Retail Business Management System

CS 532 DAtabase Systems – Project 2

Ramani Manasa Devarabhatla

Akash Munji

Venkata Sri sai surya vinay upadhyayula

Spring 2023

Contents

[Sequence 2](#_Toc133240888)

[Sequence used in this project 2](#_Toc133240889)

[Package 3](#_Toc133240890)

[Package used in this project 3](#_Toc133240891)

[Procedure and Functions 3](#_Toc133240892)

[Functions 4](#_Toc133240893)

[Procedures 4](#_Toc133240894)

[Triggers 5](#_Toc133240895)

[Triggers used in this project 5](#_Toc133240896)

[new\_employee 5](#_Toc133240897)

[new\_purchase 5](#_Toc133240898)

[Customers and products triggers 7](#_Toc133240899)

[Java – JDBC Interface 8](#_Toc133240900)

[Connection to SQL Plus server 8](#_Toc133240901)

[Inputs from the user 8](#_Toc133240902)

[Querying 8](#_Toc133240903)

[Result Display 9](#_Toc133240904)

[Exception Handling 9](#_Toc133240905)

[Team Report 9](#_Toc133240906)

[a. Meetings 9](#_Toc133240907)

[b. Plans 10](#_Toc133240908)

[c. Responsibilities 10](#_Toc133240909)

[d. Self-Assessment 10](#_Toc133240910)

[Project Execution Procedure 11](#_Toc133240911)

[Technologies used 12](#_Toc133240912)

**DB Project Documentation**

“We have done this assignment completely on our own. We have  
not copied it, nor have we given our solution to anyone else. We understand that if we are  
involved in plagiarism or cheating we will have to sign an official form that we have  
cheated and that this form will be stored in our official university records. We also  
understand that we will receive a grade of 0 for the involved assignment and our grades will  
be reduced by one level (e.g., from A to A- or from B+ to B) for our first offense, and that  
we will receive a grade of “F” for the course for any additional offense of any kind.”

This is a Java JDBC and PL/SQL project that implements Retail Business Management System application. This application provides an efficient solution to manage all the retail business operations.

The project’s key features include the ability to display all the information on employees, customers, products, purchases, purchase discounts. It also includes functions that generates reports on monthly sale activities made my induvial employee, add, and manage new employees, make new purchases, and update the customers information every time a customer makes a purchase. It also has a dedicated log table to track the changes made for each transaction in the retail business.

Following are the tables and database objects used in this project:

• Employees(eid, name, telephone#, email)

• Customers(cid, first\_name, last\_name, phone#, visits\_made, last\_visit\_date)

• Products(pid, name, qoh, qoh\_threshold, orig\_price, discnt\_category)

• Prod\_Discnt(discnt\_category, discnt\_rate)

• Purchases(pur#, eid, pid, cid, pur\_time, quantity, unit\_price, payment, saving)

• Logs(log#, user\_name, operation, op\_time, table\_name, tuple\_pkey)

Sequence

A sequence is a database object that generates a sequence of unique values when called. This property is mainly used to create primary key values in a table.

## Sequence used in this project

create sequence pur\_seq# start with 10001 increment by 1

create sequence logs\_seq# start with 1001 increment by 1

* "pur\_seq#" is a pl/sql sequence that generates the purchase id values (primary key) for the purchase table, which starts with 10001 and is incremented by 1 every time pur\_seq#.nextval is called.
* “logs\_seq#” is pl/sql sequence that generates the log id values (primary key) for the logs table, which starts with 1001 and is incremented by 1 with each call.

insert into purchases values (pur\_seq#.nextval, 'e01', 'p002', 'c001', to\_date('12-AUG-2022', 'DD-MON-YYYY'), 1, 211.65, 211.65, 37.35);

* Each time the NEXTVAL function is called in an SQL insert query to insert tuples into the table, the sequence generates unique ID values for every new row.
* The log values are automatically generated using triggers.
* The “pur\_seq#”, "logs\_seq#" sequence generate purchase Ids and log Ids respectively.

Package

In PL/SQL, a package is a named collection of related functions, procedures, variables, and other program objects that can be stored and reused in multiple places within a database.

-> A package is divided into two parts – package specification and package body.

## Package used in this project

create or replace package prj2 as

* This code either creates a new pl/sql package or replace an existing package called ‘prj2’.

type ref\_cursor is ref cursor;

* Here, the package specification contains a user-defined type “ref\_cursor” within it, which is a cursor data type that can be used to hold a reference to a cursor variable.
* Package body consists of all the procedures and functions required to implement the project’s functionality.

Procedure and Functions

Stored procedures and functions in PL/SQL are stored in compiled form in the database. Once the procedure or function is created and compiled, the resulting code is stored in the database, ready to be executed whenever it is called.

