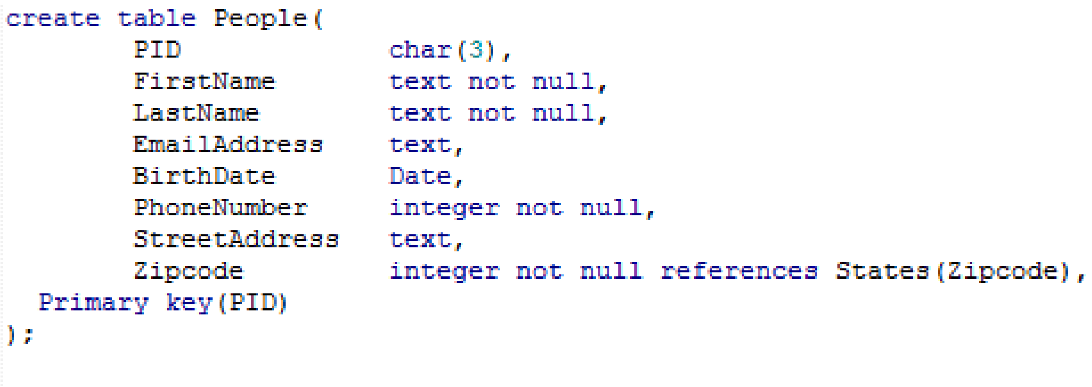


	zipcode integer	city text	state text
1	7869	Randolph	NJ
2	7644	Lodi	NJ
3	7950	Morris Plains	NJ
4	7940	Madison	NJ
5	8879	Morgan	NJ
6	7101	Newark	NJ
7	7051	Orange	NJ
8	7834	Denville	NJ
9	7928	Chatham	NJ

PEOPLE TABLE:

Since an employee can be a customer and a customer can be a supplier so this table will contain basic information about people.

CREATE STATEMENT:

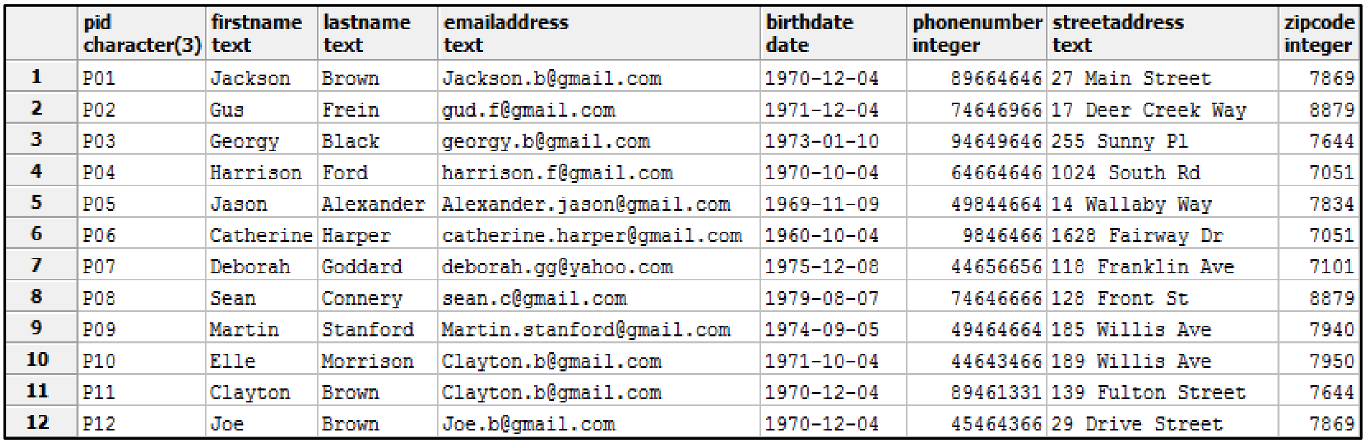


```
create table People(
        PID             char(3),
        FirstName       text not null,
        LastName        text not null,
        EmailAddress    text,
        BirthDate       Date,
        PhoneNumber     integer not null,
        StreetAddress   text,
        Zipcode         integer not null references States(Zipcode),
  Primary key(PID)
);
```

FUNCTIONAL DEPENDENCIES:

PID → FirstName, LastName, EmailAddress, BirthDate, PhoneNumber, StreetAddress, Zipcode

SAMPLE DATA:

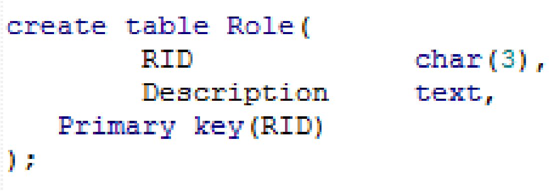


	pid character(3)	firstname text	lastname text	emailaddress text	birthdate date	phonenumber integer	streetaddress text	zipcode integer
1	P01	Jackson	Brown	Jackson.b@gmail.com	1970-12-04	89664646	27 Main Street	7869
2	P02	Gus	Frein	gud.f@gmail.com	1971-12-04	74646966	17 Deer Creek Way	8879
3	P03	Georgy	Black	georgy.b@gmail.com	1973-01-10	94649646	255 Sunny Pl	7644
4	P04	Harrison	Ford	harrison.f@gmail.com	1970-10-04	64664646	1024 South Rd	7051
5	P05	Jason	Alexander	Alexander.jason@gmail.com	1969-11-09	49844664	14 Wallaby Way	7834
6	P06	Catherine	Harper	catherine.harper@gmail.com	1960-10-04	9846466	1628 Fairway Dr	7051
7	P07	Deborah	Goddard	deborah.gg@yahoo.com	1975-12-08	44656656	118 Franklin Ave	7101
8	P08	Sean	Connery	sean.c@gmail.com	1979-08-07	74646666	128 Front St	8879
9	P09	Martin	Stanford	Martin.stanford@gmail.com	1974-09-05	49464664	185 Willis Ave	7940
10	P10	Elle	Morrison	Clayton.b@gmail.com	1971-10-04	44643466	189 Willis Ave	7950
11	P11	Clayton	Brown	Clayton.b@gmail.com	1970-12-04	89461331	139 Fulton Street	7644
12	P12	Joe	Brown	Joe.b@gmail.com	1970-12-04	45464366	29 Drive Street	7869

ROLE TABLE:

This table will contain different roles a person will perform in this database.

CREATE STATEMENT:

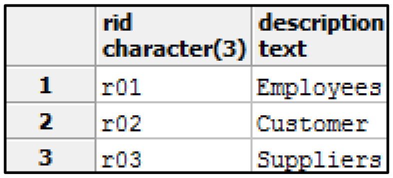


```
create table Role(
        RID             char(3),
        Description     text,
   Primary key(RID)
);
```

FUNCTIONAL DEPENDENCIES:

RID → Description

SAMPLE DATA:

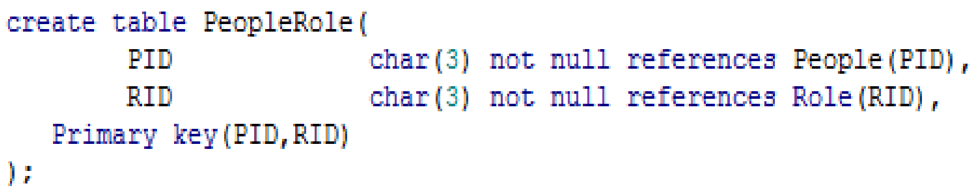


	rid character(3)	description text
1	r01	Employees
2	r02	Customer
3	r03	Suppliers

PEOPLEROLE TABLE:

This table will show which person has which role.

CREATE STATEMENT:

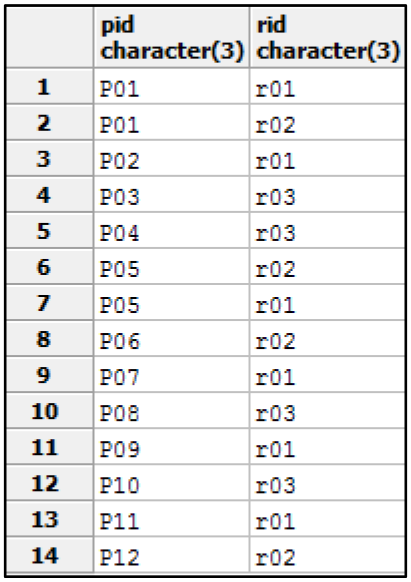


```
create table PeopleRole(
        PID             char(3) not null references People(PID),
        RID             char(3) not null references Role(RID),
   Primary key(PID,RID)
);
```

FUNCTIONAL DEPENDENCIES:

(PID,RID) →

SAMPLE DATA:

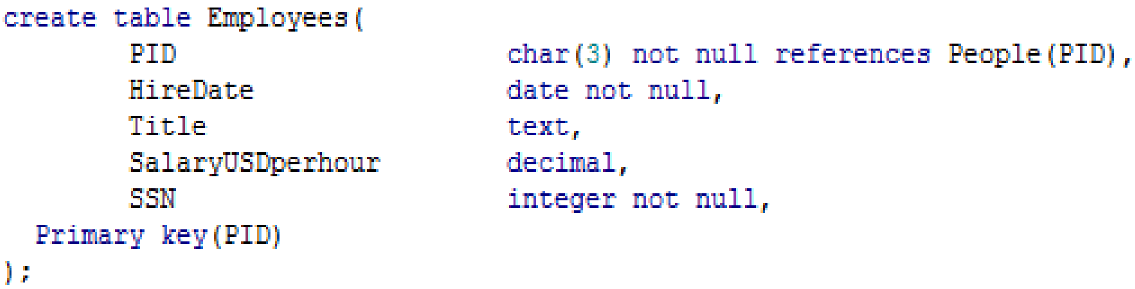


	pid character(3)	rid character(3)
1	P01	r01
2	P01	r02
3	P02	r01
4	P03	r03
5	P04	r03
6	P05	r02
7	P05	r01
8	P06	r02
9	P07	r01
10	P08	r03
11	P09	r01
12	P10	r03
13	P11	r01
14	P12	r02

EMPLOYEES TABLE:

This table will contain the required information for the employees of the grocery store other than stored in people table.

