Ecommerce project

by Vedant Singhvi

ECOMMERCE DATABASE IN ORACLE

Problem Statement:

Ecommerce is on the rise with new features and new companies implementing this model of business frequently. With this, the rise of having an efficient database system, to improve results, analysis and speed of transactions is direly needed. In addition to this, highly secure systems need to be made, thanks to the extensive amount of consumer data available online on the websites

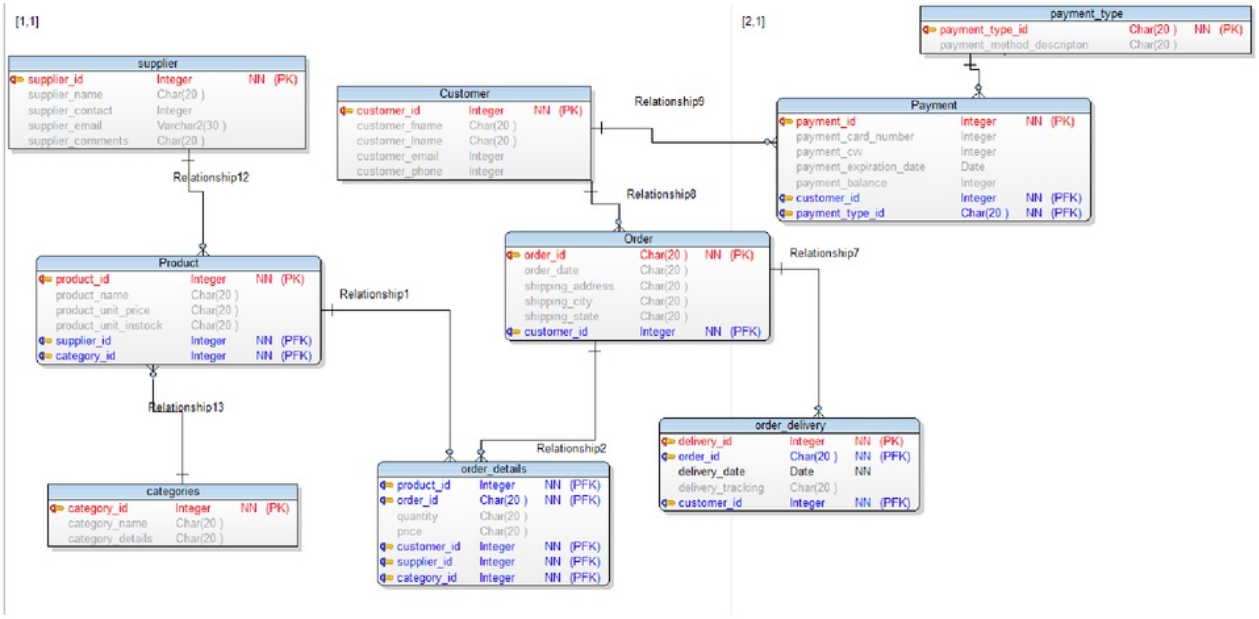
Introduction:

I have created a basic ecommerce database consisting of the following tables:

- Customer: the entity who buys the product
- Product: Product the customer buys
- Orders: order is the thing that a customer place
- Order Details: A bridge table between order and product, can be understood as a cart
- Payment: Consists of the card and cash details of the consumer
- Payment type: type of payment, can be either cash, card or balance
- Category: product category of products
- Supplier: Entity who supplied products
- Order shipping: consists of the shipping information for product

The ‘customer’ adds a ‘product’ to the ‘orders’ into the ‘order_detail’. Shipping information is generated in the ‘shipping_delivery’. Customer pays using ‘payment’, and ‘payment’ has a ‘payment_type’ as well. The ‘product’ has a ‘supplier’ and ‘categories’.

ER MODEL



WHAT HAVE I IMPLEMENTED?