**The package body contains the following procedures and functions:**

function display\_employees

    return ref\_cursor;

    function display\_customers

    return ref\_cursor;

    function display\_products

    return ref\_cursor;

    function display\_prd\_discnts

    return ref\_cursor;

    function display\_purchases

    return ref\_cursor;

    function display\_logs

    return ref\_cursor;

    procedure add\_employee

    (e\_id in employees.eid%type,

     e\_name in employees.name%type,

     e\_telephone# in employees.telephone#%type,

     e\_email in employees.email%type);

    procedure add\_purchase

    (

        e\_id in purchases.eid%type,

        p\_id in purchases.pid%type,

        c\_id in purchases.cid%type,

        pur\_qty in purchases.quantity%type,

        unit\_price in purchases.unit\_price%type

    );

Functions

* **display\_employee**: This function returns a cursor containing all the employee information.
* **display\_customers**: This function returns a cursor containing all the customer information.
* **display\_products:** This function returns a cursor containing all the products information.
* **display\_prd\_discnts:** This function returns a cursor containing all the product discount information.
* **display\_purchase:** This function returns a cursor containing all the purchases information.
* **display\_logs:** This function returns a cursor containing all the logs information.
* **Monthly\_sale\_activities:** Function named monthly\_sale\_activities is a function that takes an employee ID as input and returns a cursor of monthly sales activities for that employee. The cursor is declared as ref\_cursor.

-> Inside the function, the dbms\_output.enable() statement enables output messages to be displayed in the console.

-> The open statement opens the cursor sale\_cursor and runs a SELECT query that joins the employees and purchases tables on the eid column and retrieves monthly sales activities for the employee with the given ID.

-> The columns retrieved are the employee ID and name, the month and year of the purchase, the count of sales, the total quantity sold, and the total sales amount. The group by clause groups the results by the employee ID, name, month, and year.

-> After that, the dbms\_output.put\_line() statement displays a message in the console indicating that the function is returning monthly sale activities.

-> Finally, the function returns the cursor sale\_cursor containing the monthly sales activities for the employee with the given ID.

Procedures

* **add\_employee**: This procedure adds a new row to the employee table with the given employee id, employee name, employee phone and employee email address.
* **add\_purchase:** This procedure adds a new purchase row into the purchase table with the given employee ID, product ID, customer ID, quantity, and unit price.

-> Local variables org\_price and qoh are initialized and their information is retrieved from the products tables with product id using the select statement.

-> if the purchase quantity is greater than that of the quantity on hand, then it returns insufficient quantity in stock, otherwise it inserts a new row into the purchase table with the given employee ID, product ID, customer ID, quantity, and unit price values.

-> It also calculates the total payment and savings based on the original price and inserts these values into the corresponding columns of the new row.

**Note:** **%TYPE** attribute is used to initialize/ declare a variable to have the same data type as the given column in the given table.

Triggers

These are blocks of code that are automatically executed in response to any event occurring in a database. There are two types of triggers, row-level, and statement-level. Row-level is executed every time a row is affected. Statement-level is executed for every SQL statement

## Triggers used in this project

### new\_employee

set serveroutput on

create or replace trigger new\_employee

after insert on employees

for each row

begin

  insert into logs values(log#.nextval,USER,'INSERT',SYSDATE,'employees',:new.eid);

    dbms\_output.put\_line('done.');

end;

* This code creates a new trigger called ‘new\_employee’ or replace one if it already exists. ‘After insert on’ command fires a trigger every time a new row is inserted into the employee’s table.

-> The primary key log# of the logs table is filled using a sequence(log# as discussed in ) that generates a unique ID each time a new row is inserted into the log table.

-> Inside the trigger's body, an insert statement is executed to insert a new row into the logs table with values for the "log#" (generated by the sequence), "USER" (the username of the user who executed the insert statement), "INSERT" (the type of operation performed), "SYSDATE" (the current system date and time), "employees" (the name of the table on which the insert statement was executed), and ":new.eid" (the value of the "eid" column for the newly inserted row).

### new\_purchase

create or replace trigger new\_purchase

after insert on purchases

for each row

declare

  qoh products.qoh%type;

  qoh\_threshold products.qoh\_threshold%type;

  p\_id products.pid%type;

  visits\_made customers.visits\_made%type;

  last\_visit\_date customers.last\_visit\_date%type;

begin

  dbms\_output.enable();

  insert into logs values(log#.nextval,USER,'INSERT',SYSDATE,'purchases',:new.pur#);

  dbms\_output.put\_line('done.');

  select products.qoh, products.qoh\_threshold, products.pid into qoh, qoh\_threshold, p\_id

  from products where pid = :new.pid;

  if ((qoh - :new.quantity)< qoh\_threshold) then

    dbms\_output.put\_line('The current qoh of the product is below the required threshold and new supply is required.');

    update products

    SET qoh = qoh\_threshold + 20

    where products.pid = p\_id;

  else

    update products

    SET qoh = qoh - :new.quantity

    where products.pid = p\_id;

  end if;

  select customers.visits\_made, customers.last\_visit\_date into visits\_made, last\_visit\_date

  from customers where cid = :new.cid;

  update customers

  SET visits\_made = visits\_made + 1

  where customers.cid = :new.cid;

  if(last\_visit\_date != :new.pur\_time) then

    update customers

    SET last\_visit\_date = :new.pur\_time

    where customers.cid = :new.cid;

  end if;

  dbms\_output.put\_line('New qoh for this product is updated');

  end;