CREATE STATEMENT:

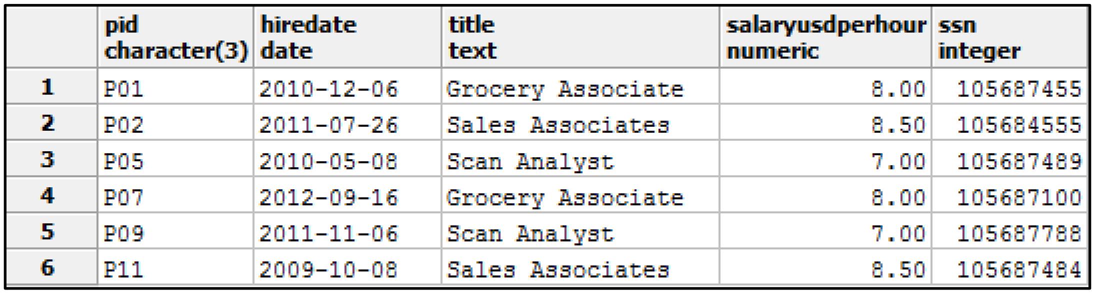


```
create table Employees(
        PID                         char(3) not null references People(PID),
        HireDate                    date not null,
        Title                       text,
        SalaryUSDperhour            decimal,
        SSN                         integer not null,
  Primary key(PID)
);
```

FUNCTIONAL DEPENDENCIES:

PID → HireDate, Title, SalaryUSDperhour, SSN

SAMPLE DATA:

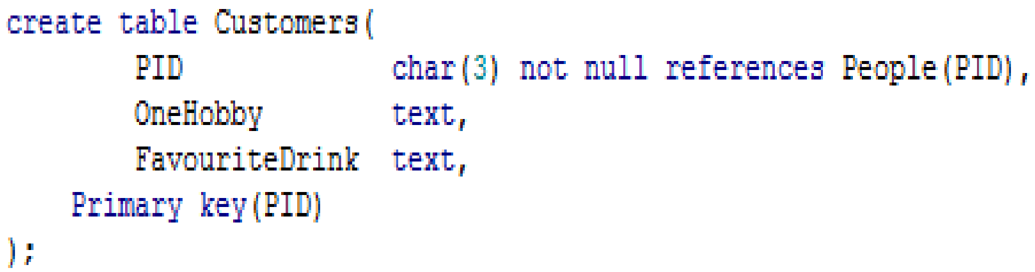


	pid	hiredate	title	salaryusdperhour	ssn
	character(3)	date	text	numeric	integer
1	P01	2010-12-06	Grocery Associate	8.00	105687455
2	P02	2011-07-26	Sales Associates	8.50	105684555
3	P05	2010-05-08	Scan Analyst	7.00	105687489
4	P07	2012-09-16	Grocery Associate	8.00	105687100
5	P09	2011-11-06	Scan Analyst	7.00	105687788
6	P11	2009-10-08	Sales Associates	8.50	105687484

CUSTOMERS TABLE:

This table will contain the information about customers of grocery store.

CREATE STATEMENT:

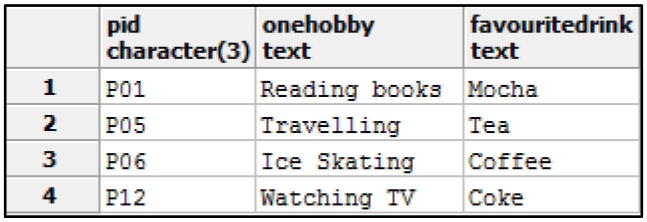


```
create table Customers(
        PID             char(3) not null references People(PID),
        OneHobby        text,
        FavouriteDrink  text,
    Primary key(PID)
);
```

FUNCTIONAL DEPENDENCIES:

PID → OneHobby, FavouriteDrink

SAMPLE DATA:

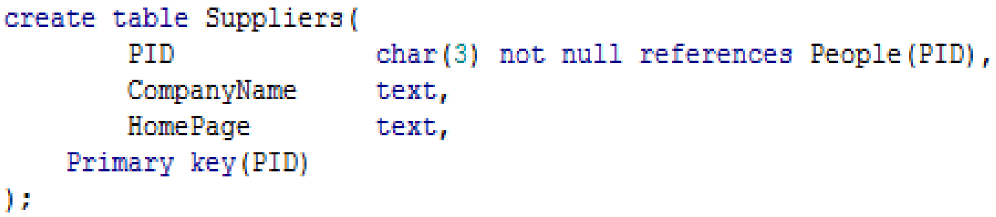


	pid character(3)	onehobby text	favouritedrink text
1	P01	Reading books	Mocha
2	P05	Travelling	Tea
3	P06	Ice Skating	Coffee
4	P12	Watching TV	Coke

SUPPLIERS TABLE:

This table will contain required information of suppliers of store.

CREATE STATEMENT:

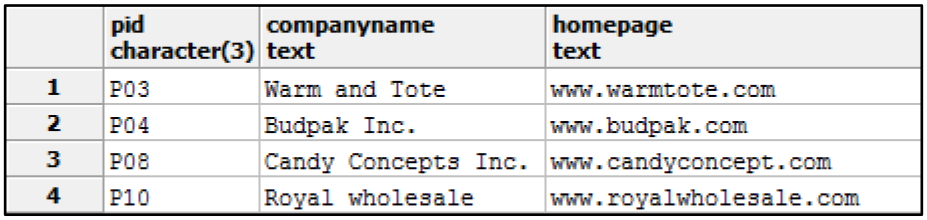


```
create table Suppliers(
        PID             char(3) not null references People(PID),
        CompanyName     text,
        HomePage        text,
    Primary key(PID)
);
```

FUNCTIONAL DEPENDENCIES:

PID → CompanyName, HomePage

SAMPLE DATA:

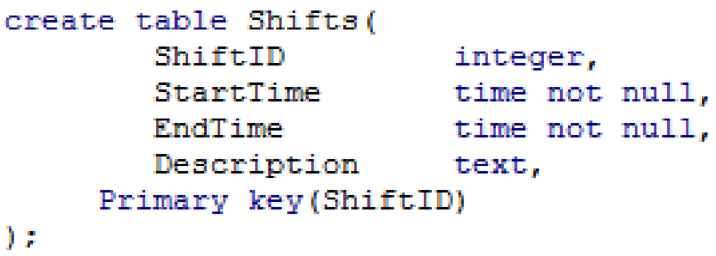


	pid character(3)	companyname text	homepage text
1	P03	Warm and Tote	www.warmtote.com
2	P04	Budpak Inc.	www.budpak.com
3	P08	Candy Concepts Inc.	www.candyconcept.com
4	P10	Royal wholesale	www.royalwholesale.com

SHIFTS TABLE:

This table will store the different work schedules that a store employee can be assigned. For instance: there are normally three shifts in store of 8 hours and it is important to know if an employee consistently be working overtime or if he is on duty but never completes his work in a timely fashion.

CREATE STATEMENT:

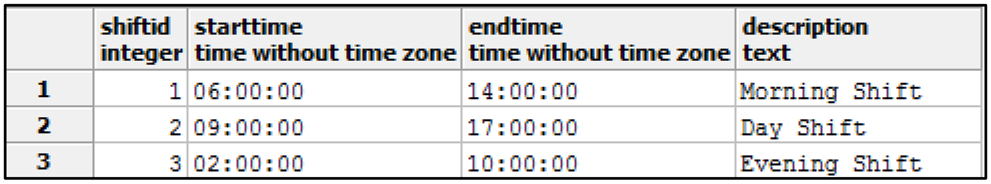


```
create table Shifts(
        ShiftID         integer,
        StartTime       time not null,
        EndTime         time not null,
        Description     text,
     Primary key(ShiftID)
);
```

FUNCTIONAL DEPENDENCIES:

ShiftID → StartTime, EndTime, Description

SAMPLE DATA:

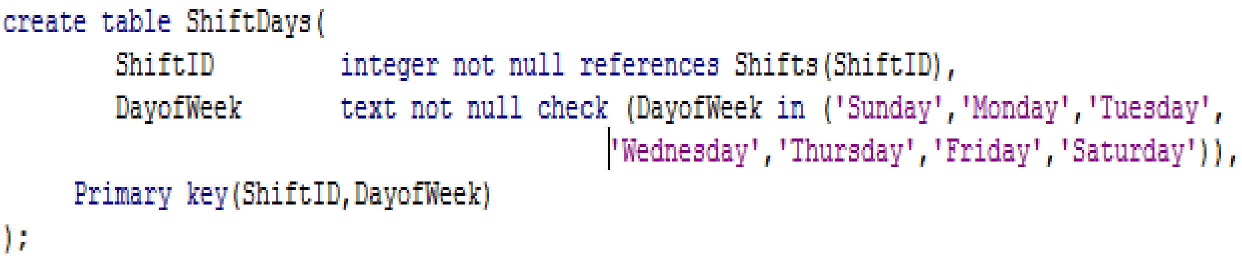


	shiftid integer	starttime time without time zone	endtime time without time zone	description text
1	1	06:00:00	14:00:00	Morning Shift
2	2	09:00:00	17:00:00	Day Shift
3	3	02:00:00	10:00:00	Evening Shift

SHIFTDAYS TABLE:

This table describes the days in which the above stated shifts are applicable.

CREATE STATEMENT:

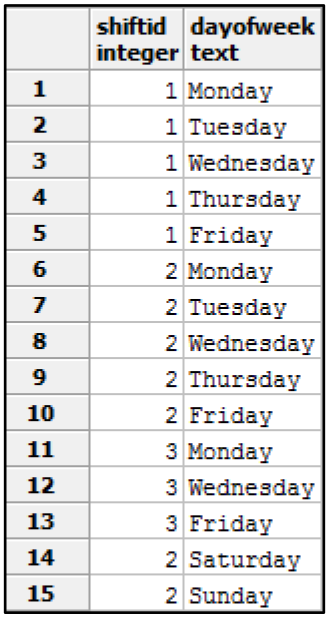


```
create table ShiftDays(
        ShiftID         integer not null references Shifts(ShiftID),
        DayofWeek       text not null check (DayofWeek in ('Sunday','Monday','Tuesday',
                                           'Wednesday','Thursday','Friday','Saturday')),
     Primary key(ShiftID,DayofWeek)
);
```

FUNCTIONAL DEPENDENCIES:

(ShiftID,DayofWeek) →

SAMPLE DATA:

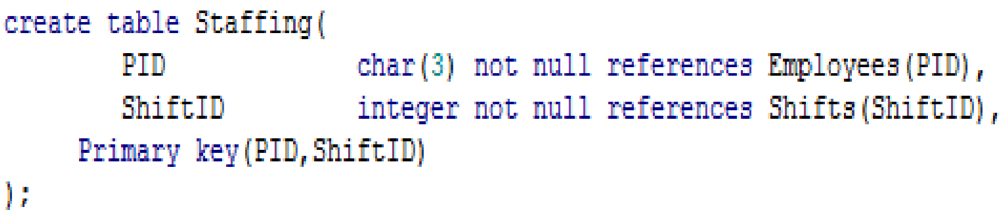


	shiftid integer	dayofweek text
1	1	Monday
2	1	Tuesday
3	1	Wednesday
4	1	Thursday
5	1	Friday
6	2	Monday
7	2	Tuesday
8	2	Wednesday
9	2	Thursday
10	2	Friday
11	3	Monday
12	3	Wednesday
13	3	Friday
14	2	Saturday
15	2	Sunday

STAFFING TABLE:

This table will include the information of which employee come in which shift.