- STORED PROCEDURES
- TRIGGERS
- VIEWS
- FUNCTIONS
- JOINS
- PACKAGE
- USER CREATION
- CURSOR
- EXCEPTION HANDLING

-STORED PROCEDURES

Developed stored procedures, suitable for the database. Used cursor in stored procedure, to retrieve multiple output columns.

1. Stored procedure to change the price of a product by a multiplicative variable when given the product id and the increase amount.
2. Stored procedure to view the delivery information of all the products at once.
3. Stored procedure to view any customer information when given customer id as the input.

```
CREATE OR REPLACE PROCEDURE product_price( pro_id NUMBER, price_raise NUMBER)
 IS
 BEGIN
   UPDATE product SET product_unit_price = product_unit_price * price_raise WHERE product_id = pro_id;
 EXCEPTION
    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('No product exists');
 END;
 /

 exec product_price(1, 1.2);
 select product_unit_price, product_id from product where product_id=1;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 1 in 0 seconds

	PRODUCT_UNIT_PRICE	PRODUCT_ID
1	60	1

```
select product_unit_price, product_id from product where product_id=1;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 1 in 0 seconds

	PRODUCT_UNIT_PRICE	PRODUCT_ID
1	50	1

```
create or replace procedure printDelivery is
      cursor c_emp is (select delivery_date, delivery_tracking, od.customer_id, shipping_address, shipping_city, shipping_state from ord
      r_emp c_emp%ROWTYPE;
begin
      open c_emp;
      loop
        fetch c_emp into r_emp;
        exit when c_emp%NOTFOUND;
        DBMS_OUTPUT.put_line(r_emp.delivery_tracking);
      end loop;
      close c_emp;
end;
/

exec printDelivery();
```

Script Output × Query Result ×

Task completed in 0.031 seconds

```
23224420
23223450
29093450
29093410
```

```
CREATE OR REPLACE PROCEDURE print_contact(
    p_customer_id NUMBER )
IS
  r_customer customer%ROWTYPE;
BEGIN
  SELECT * INTO
    r_customer
  FROM
    customer c
  WHERE
    customer_id = p_customer_id;

  DBMS_OUTPUT.PUT_LINE(r_customer.customer_fname || '  ' ||
  r_customer.customer_phone);
 EXCEPTION
WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE( SQLERRM );
END;
exec print_contact(1)
```

Script Output × Query Result ×

Task completed in 0.031 seconds

```
customera              123456
```

TRIGGER:
Implemented a useful trigger, wherein the product stock decreases if the product is added to the order_details cart by the customer.

```
TRIGGER

create or replace trigger stock_change
after insert ON order_details
FOR each row
begin
update product
set product.product_unit_instock= product.product_unit_instock-:New.quantity
where product.product_id=:New.product_id;
end;
```

FUNCTION:
Implemented a function to classify a product in-stock quantity as 'extra', 'right amount' or 'more stock needed'. Used cursor and if-else condition to return the value in the form of a statement.

```
create or replace function instock1(product_id in NUMBER) return varchar2
is
    tmp_stock number;
    stocklevel varchar2(60);

    cursor c1 is
    select product_unit_instock
    from product
    where product_id=product_id;
begin
    open c1;
    fetch c1 into tmp_stock;
    close c1;

    if tmp_stock between 20 and 200 then
        stocklevel:='in stock';
    elsif tmp_stock between 100 and 1000 then
            stocklevel:='over stock';
    elsif tmp_stock between 0 and 10 then
        stocklevel:='please restock';
    end if;
end;
/
```

