**There are five workouts**

**To be completed before Sunday 30-Jan-2021**

**Workout-5 (Postgresql DVDRENTAL Database)**

**Problem Statement**: Write a function which will accept two inputs and return a grade depending on the inputs provided with the below criteria.

**Function Name**: calculate\_grade

**Inputs:**

1. rate numeric
2. cost numeric

**Logic:**

|  |  |  |
| --- | --- | --- |
| value of "rate" | value of "cost" | grade \* |
| less than 1.00 | less than 10 | A |
| less than 1.00 | 10 or more but less than 20 | B |
| less than 1.00 | 20 or more | C |
| 1 or more but less than 4 | less than 12 | D |
| 1 or more but less than 4 | 12 or more but less than 24 | E |
| 1 or more but less than 4 | 24 or more | F |
| 4 or more | less than 13.5 | G |
| 4 or more | 13.5 or more but less than 19.2 | H |
| 4 or more | 19.2 or more | K |

\* A is minimum and K is maximum grade

CREATE FUNCTION public.calculate\_grade(IN rate numeric, IN cost numeric)

RETURNS "char"

LANGUAGE 'plpgsql'

AS $BODY$declare

grade char;

begin

if rate<1 then

if cost<10 then

grade='A';

elsif cost>=10 and cost<20 then

grade='B';

else

grade='C';

end if;

elsif rate>=1 and rate<4 then

if cost<12 then

grade='D';

elsif cost>=12 and cost<24 then

grade='E';

else

grade='F';

end if;

else

if cost<13.5 then

grade='G';

elsif cost>=13.5 and cost<19.2 then

grade='H';

else

grade='K';

end if;

end if;

return grade;

end;

$BODY$;

ALTER FUNCTION public.calculate\_grade(numeric, numeric)

OWNER TO postgres;

**Workout-6 (Postgresql DVDRENTAL Database)**

**Problem Statement**: Write a procedure which will populate a target table for all actor information stated below.

**Target table name:** actor\_performance

**Target table columns:**

|  |  |  |
| --- | --- | --- |
| column name | column type | description |
| actor\_id | integer | actor\_id from actor table |
| actor\_name | character varying | concatenation of first\_name and last\_name from actor |
| max\_grade | character(1) | any value from A to K |
| min\_grade | character(1) | any value from A to K |
| ratings | character varying | comma separated ratings from film table |

**Hint**: Use tables – actor, film\_actor, film

For each actor, find all the films done by that actor and then find all the grades from the above function, *calculate\_grade*, using rental\_rate and replacement\_cost. Between the grades, A is the min grade and K is the maximum grade. Use these values to store min\_grade and max\_grade columns of *actor\_performance* table. Also find the comma separated values of rating from *film* table per actor.

CREATE OR REPLACE PROCEDURE public.actor\_data()

LANGUAGE 'plpgsql'

AS $BODY$declare

begin

delete from actor\_performance;

commit;

insert into actor\_performance(actor\_id,actor\_name,min\_grade,max\_grade,ratings)

select actorid,actor\_name,min\_grade,max\_grade

,( select string\_agg(cast(r.rating as varchar),',' order by r.rating)