CREATE STATEMENT:

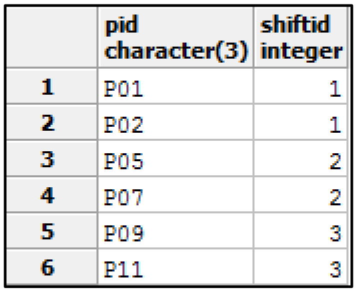


```
create table Staffing(
        PID             char(3) not null references Employees(PID),
        ShiftID         integer not null references Shifts(ShiftID),
     Primary key(PID,ShiftID)
);
```

FUNCTIONAL DEPENDENCIES:

(PID,ShiftID) →

SAMPLE DATA:

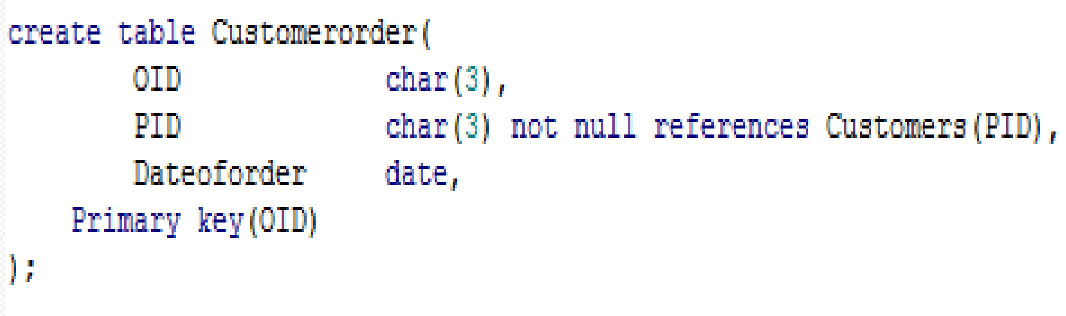


	pid character(3)	shiftid integer
1	P01	1
2	P02	1
3	P05	2
4	P07	2
5	P09	3
6	P11	3

CUSTOMERORDER TABLE:

This table will include the id of the order that a customer has made for his purchased products as well as the date of order.

CREATE STATEMENT:

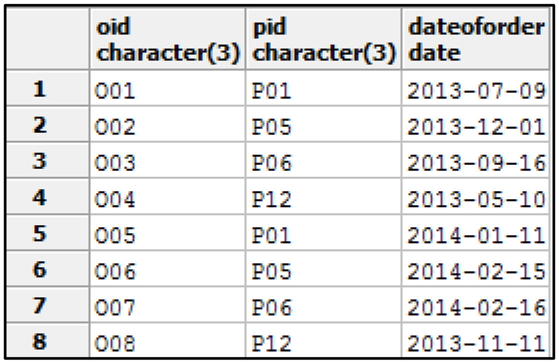


```
create table Customerorder(
        OID             char(3),
        PID             char(3) not null references Customers(PID),
        Dateoforder     date,
    Primary key(OID)
);
```

FUNCTIONAL DEPENDENCIES:

OID → PID, Dateoforder

SAMPLE DATA:

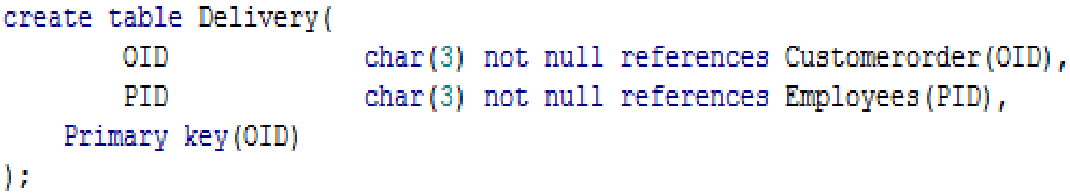


	oid character(3)	pid character(3)	dateoforder date
1	001	P01	2013-07-09
2	002	P05	2013-12-01
3	003	P06	2013-09-16
4	004	P12	2013-05-10
5	005	P01	2014-01-11
6	006	P05	2014-02-15
7	007	P06	2014-02-16
8	008	P12	2013-11-11

DELIVERY TABLE:

This table will include which employee has served which orders. Since the OID for each order will be different so only OID will be the primary key of this table.

CREATE STATEMENT:

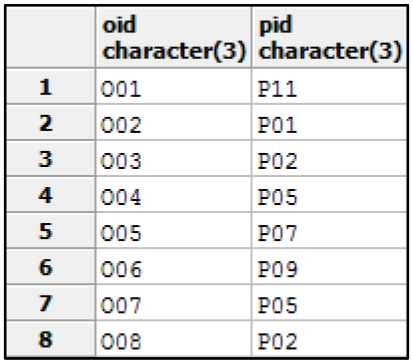


```
create table Delivery(
        OID             char(3) not null references Customerorder(OID),
        PID             char(3) not null references Employees(PID),
    Primary key(OID)
);
```

FUNCTIONAL DEPENDENCIES:

OID → PID

SAMPLE DATA:

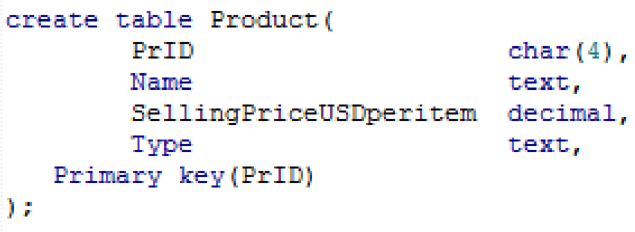


	oid character(3)	pid character(3)
1	O01	P11
2	O02	P01
3	O03	P02
4	O04	P05
5	O05	P07
6	O06	P09
7	O07	P05
8	O08	P02

PRODUCT TABLE:

This table will include the products which are available in the grocery store along with their product ID. All products will have different product ID which will be the primary key of this table.

CREATE STATEMENT:

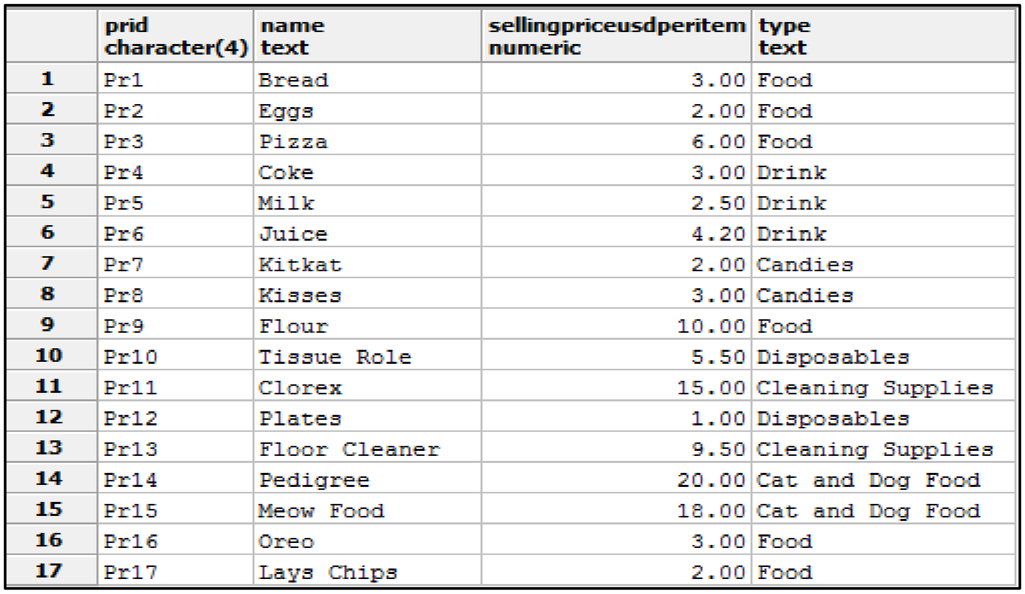


```
create table Product(
        PrID                        char(4),
        Name                        text,
        SellingPriceUSDperitem      decimal,
        Type                        text,
   Primary key(PrID)
);
```

FUNCTIONAL DEPENDENCIES:

PrID → Name, SellingPriceUSDperitem,Type

SAMPLE DATA:

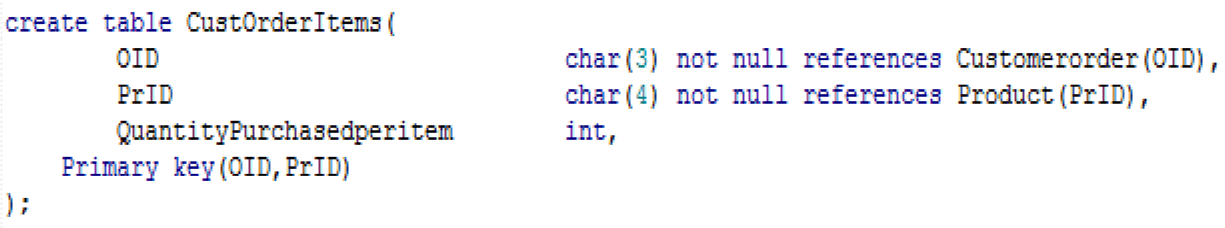


	prid character(4)	name text	sellingpriceusdperitem numeric	type text
1	Pr1	Bread	3.00	Food
2	Pr2	Eggs	2.00	Food
3	Pr3	Pizza	6.00	Food
4	Pr4	Coke	3.00	Drink
5	Pr5	Milk	2.50	Drink
6	Pr6	Juice	4.20	Drink
7	Pr7	Kitkat	2.00	Candies
8	Pr8	Kisses	3.00	Candies
9	Pr9	Flour	10.00	Food
10	Pr10	Tissue Role	5.50	Disposables
11	Pr11	Clorex	15.00	Cleaning Supplies
12	Pr12	Plates	1.00	Disposables
13	Pr13	Floor Cleaner	9.50	Cleaning Supplies
14	Pr14	Pedigree	20.00	Cat and Dog Food
15	Pr15	Meow Food	18.00	Cat and Dog Food
16	Pr16	Oreo	3.00	Food
17	Pr17	Lays Chips	2.00	Food

CUSTORDERITEMS TABLE:

This table will include the items that a customer has ordered in an order along with the quantity purchased.