PACKAGE:
Used 2 stored procedures, to combine them into one single package called pkg_ecommerce. Performed troubleshooting to make sure both the procedures execute properly together or independently.

```
create or replace PACKAGE pkg_ecommerce IS
  PROCEDURE prnt_emps(p_customer_id NUMBER );
  PROCEDURE print_dicount( pro_id NUMBER, price_raise NUMBER);
END pkg_ecommerce;
```

```
create or replace PACKAGE BODY pkg_ecommerce IS
  PROCEDURE prnt_emps(p_customer_id NUMBER ) IS
  r_customer customer%ROWTYPE;
  BEGIN
 -- get contact based on customer id
  SELECT...;

 -- print out contact's information
  DBMS_OUTPUT.PUT_LINE(r_customer.customer_fname || ' ' ||
  r_customer.customer_phone);

  EXCEPTION
  WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE( SQLERRM );
  END;

PROCEDURE print_dicount( pro_id NUMBER, price_raise NUMBER) IS
BEGIN
  UPDATE product SET product_unit_price = product_unit_price * price_raise WHERE product_id = pro_id;
 EXCEPTION
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('No product exists');
END;

END pkg_ecommerce;
```

CURSOR & exception handling:

Used cursor to reiterate over queries and return dbms output queries with statement. Also used ref-cursor to get customer input and present the output. Also implemented some basic exception handling in the ref cursor.

- CURSOR FOR LOOP

```
set serveroutput on;
declare
begin
    for customer_sales in (select product.product_name, sum(product.product_unit_instock) instock, sum(product_unit_price) stock from pro
    loop
        dbms_output.put_line('The product of ' ||customer_sales.product_name|| ' has ' || customer_sales.instock || 'quantities in stock
    end loop;
end;
/
```

```
Script Output ×   Query Result ×
Task completed in 0.047 seconds
The product of phone            has 40quantities in stock and each costs 450
The product of earphones        has 100quantities in stock and each costs 55
The product of chair            has 200quantities in stock and each costs 20
The product of mat              has 190quantities in stock and each costs 60
The product of mattress         has 200quantities in stock and each costs 50

PL/SQL procedure successfully completed.
```

- REF-CURSOR

```
set echo off
set serveroutput on
set verify off
set define '&'

prompt
prompt 'What data would you like to see:'
accept tab prompt '(B)ookings or (C)ustomers : '
prompt

declare
    type refcur_t is ref cursor;
    refcur refcur_t;

    selection varchar2(1) := upper(substr('&tab',1,1));
    sample number;
begin
    if (selection = 'B') then
        open refcur for
            select customer_id
            from customer
            where rownum < 1

            order by customer_id;
        dbms output.put line('Customer data');
```

```
            order by customer_id;
        dbms_output.put_line('Customer data');

    elsIf (selection = 'C') then
        open refcur for
            select product_id, product_unit_instock, product_unit_price
                from product
                where rownum < 1
                order by product_id;
        dbms_output.put_line('Product Data');
    ELSE
        dbms_output.put_line('Please Enter ''B'' or ''C''');
        return;
    end if;

    loop
        fetch refcur into sample;
        EXIT WHEN refcur%NOTFOUND ;
        dbms_output.put_line('nothing found');

    end loop;
    close refcur;

End;
```

VIEWS:

Created an object view and materialized view to display insights about data.

```
--create type customer_addresses is object(
--    address_id number,
  --  address1 varchar2(50),
   --city varchar2(50),
 --    country varchar2(30)
--);
--create view object_customer_details of customer_addresses
  --  with object oid (address_id) as
    ---     select customer_phone,
       -- customer_lname,
        --customer_email,
        --customer_fname
   --from customer;
--/
SELECT * FROM object_customer_details;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 5 in 0 seconds

	ADDRESS_ID	ADDRESS1	CITY	COUNTRY
1	123456	a	customera@a.com	customera
2	234455	b	customerb@b.com	customerb
3	987654	c	customerc@c.com	customerc

Types
Sequences
Materialized Views
FULL_PRICE
MENU
Materialized View Logs
Synonyms
Public Synonyms
Database Links
Public Database Links
Directories
DEFAULTDIR

Reports
All Reports
Analytic View Reports
Data Dictionary Reports
Data Modeler Reports

Worksheet | Query Builder

```
select * from full_price;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 5 in 0 seconds

	CUSTOMER_ID	TOTAL_EXPENSE	PAYMENT_METHOD_DESCRIPTON
1	4	180	card
2	1	110	card
3	2	525	card
4	5	70	balance
5	3	160	balance

-TRIED IMPLEMETING EXTERNAL TABLE. KEPT GETTING THE 'ODCIEXITTABLEOPEN' callout, data cartridges error, along with directory not found error. Faced the issue even after elevating the privileges of the user.