from (select distinct actor.actor\_id,film.rating

from actor,film\_actor,film

where actor.actor\_id=film\_actor.actor\_id

and film\_actor.film\_id=film.film\_id

and actor.actor\_id=actorid

) r

) ratings

from (

select actor.actor\_id actorid

,actor.first\_name||' '||actor.last\_name actor\_name

,min(calculate\_grade(film.rental\_rate,film.replacement\_cost)) min\_grade

,max(calculate\_grade(film.rental\_rate,film.replacement\_cost)) max\_grade

from actor,film\_actor,film

where

actor.actor\_id=film\_actor.actor\_id

and film\_actor.film\_id=film.film\_id

group by

actorid

,actor\_name

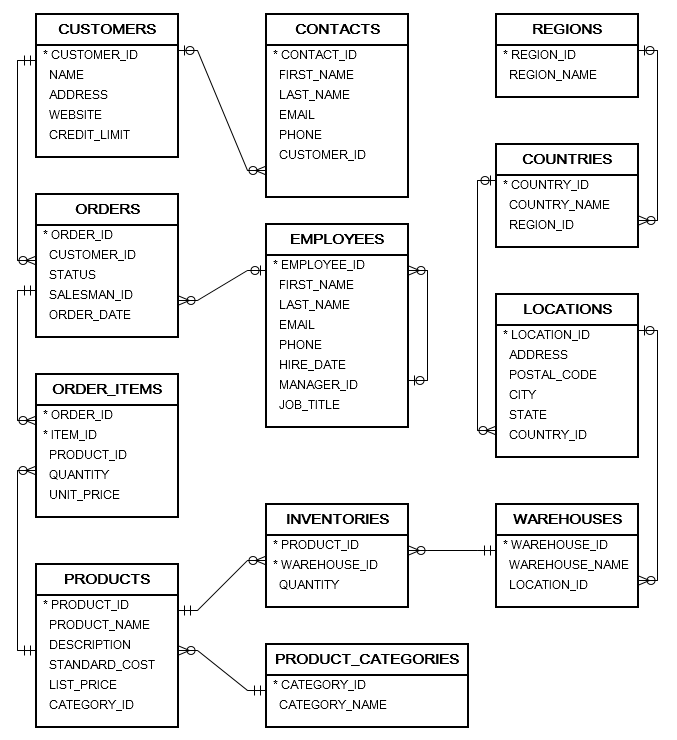
) v

order by actorid;

commit;

end;$BODY$;

**Use the below schema diagram for better understanding of oracle table associations.**



**Workout-7 (Oracle Sample Data, Oracle database)**

**Problem Statement**: Write a function which take input two values and return a string.

**Function name:** get\_assesment

**Input:** amount numeric, ranges numeric

**Output:** Function will take the amount and ranges as inputs and find those amounts in list\_price of products table such that list\_price lies between amount+ranges and amount-ranges. After finding the rows from products table, it will find the category\_names using category\_id along with company names from product\_name (use 1st word). Concatenate the category\_name and company name using "@" sign [category\_name@ company name]. All these strings will be concatenated using "OR" and return the value.

**Hint**: Use tables – products, product\_categories,

create or replace FUNCTION get\_assesment (amount NUMERIC, ranges NUMERIC)

RETURN clob

AS

result clob;

BEGIN

begin

select listagg(company\_name||'@'||category\_name,' OR ')

within group (order by company\_name) into result

from

(select distinct trim(substr(p.product\_name,1,decode(instr(p.product\_name,' '),0,length(p.product\_name),instr(p.product\_name,' ')))) company\_name, c.category\_name

from products p, product\_categories c

where p.category\_id=c.category\_id

and p.list\_price between amount-ranges and amount+ranges);

exception

when others then

return null;

end;

return result;

END;

**Workout-8 (Oracle Sample Data, Oracle database)**

**Problem Statement**: Write a procedure to populate a table having columns customer\_id, customer\_name and purchase\_posibilities varchar2(4000).

**Hint**: Use tables – customers,orders, order\_items,products

**Rule to fillup purchase\_posibilities:** find all the purchases per customer. Find average of standard\_cost and average of list\_price. Send average of standard\_cost as amount and difference of averages between standard\_cost and list\_price as ranges in the above function. The output of the function will be the purchase\_posibilities.

CREATE OR REPLACE PROCEDURE CUSTOMER\_POSSIBILITIES AS

cursor c1 is

select customer\_id,name

,get\_assesment(standard\_cost,abs(standard\_cost-list\_price)) possibility

from (

select customers.customer\_id,customers.name

,avg(products.standard\_cost) standard\_cost

,avg(products.list\_price) list\_price

from

customers,orders, order\_items,products

where

customers.customer\_id=orders.customer\_id

and orders.order\_id=order\_items.order\_id

and order\_items.product\_id=products.product\_id

group by customers.customer\_id,customers.name

);

BEGIN

begin

delete from customer\_purchase\_possibility;

for r1 in c1 loop

insert into customer\_purchase\_possibility(customer\_id, customer\_name , purchase\_posibilities)

values (r1.customer\_id,r1.name,r1.possibility);

end loop;

exception

when others then

rollback;

return;

end;

commit;

END CUSTOMER\_POSSIBILITIES;

**Workout-9 Design Database**

Consider a Medicine shop management system. System must have the below capabilities:

1. Keep medicine information.
2. Keep seller information.
3. Keep purchase information by which it can increase stock.
4. Sell medicine and keep information for sell along with seller and purchaser.
5. Able to provide current stock of all medicines.
6. Can produce purchase report for a specified period time.
7. Able to find Moving Averages.

Try to design the required tables along with columns which you may find best suitable from your practical experience.