CREATE STATEMENT:

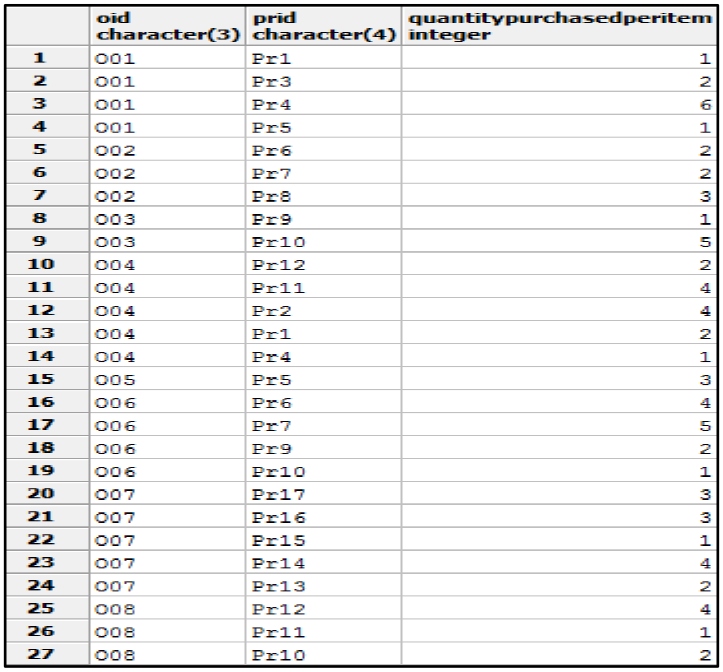


```
create table CustOrderItems(
        OID                             char(3) not null references Customerorder(OID),
        PrID                            char(4) not null references Product(PrID),
        QuantityPurchasedperitem        int,
    Primary key(OID,PrID)
);
```

FUNCTIONAL DEPENDENCIES:

(OID,PrID) → QuantityPurchasedperitem

SAMPLE DATA:

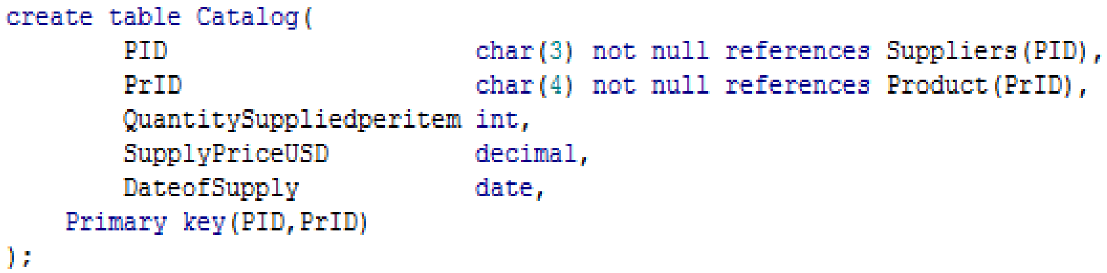


	oid character(3)	prid character(4)	quantitypurchasedperitem integer
1	001	Pr1	1
2	001	Pr3	2
3	001	Pr4	6
4	001	Pr5	1
5	002	Pr6	2
6	002	Pr7	2
7	002	Pr8	3
8	003	Pr9	1
9	003	Pr10	5
10	004	Pr12	2
11	004	Pr11	4
12	004	Pr2	4
13	004	Pr1	2
14	004	Pr4	1
15	005	Pr5	3
16	006	Pr6	4
17	006	Pr7	5
18	006	Pr9	2
19	006	Pr10	1
20	007	Pr17	3
21	007	Pr16	3
22	007	Pr15	1
23	007	Pr14	4
24	007	Pr13	2
25	008	Pr12	4
26	008	Pr11	1
27	008	Pr10	2

CATALOG TABLE:

This table will include the information about which product has been supplied by which Supplier alongwith the quantity , price and date.

CREATE STATEMENT:

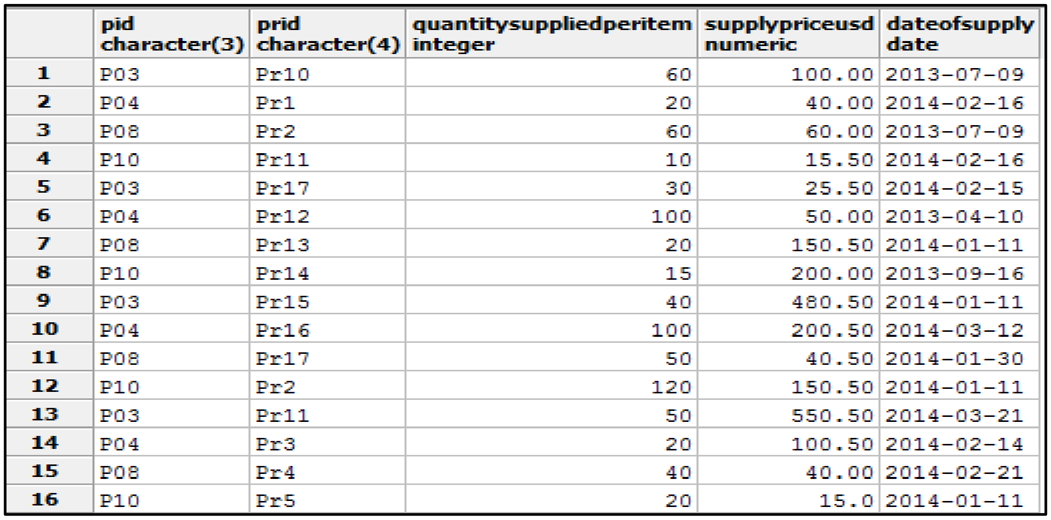


```
create table Catalog(
        PID                     char(3) not null references Suppliers(PID),
        PrID                    char(4) not null references Product(PrID),
        QuantitySuppliedperitem int,
        SupplyPriceUSD          decimal,
        DateofSupply            date,
    Primary key(PID,PrID)
);
```

FUNCTIONAL DEPENDENCIES:

(PID,PrID) → QuantitySuppliedperitem, SupplyPriceUSD,DateofSupply

SAMPLE DATA:

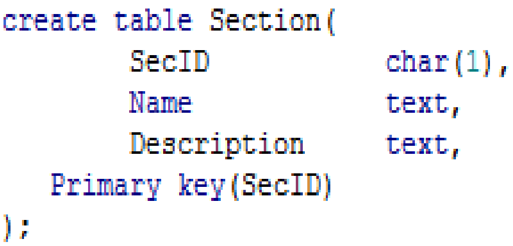


	pid character(3)	prid character(4)	quantitysuppliedperitem integer	supplypriceusd numeric	dateofsupply date
1	P03	Pr10	60	100.00	2013-07-09
2	P04	Pr1	20	40.00	2014-02-16
3	P08	Pr2	60	60.00	2013-07-09
4	P10	Pr11	10	15.50	2014-02-16
5	P03	Pr17	30	25.50	2014-02-15
6	P04	Pr12	100	50.00	2013-04-10
7	P08	Pr13	20	150.50	2014-01-11
8	P10	Pr14	15	200.00	2013-09-16
9	P03	Pr15	40	480.50	2014-01-11
10	P04	Pr16	100	200.50	2014-03-12
11	P08	Pr17	50	40.50	2014-01-30
12	P10	Pr2	120	150.50	2014-01-11
13	P03	Pr11	50	550.50	2014-03-21
14	P04	Pr3	20	100.50	2014-02-14
15	P08	Pr4	40	40.00	2014-02-21
16	P10	Pr5	20	15.0	2014-01-11

SECTION TABLE:

This table will include the information about different sections in store.

CREATE STATEMENT:

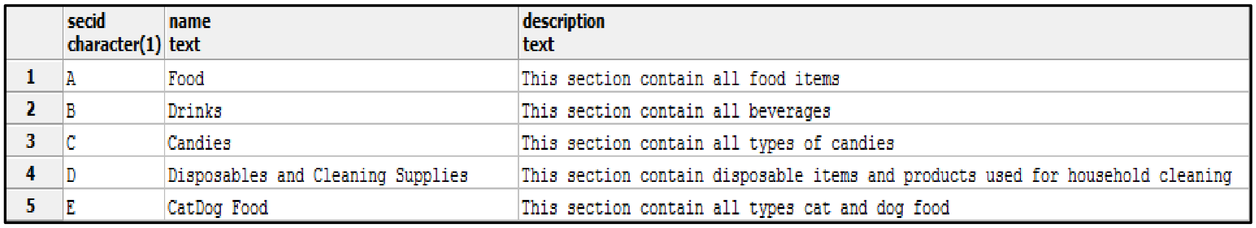


```
create table Section(
        SecID           char(1),
        Name            text,
        Description     text,
   Primary key(SecID)
);
```

FUNCTIONAL DEPENDENCIES:

SecID → Name, Description

SAMPLE DATA:

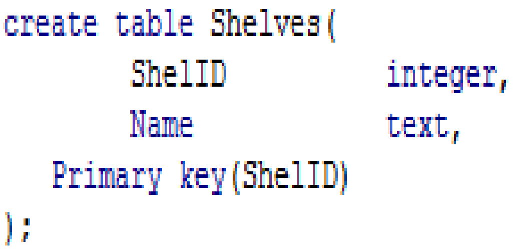


	secid	name	description
	character(1)	text	text
1	A	Food	This section contain all food items
2	B	Drinks	This section contain all beverages
3	C	Candies	This section contain all types of candies
4	D	Disposables and Cleaning Supplies	This section contain disposable items and products used for household cleaning
5	E	CatDog Food	This section contain all types cat and dog food

SHELVES TABLE:

This table will include the shelves information a section can have in store.