ANALYSIS:

Analyzed the improvement of a materialized view.

Non materialized query took 0.227 sec, while the materialized query to 0.001 sec. Huge improvement.

Worksheet | Query Builder

```
select o.customer_id, sum(quantity*price) as total_expense,  pt.payment_method_descripton
from order_details o
join payment p on o.customer_id=p.customer_id
inner join payment_type pt
on pt.payment_type_id=p.payment_type_id
group by o.customer_id,  pt.payment_method_descripton;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.227 seconds

	CUSTOMER_ID	TOTAL_EXPENSE	PAYMENT_METHOD_DESCRIPTON
1	4	180	card
2	1	350	card
3	2	525	card
4	5	70	balance
5	3	160	balance

```
select * from full_price;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.001 seconds

	CUSTOMER_ID	TOTAL_EXPENSE	PAYMENT_METHOD_DESCRIPTON
1	4	180	card
2	1	110	card
3	2	525	card
4	5	70	balance
5	3	160	balance

CONCLUSION:

Thus, as you can see, I have implemented the majority of the PLSQL and ORACLE features, and we can confidently say that it a small building block towards an exhaustive fe-commerce database.

APPENDIX:

- **Stored procedure 1:**

```
CREATE OR REPLACE PROCEDURE product_price( pro_id NUMBER, price_raise NUMBER)

IS

BEGIN

  UPDATE product SET product_unit_price = product_unit_price * price_raise WHERE product_id
= pro_id;

 EXCEPTION

   WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('No product exists');

END;

/
```

- **Stored procedure 2:**

```
create or replace procedure p_printEmps is

    cursor c_emp is (select delivery_date, delivery_tracking, od.customer_id, shipping_address,
shipping_city, shipping_state from order_delivery od join orders o on od.orders_id=o.orders_id);

    r_emp c_emp%ROWTYPE;

begin

    open c_emp;

    loop

     fetch c_emp into r_emp;

     exit when c_emp%NOTFOUND;

     DBMS_OUTPUT.put_line(r_emp.delivery_tracking);

    end loop;

     close c_emp;

end;

/
```

- **Stored Procedure 3:**

```
CREATE OR REPLACE PROCEDURE print_contact(
   p_customer_id NUMBER )
IS
  r_customer customer%ROWTYPE;
BEGIN
  -- get contact based on customer id
  SELECT
    *
  INTO
    r_customer
  FROM
    customer c
  WHERE
    customer_id = p_customer_id;

  -- print out contact's information
  DBMS_OUTPUT.PUT_LINE(r_customer.customer_fname || ' ' ||
  r_customer.customer_phone);

EXCEPTION
WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE( SQLERRM );
END;
```

- **For loop cursor:**

```
set serveroutput on;
declare
begin
   for customer_sales in (select product.product_name,
sum(product.product_unit_instock) instock, sum(product_unit_price) stock from product
group by product.product_name)
   loop
      dbms_output.put_line('The product of ' ||customer_sales.product_name|| ' has ' ||
customer_sales.instock || 'quantities in stock and each costs ' || customer_sales.stock);
   end loop;
end;
/
```