Try to define table constraints (PK, FK, UNIQUE, CHECK) which may be imposed to check bad data.

Try to guess what indexes are required to build for faster queries.

Try to define some function for getting current stock.

Try to define procedure for outputs (output tables may also be defined).

**SELLER\_MASTER**

SELLER\_ID NUMBER PRIMARY KEY

SELLER\_NAME VARCJAR2(30)

SELLER\_ADDRESS VARCHAR2(1000)

SELLER\_PHONE NUMBER

**MEDICINE\_MASTER**

MED\_ID NUMBER PRIMARY KEY

MED\_NAME VARCHAR2(500)

CURRENT\_STOCK NUMBER CHECK (CURRENT\_STOCK>=0)

MED\_POWER VARCHAR2(20)

SELLER\_ID NUMBER REFERENCES SELLER\_MASTER(SELLER\_ID)

CREATE INDEX MEDICINE\_MASTER\_IDX1 ON MEDICINE\_MASTER (MED\_NAME);

CREATE INDEX MEDICINE\_MASTER\_IDX2 ON MEDICINE\_MASTER (SELLER\_ID);

**MED\_PURCHASE**

PURCHASE\_ID NUMBER PRIMARY\_KEY

MED\_ID NUMBER REFERENCES MEDICINE\_MASTER

SELLER\_ID NUMBER REFERENCES SELLER\_MASTER

PURCHASE\_AMOUNT NUMBER CHECK PURCHASE\_AMOUNT>0

PURCHASE\_QTY NUMBER CHECK PURCHASE\_QTY>0

PURCHASE\_DATE DATE

BATCH\_NUMBER VARCHAR2(100)

EXPIRY\_DATE DATE

MANUFACTURER\_NAME VARCHAR2(200)

CREATE INDEX MED\_PURCHASE\_IDX1 ON MED\_PURCHASE(MED\_ID);

CREATE INDEX MED\_PURCHASE\_IDX2 ON MED\_PURCHASE(PURCHASE\_DATE);

CREATE INDEX MED\_PURCHASE\_IDX3 ON MED\_PURCHASE(BATCH\_NUMBER);

CREATE INDEX MED\_PURCHASE\_IDX4 ON MED\_PURCHASE(EXPIRY\_DATE);

CREATE INDEX MED\_PURCHASE\_IDX5 ON MED\_PURCHASE(COMPANY\_NAME);

**MED\_SELL**

SELL\_ID NUMBER PRIMARY\_KEY

VOUCHER\_NUMBER VARCHAR2(200) UNIQUE

MED\_ID NUMBER REFERENCES MEDICINE\_MASTER

SELLER\_ID NUMBER REFERENCES SELLER\_MASTER

SELL\_AMOUNT NUMBER CHECK PURCHASE\_AMOUNT>0

SELL\_QTY NUMBER CHECK PURCHASE\_QTY>0

SELL\_DATE DATE

BUYER\_NAME VARCHAR2(30)

BUYER\_DOCTOR\_NAME VARCHAR2(30)

BUYER\_MOBILE\_NO NUMBER

BUYER\_ADDRESS VARCHAR2(2000)

CREATE INDEX MED\_SELL\_IDX1 ON MED\_SELL(BUYER\_NAME, BUYER\_MOBILE\_NO);

CREATE INDEX MED\_SELL\_IDX2 ON MED\_SELL(MED\_ID);

CREATE INDEX MED\_SELL\_IDX3 ON MED\_SELL(SELLER\_ID);

CREATE INDEX MED\_SELL\_IDX4 ON MED\_SELL(SELL\_DATE);

CREATE OR REPLACE PROCEDURE UPDATE\_STOCK(SELLID IN NUMBER) AS

BEGIN

UPDATE MEDICINE\_MASTER

SET CURRENT\_STOCK:=CURRENT\_STOCK-(SELECT SELL\_QTY FROM MED\_SELL WHERE SELL\_ID=SELLID)

WHERE MED\_ID=(SELECT MED\_ID FROM MED\_SELL WHERE SELL\_ID=SELLID);

COMMIT;

END;

Some queries:

* select all medicine sold by any seller for a period of time.
* select all medicine sold in a period of time.
* select whine is sold very frequently and sold maximum in a period of time.
* find voucher by buyer mobile number or name.
* find which medicines are expired and about to expire in some days.
* find which seller sold maximum medicines in a period of time.
* find profit in a period of time.