CREATE STATEMENT:

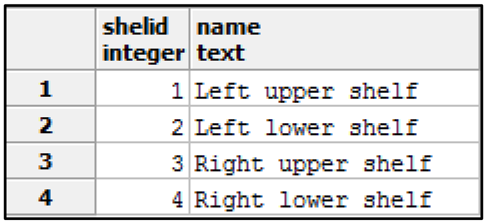


```
create table Shelves(
        ShelID          integer,
        Name            text,
   Primary key(ShelID)
);
```

FUNCTIONAL DEPENDENCIES:

ShelID → Name

SAMPLE DATA:

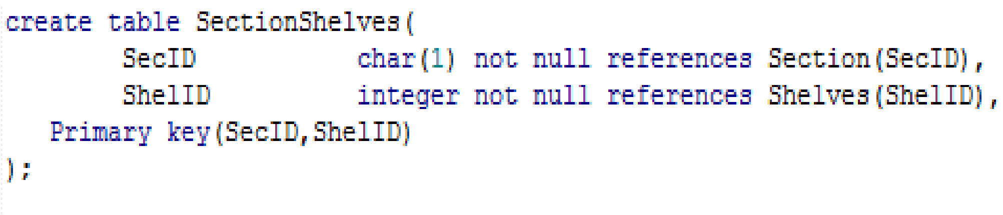


	shelid integer	name text
1	1	Left upper shelf
2	2	Left lower shelf
3	3	Right upper shelf
4	4	Right lower shelf

SECTIONSHELVES TABLE:

This table will contain information about how many shelves a section has.

CREATE STATEMENT:

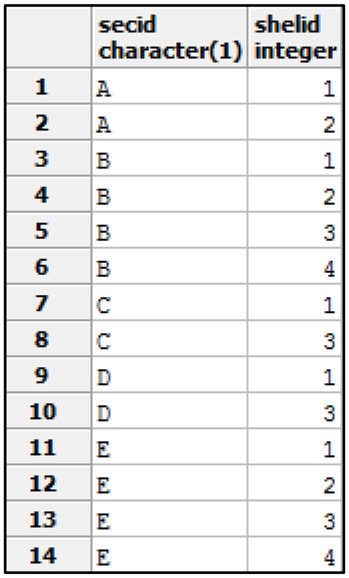


```
create table SectionShelves(
        SecID           char(1) not null references Section(SecID),
        ShelID          integer not null references Shelves(ShelID),
   Primary key(SecID,ShelID)
);
```

FUNCTIONAL DEPENDENCIES:

(SecID,ShelID) →

SAMPLE DATA:

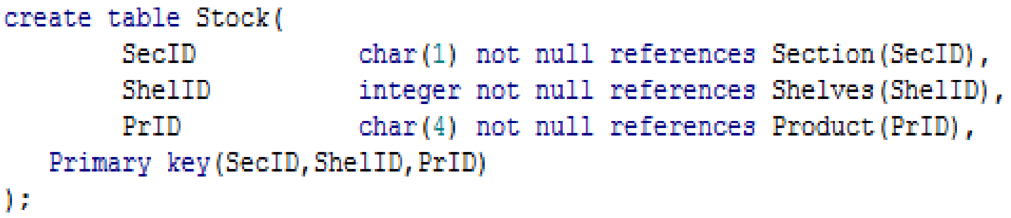


	secid character(1)	shelid integer
1	A	1
2	A	2
3	B	1
4	B	2
5	B	3
6	B	4
7	C	1
8	C	3
9	D	1
10	D	3
11	E	1
12	E	2
13	E	3
14	E	4

STOCK TABLE:

This table will contain the information about which product is available in which section and on which shelf.

CREATE STATEMENT:

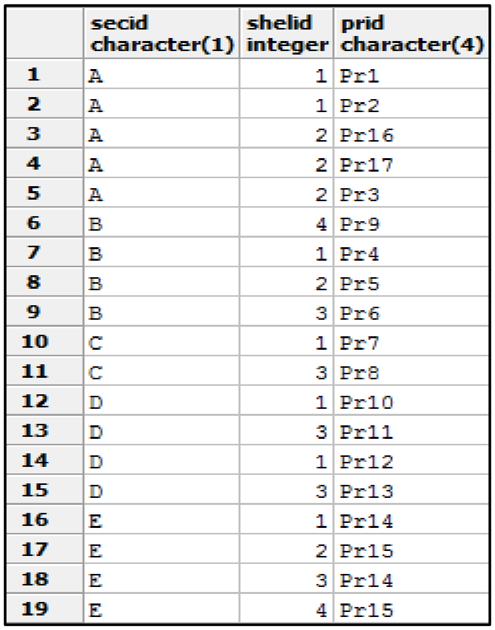


```
create table Stock(
        SecID           char(1) not null references Section(SecID),
        ShelID          integer not null references Shelves(ShelID),
        PrID            char(4) not null references Product(PrID),
   Primary key(SecID,ShelID,PrID)
);
```

FUNCTIONAL DEPENDENCIES:

(SecID,ShelID,PrID) →

SAMPLE DATA:



	secid	shelid	prid
	character(1)	integer	character(4)
1	A	1	Pr1
2	A	1	Pr2
3	A	2	Pr16
4	A	2	Pr17
5	A	2	Pr3
6	B	4	Pr9
7	B	1	Pr4
8	B	2	Pr5
9	B	3	Pr6
10	C	1	Pr7
11	C	3	Pr8
12	D	1	Pr10
13	D	3	Pr11
14	D	1	Pr12
15	D	3	Pr13
16	E	1	Pr14
17	E	2	Pr15
18	E	3	Pr14
19	E	4	Pr15

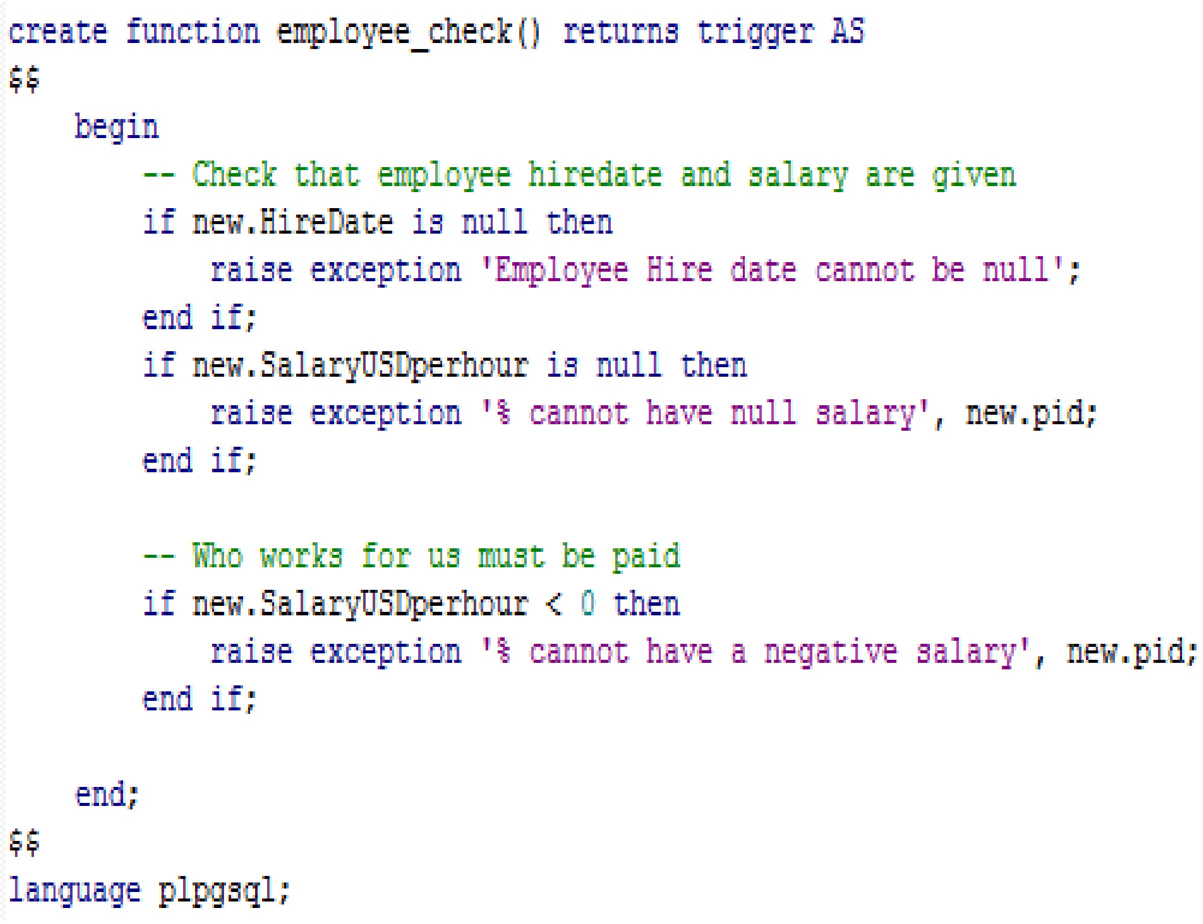
TRIGGER

A trigger is a special kind of stored procedure that automatically executes when an event occurs in the **database** server. DML triggers execute when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view.

EMPLOYEE_CHECK()

PURPOSE:

The below trigger ensures that any time a row is inserted or updated in the employee table, it checks that the employee PID already present in the table and his salary is a positive value.