- **REF Cursor:**

```
set echo off
set serveroutput on
set verify off
set define '&'
prompt
prompt 'What data would you like to see:'
accept tab prompt '(B)ookings or (C)ustomers : '
prompt
declare
        type refcur_t is ref cursor;
        refcur refcur_t;

        selection varchar2(1) := upper(substr('&tab',1,1));
   sample number;
begin
        if (selection = 'B') then
                open refcur for
                        select customer_id
                        from customer
                        where rownum < 1

                        order by customer_id;
                dbms_output.put_line('Customer data');
        elsIf (selection = 'C') then
                open refcur for
                        select product_id, product_unit_instock, product_unit_price
                                from product
                                where rownum < 1
                                order by product_id;
                dbms_output.put_line('Product Data');
        ELSE
```

```
                dbms_output.put_line('Please Enter "B" or "C"');
                return;
        end if;
        loop
                fetch refcur into sample;
                EXIT WHEN refcur%NOTFOUND ;
                dbms_output.put_line('nothing found');

        end loop;
        close refcur;
End;
```

- **Trigger:**

```
create or replace trigger stock_change

after insert ON order_details

FOR each row

begin

update product

set product.product_unit_instock= product.product_unit_instock-:New.quantity

where product.product_id=:New.product_id;

end;
```

- **Package**:

```
CREATE OR REPLACE PACKAGE pkg_ecommerce IS
  PROCEDURE prnt_emps(p_customer_id NUMBER );
  PROCEDURE print_dicount( pro_id NUMBER, price_raise NUMBER);
END pkg_ecommerce;

--Package Body
CREATE OR REPLACE PACKAGE BODY pkg_ecommerce IS
  --Function Implimentation
PROCEDURE print_contact(p_customer_id NUMBER )IS
  r_customer customer%ROWTYPE;
BEGIN
  -- get contact based on customer id
  SELECT
    *
  INTO
    r_customer
  FROM
    customer c
```

```
    WHERE
      customer_id = p_customer_id;

    -- print out contact's information
    DBMS_OUTPUT.PUT_LINE(r_customer.customer_fname || ' ' ||
    r_customer.customer_phone);

  EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE( SQLERRM );
  END;

  PROCEDURE product_price( pro_id NUMBER, price_raise NUMBER) IS
  BEGIN
    UPDATE product SET product_unit_price = product_unit_price * price_raise WHERE
  product_id = pro_id;
   EXCEPTION
     WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('No product exists');
  END;

  END pkg_ecommerce;
```

- **Function:**

```
create or replace function instock1(product_id in NUMBER) return varchar2

is

   tmp_stock number;

   stocklevel varchar2(60);

   cursor c1 is

   select product_unit_instock

   from product

   where product_id=product_id;

begin

   open c1;

   fetch c1 into tmp_stock;

   close c1;

   if tmp_stock between 20 and 200 then

      stocklevel:='in stock';
```

```
  elsif tmp_stock between 100 and 1000 then
      stocklevel:='over stock';
  elsif tmp_stock between 0 and 10 then
     stocklevel:='please restock';
  end if;
end;
/
```

- **OBJECT VIEW:**

```
create type customer_addresses is object(
   address_id number,
   address1 varchar2(50),
   city varchar2(50),
   country varchar2(30)
);
create view object_customer_details of customer_addresses
   with object oid (address_id) as
      select customer_phone,
      customer_lname,
      customer_email,
      customer_fname
   from customer;
/
```

- **MATERIALIZED VIEW:**

```
CREATE MATERIALIZED VIEW full_price
BUILD IMMEDIATE
REFRESH ON COMMIT
AS
SELECT o.customer_id, sum(quantity*price) as total_expense,
pt.payment_method_descripton
from order_details o
join payment p on o.customer_id=p.customer_id
inner join payment_type pt
on pt.payment_type_id=p.payment_type_id
group by o.customer_id,  pt.payment_method_descripton;
```

- **EXTERNAL TABLE**

```
create or replace directory defaultdir as 'C:/Users/vedants/Desktop';
drop table contract;
create table contract (
contract_id number(4),
```

```
contract_type varchar(20)
)
organization external (
   type oracle_loader
   default directory defaultdir
   access parameters(
   fields terminated by ',')
   location('contract.txt')
)
```