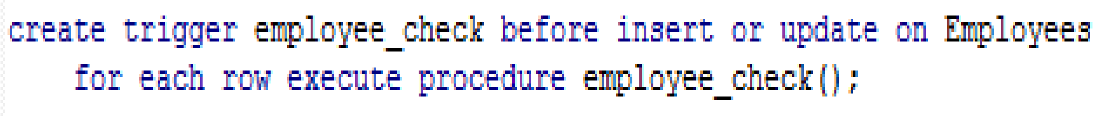


```
create function employee_check() returns trigger AS
$$
    begin
        -- Check that employee hiredate and salary are given
        if new.HireDate is null then
            raise exception 'Employee Hire date cannot be null';
        end if;
        if new.SalaryUSDperhour is null then
            raise exception '% cannot have null salary', new.pid;
        end if;

        -- Who works for us must be paid
        if new.SalaryUSDperhour < 0 then
            raise exception '% cannot have a negative salary', new.pid;
        end if;

    end;
$$
language plpgsql;
```

Usage:



```
create trigger employee_check before insert or update on Employees
    for each row execute procedure employee_check();
```

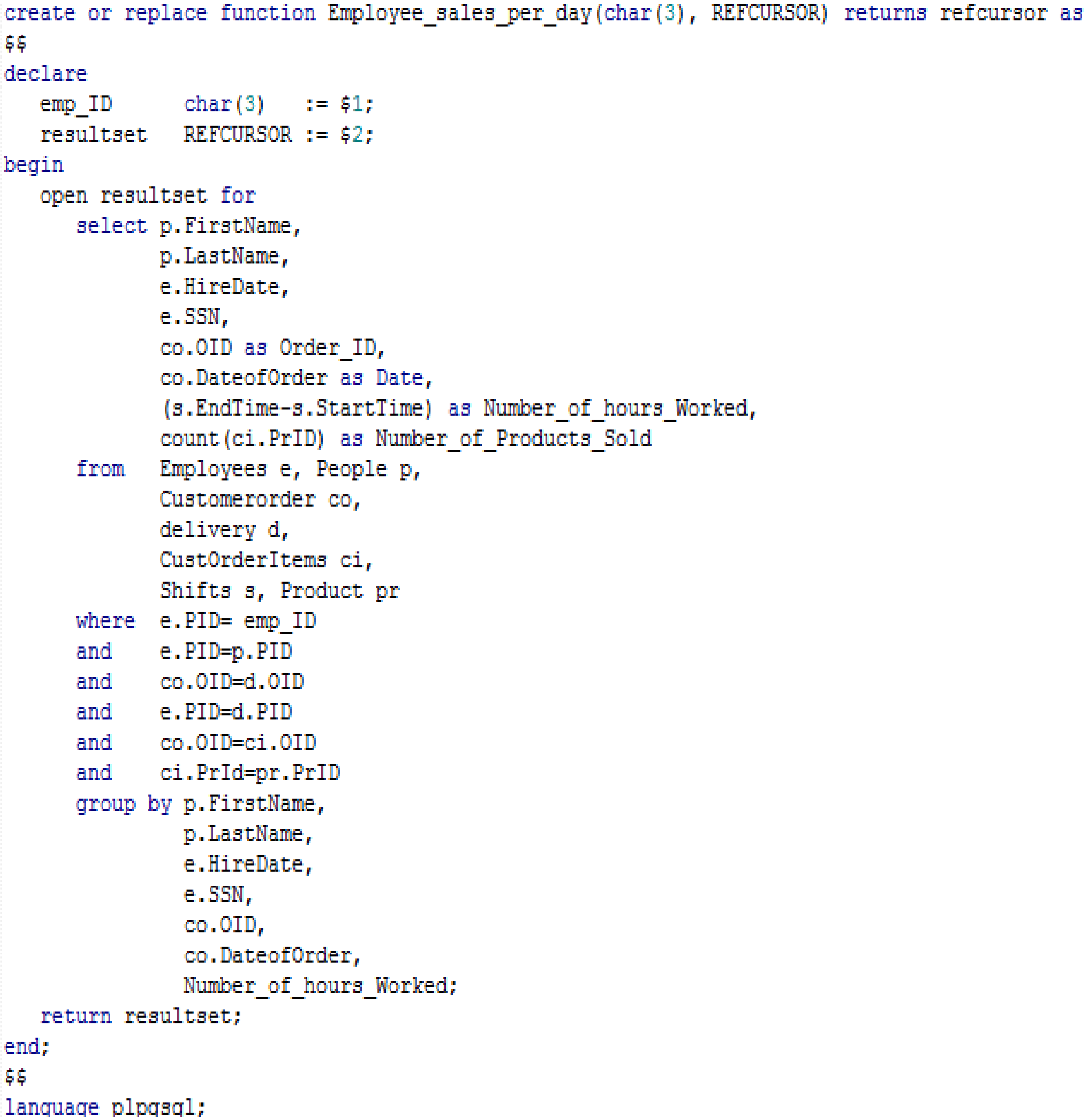
STORED PROCEDURE:

Stored procedure in any database is the collection of queries that needs to be executed on daily basis. In postgres, they are written in PL/pgSQL language and are called functions like triggers.

EMPLOYEE_SALES_PER_DAY

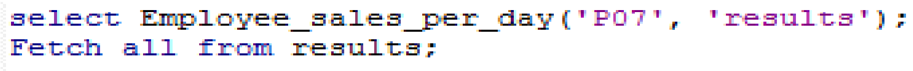
PURPOSE:

This stored procedure will help in getting the information related to employee like how many customers he has delivered, his salary and hours he worked. It will take employee id as input and gives us the required output:



```
create or replace function Employee_sales_per_day(char(3), REFCURSOR) returns refcursor as
$$
declare
   emp_ID      char(3)   := $1;
   resultset   REFCURSOR := $2;
begin
   open resultset for
      select p.FirstName,
             p.LastName,
             e.HireDate,
             e.SSN,
             co.OID as Order_ID,
             co.DateofOrder as Date,
             (s.EndTime-s.StartTime) as Number_of_hours_Worked,
             count(ci.PrID) as Number_of_Products_Sold
      from   Employees e, People p,
             Customerorder co,
             delivery d,
             CustOrderItems ci,
             Shifts s, Product pr
      where  e.PID= emp_ID
      and    e.PID=p.PID
      and    co.OID=d.OID
      and    e.PID=d.PID
      and    co.OID=ci.OID
      and    ci.PrId=pr.PrID
      group by p.FirstName,
               p.LastName,
               e.HireDate,
               e.SSN,
               co.OID,
               co.DateofOrder,
               Number_of_hours_Worked;
   return resultset;
end;
$$
language plpgsql;
```

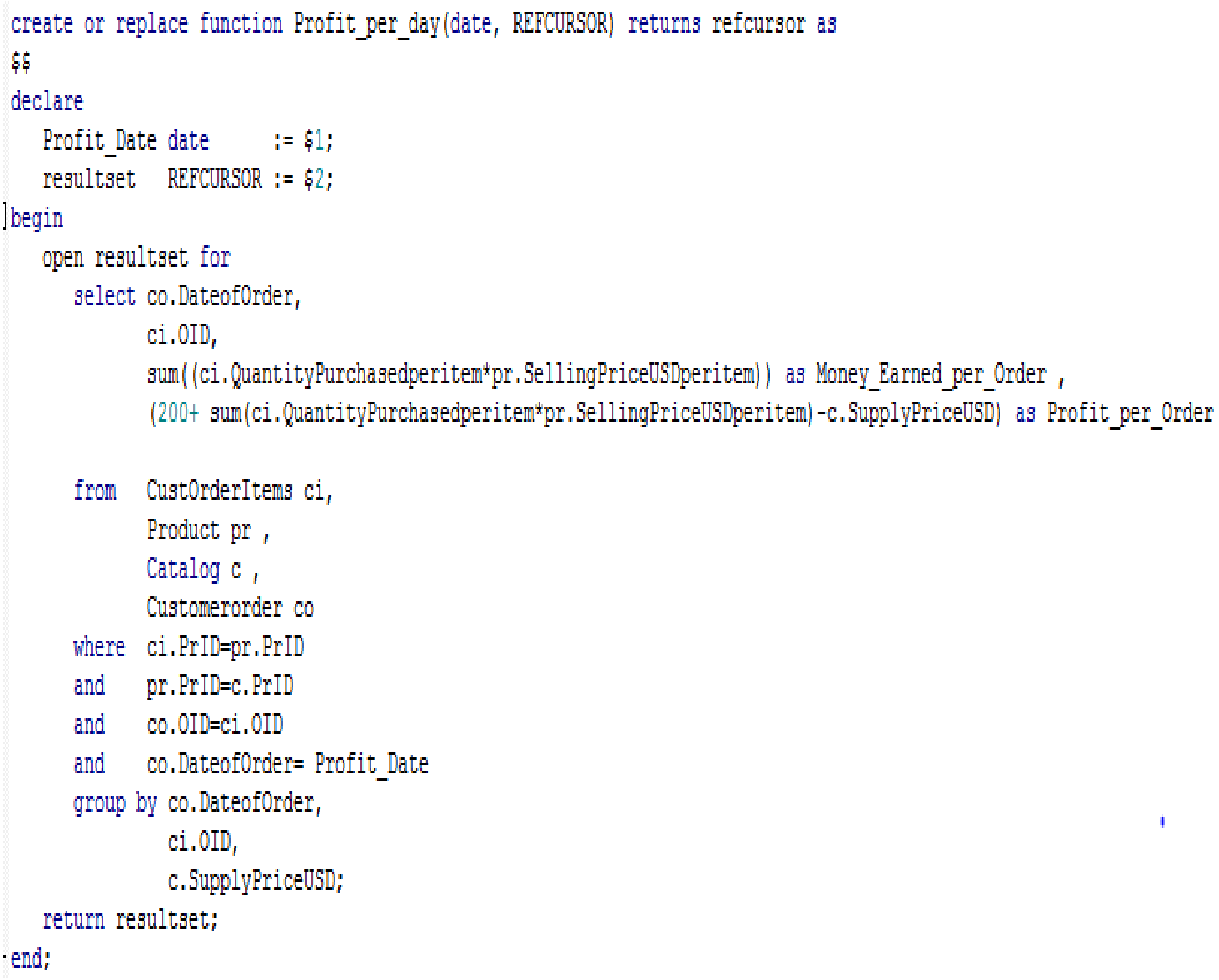
USAGE:



```
select Employee_sales_per_day('P07', 'results');
Fetch all from results;
```

PROFIT_PER_DAY

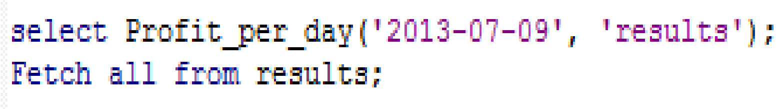
This Stored procedure will help us to calculate the profit of the day. It will take date as input and calculate the profit as required. It is an assumption here that the owner starts the opening with 200$.



```
create or replace function Profit_per_day(date, REFCURSOR) returns refcursor as
$$
declare
   Profit_Date date      := $1;
   resultset   REFCURSOR := $2;
begin
   open resultset for
      select co.DateofOrder,
             ci.OID,
             sum((ci.QuantityPurchasedperitem*pr.SellingPriceUSDperitem)) as Money_Earned_per_Order ,
             (200+ sum(ci.QuantityPurchasedperitem*pr.SellingPriceUSDperitem)-c.SupplyPriceUSD) as Profit_per_Order

      from   CustOrderItems ci,
             Product pr ,
             Catalog c ,
             Customerorder co
      where  ci.PrID=pr.PrID
      and    pr.PrID=c.PrID
      and    co.OID=ci.OID
      and    co.DateofOrder= Profit_Date
      group by co.DateofOrder,
               ci.OID,
               c.SupplyPriceUSD;
   return resultset;
end;
```

USAGE:



```
select Profit_per_day('2013-07-09', 'results');
Fetch all from results;
```

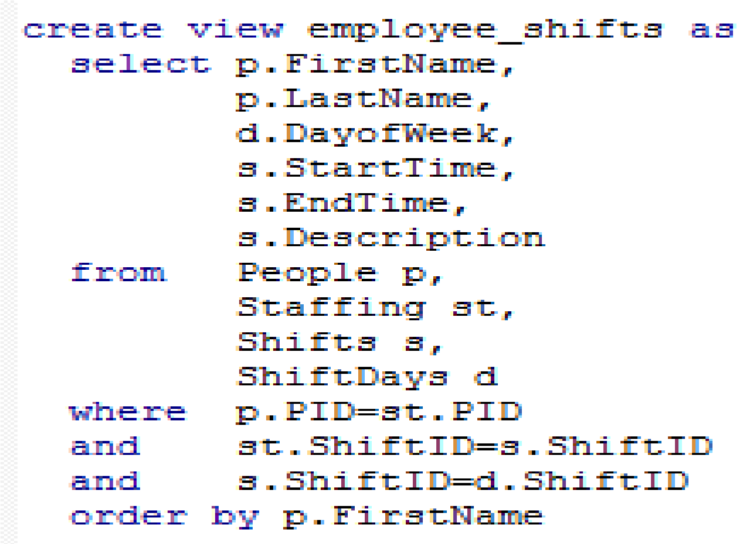
VIEWS:

A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

EMPLOYEE_SHIFTS

PURPOSE:

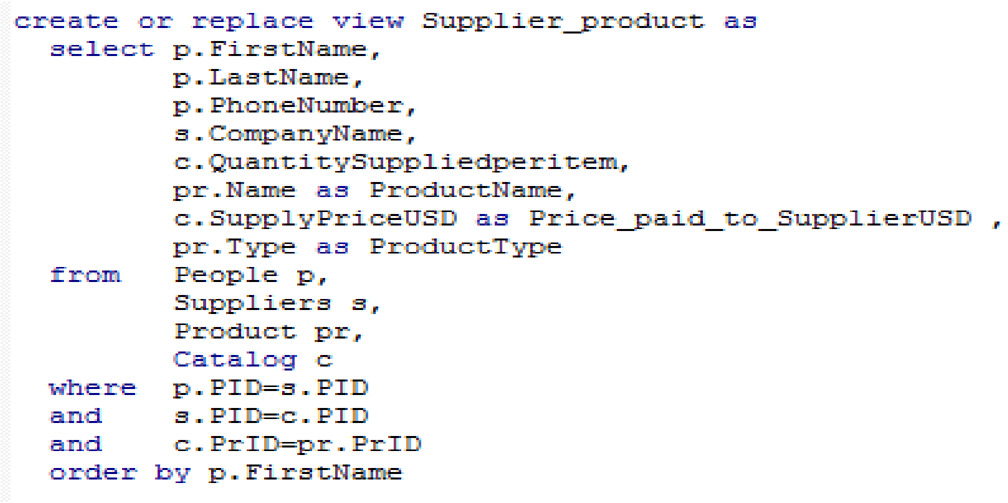
This view will give us the clear picture of the shifts of employee when an employee has to work.



```
create view employee_shifts as
  select p.FirstName,
         p.LastName,
         d.DayofWeek,
         s.StartTime,
         s.EndTime,
         s.Description
  from   People p,
         Staffing st,
         Shifts s,
         ShiftDays d
  where  p.PID=st.PID
  and    st.ShiftID=s.ShiftID
  and    s.ShiftID=d.ShiftID
  order by p.FirstName
```

SUPPLIER_PRODUCT:

This view helps in getting information about which supplier has supplied which products, at what price and in what quantity.

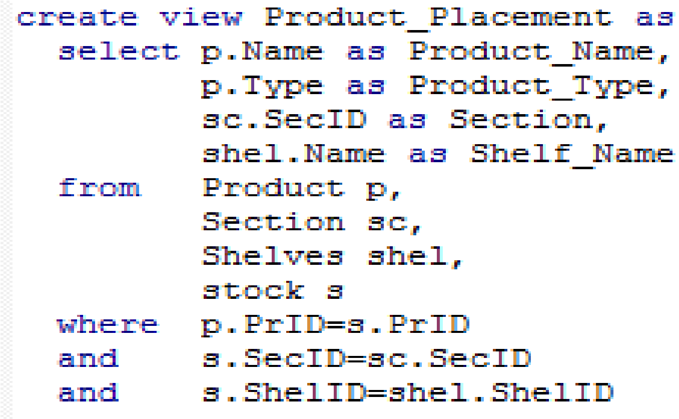


```
create or replace view Supplier_product as
  select p.FirstName,
         p.LastName,
         p.PhoneNumber,
         s.CompanyName,
         c.QuantitySuppliedperitem,
         pr.Name as ProductName,
         c.SupplyPriceUSD as Price_paid_to_SupplierUSD ,
         pr.Type as ProductType
  from   People p,
         Suppliers s,
         Product pr,
         Catalog c
  where  p.PID=s.PID
  and    s.PID=c.PID
  and    c.PrID=pr.PrID
  order by p.FirstName
```

PRODUCT_PLACEMENT:

PURPOSE:

This view help us in getting information about which product is placed on which section shelf.

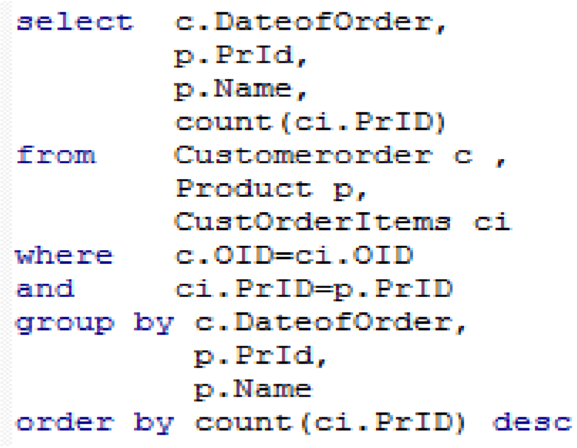


```
create view Product_Placement as
  select p.Name as Product_Name,
         p.Type as Product_Type,
         sc.SecID as Section,
         shel.Name as Shelf_Name
  from   Product p,
         Section sc,
         Shelves shel,
         stock s
  where  p.PrID=s.PrID
  and    s.SecID=sc.SecID
  and    s.ShelID=shel.ShelID
```

REPORTS:

HIGHEST SALES PER DAY:

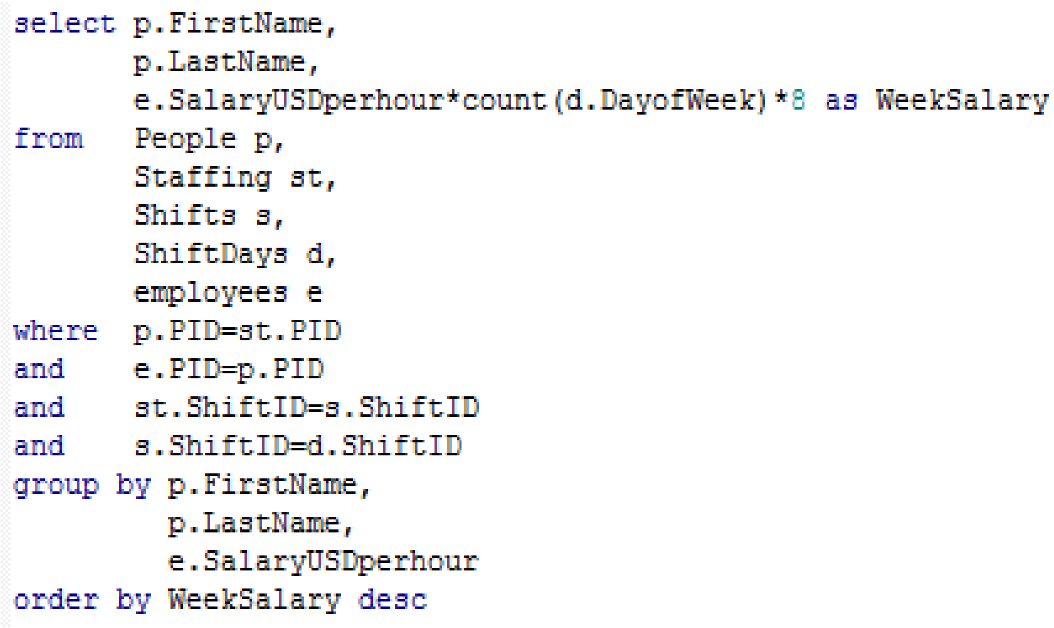
This report will help in getting information about highest sales per day product wise.



```
select  c.DateofOrder,
        p.PrId,
        p.Name,
        count(ci.PrID)
from    Customerorder c ,
        Product p,
        CustOrderItems ci
where   c.OID=ci.OID
and     ci.PrID=p.PrID
group by c.DateofOrder,
         p.PrId,
         p.Name
order by count(ci.PrID) desc
```

SALARY OF EMPLOYEE PER WEEK:

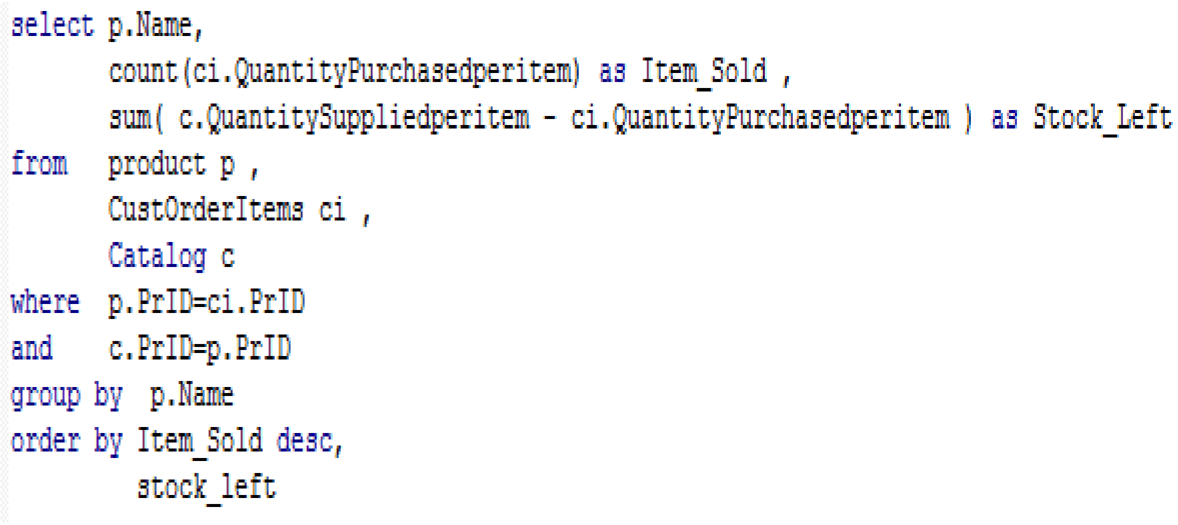
This report will help in getting salary of employees on weekly basis.



```
select p.FirstName,
       p.LastName,
       e.SalaryUSDperhour*count(d.DayofWeek)*8 as WeekSalary
from   People p,
       Staffing st,
       Shifts s,
       ShiftDays d,
       employees e
where  p.PID=st.PID
and    e.PID=p.PID
and    st.ShiftID=s.ShiftID
and    s.ShiftID=d.ShiftID
group by p.FirstName,
         p.LastName,
         e.SalaryUSDperhour
order by WeekSalary desc
```

PRODUCTS STOCK:

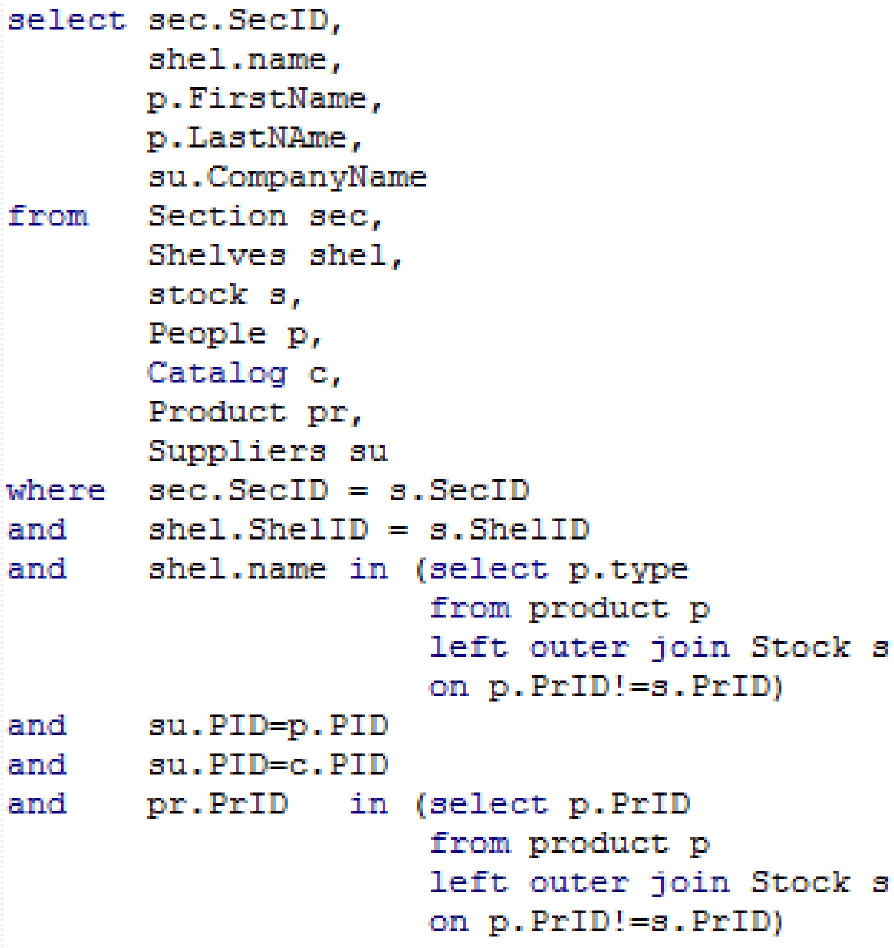
This report will help in getting the product stock available at store. This will be useful while purchasing products from suppliers.



```
select p.Name,
       count(ci.QuantityPurchasedperitem) as Item_Sold ,
       sum( c.QuantitySuppliedperitem - ci.QuantityPurchasedperitem ) as Stock_Left
from   product p ,
       CustOrderItems ci ,
       Catalog c
where  p.PrID=ci.PrID
and    c.PrID=p.PrID
group by  p.Name
order by Item_Sold desc,
         stock_left
```

PRODUCTS WHICH ARE NOT PLACED IN SHELF AND FROM WHICH SUPPLIER THEY HAVE COME:

This report will help in getting information about the products which are not placed into any section shelf and from which supplier they have been purchased.



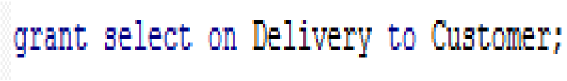
```
select sec.SecID,
       shel.name,
       p.FirstName,
       p.LastNAme,
       su.CompanyName
from   Section sec,
       Shelves shel,
       stock s,
       People p,
       Catalog c,
       Product pr,
       Suppliers su
where  sec.SecID = s.SecID
and    shel.ShelID = s.ShelID
and    shel.name in (select p.type
                     from product p
                     left outer join Stock s
                     on p.PrID!=s.PrID)
and    su.PID=p.PID
and    su.PID=c.PID
and    pr.PrID   in (select p.PrID
                     from product p
                     left outer join Stock s
                     on p.PrID!=s.PrID)
```

SECURITY:

Security is the most important part of every database. There are four people who are will interact with the database on daily basis. They would be required to provide access to database. So roles will be created for each category and assigned to person in that category.

CUSTOMER:

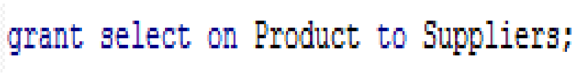
Customer are interested in Delivery table only to check the employees how are free at the moment.



```
grant select on Delivery to Customer;
```

SUPPLIERS:

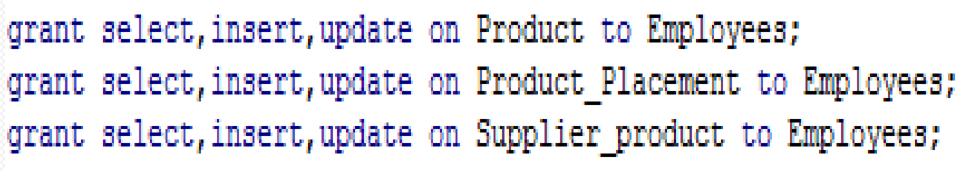
Suppliers will be interested in Product table to check the type of product the store sells.



```
grant select on Product to Suppliers;
```

EMPLOYEES:

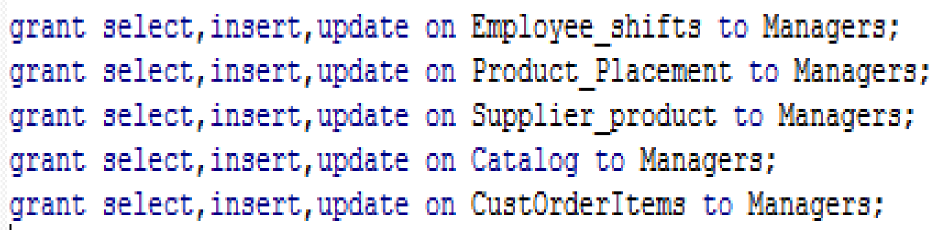
They will need to access on Products table as well as the views Product_Placement and Supplier_product



```
grant select,insert,update on Product to Employees;
grant select,insert,update on Product_Placement to Employees;
grant select,insert,update on Supplier_product to Employees;
```

MANAGER:

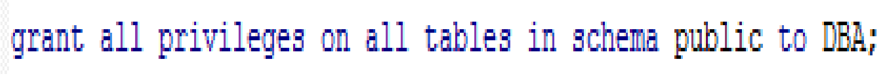
They will be interested in views like employee_shifts, Product_Placement and Supplier_product and ofcourse the tables like Catalog, CustorderItems.



```
grant select,insert,update on Employee_shifts to Managers;
grant select,insert,update on Product_Placement to Managers;
grant select,insert,update on Supplier_product to Managers;
grant select,insert,update on Catalog to Managers;
grant select,insert,update on CustOrderItems to Managers;
```

DBA (DATABASE ADMINISTRATORS):

They will be interested in all tables as they need to control everything.



```
grant all privileges on all tables in schema public to DBA;
```

Implementation notes

The following things need to be considered for implementation:

- The store can change the salary of employees according to their profit and performance of employee. Similarly, the prices of products are not fixed; they will be changed with market position.
- The shifts of employees need to be given in 24 hour format else the number of hours will come out in negative. SSN and Title of employees should be given according to the SSA norms and store structure respectively.
- The shelves of section will keep moving depending upon the factors like demand, festival, occasions or discount season.
- Quantity in this store database has been taken as pound but that cannot be common among all products.

KNOWN PROBLEM:

- Only DBA have access to change the information in the tables. This means an employee cannot insert any data into customer table while delivering product to customer.
- Only one employee can serve a customer at a time which leads to other employees sitting idle.
- Storing this much information about customers is not enough. If they will pay by credit card, many more things need to be checked before selling product like credit history, background, identification number etc.
- More roles and views need to be created to protect the data in the tables.

FUTURE ENHANCEMENTS:

- When store will be expanded, it will need extra employees which are very easy to add as common information will go in people table and rest will go in employee table.
- Similarly products need to be added with the store extension which can be added directly to products table and placed in storeshelves table.
- Allowing an employee to work in two shifts if needed.
- Assigning a team leader in case owner/manager not present in the store.
- Information about carts can also be added if necessary.
- Payout table can be added to keep track of money. For instance, money paid to electrician for correcting the wire fault should be recorded somewhere to calculate correct profit.