/

show errors

* This code creates an Oracle trigger named "new\_purchase" that executes after an insert on the "purchases" table for each row. The trigger updates the "qoh" (quantity on hand) and "visits\_made" fields of the "products" and "customers" tables respectively.
* The trigger first inserts a row into a "logs" table with information about the user, operation type, date, table name, and the primary key of the inserted row.
* Then, it selects the "qoh" and "qoh\_threshold" fields from the "products" table for the product ID and stores them in local variables. If the quantity on hand is less than the qoh\_threshold after the purchase, it prints an output saying, “The current qoh of the product is below the required threshold and new supply is required.” and updates the "qoh" of the "products" table with a value that is qoh threshold plus 20. Otherwise, it subtracts the quantity of the purchased product from the "qoh" field.
* The trigger then selects the "visits\_made" and "last\_visit\_date" fields from the "customers" table for the given customer ID and stores them in variables. Every time the customers makes a purchase, it increments the "visits\_made" field by 1 and updates it in the "customers" table. If the last visit date is not the same as the purchase date, it updates the "last\_visit\_date" field in the "customers" table with the purchase date.
* Finally, it prints an output whenever a new qoh value of the product is updated in the products table.

### Customers and products triggers

create or replace trigger cust\_last\_visit\_date

after update of last\_visit\_date on customers

for each row

begin

  insert into logs values(log#.nextval,USER,'update',SYSDATE,'customers',:new.cid);

end;

/

create or replace trigger cust\_visits\_made

after update of visits\_made on customers

for each row

begin

  insert into logs values(log#.nextval,USER,'update',SYSDATE,'customers',:new.cid);

end;

/

create or replace trigger prd\_qoh

after update of qoh on products

for each row

begin

  insert into logs values(log#.nextval,USER,'update',SYSDATE,'products',:new.pid);

end;

/

show errors

* The above three trigger codes are used to update the logs table each time the last\_visit\_date on the customers table, visits\_made field on the customers table and the qoh field on the products table are updated executing an insert statement by inserting a new row into the logs table with values for the "log#" (generated by the sequence), "USER" (the username of the user who executed the insert statement), "INSERT" (the type of operation performed), "SYSDATE" (the current date and time), "customers/products" (the name of the table on which the insert statement was executed), and ":new.cid/ new:pid" (the value of the "cid/pid" column for the newly inserted row).

# Java – JDBC Interface

The RBMS project is made interactive to the user by a choice-based menu driven java/ jdbc interface, hiding actual pl/sql implementation thereby achieving abstraction. The entire functionality of the project can be tested using the RBMSInterface.java class. The following is an overview of java implementation

## Connection to SQL Plus server

The connection to harveyv sqlplus server is done using the oracle jdbc connection pool provided by java. Then the user is prompted for his username and password. Upon successful authentication connection is established between java application and sql plus.

Text

Description automatically generated

## Inputs from the user

User inputs and read by the Scanner class provided by java.util package.

Text

Description automatically generated

## Querying

Query to procedures/ triggers is done using PreparedStatement and QueryStatement interfaces and their corresponding methods provided by jdbc.

Text

Description automatically generated

## Result Display

The query result returned by sql is retrieved using ResultSet of jdbc and displayed to the user.

Text

Description automatically generated

## Exception Handling

Exceptions while executing SQL or java code is handled by java exception handling mechanisms using “try catch”. Exception in each method is thrown using throws Exception/SQLException and is caught in the main method. Based on the exception error code, a utility function “error\_handler” displays custom error messages to the user along with the SQL error messages.

Text

Description automatically generated

# Team Report

## a. Meetings

## 

|  |  |
| --- | --- |
| Meeting Date | Discussion |
| 04/02/2023 | Project Requirements read out |
| 04/04/2023 | -> Task distribution to each teammate  -> Key challenges identification |
| 04/06/2023 | -> Solving any issues while setting up the environment |
| 04/08/2023 | -> Started working and testing on individual tasks. |
| 04/12/2023 | -> Updated every teammate on the progress of each task assigned.  -> Resolved any issues if persisted. |
| 04/15/2023 | -> Merged each individual task as one single project and tested the entire project  -> Resolved any issues if persisted.  -> Started documenting the project |

b. Plans

Our team devised a plan to ensure that each team member would handle both Java JDBC and PL/SQL tasks as part of the project. We set internal deadlines for each task and encouraged everyone to complete them within the stipulated time. The individual tasks were meant to be discussed amongst the team members only after they were fully functional or in case there were any obstacles that were blocking and needed to be addressed.

## c. Responsibilities

|  |  |  |
| --- | --- | --- |
| Name | Tasks responsible for | Contribution |
| Vinay Upadhyayula | Project questions 1, 2, 4, 6 | -> Sequence  -> Display all tables  -> Add an employee  -> Triggers  -> Java JDBC to SQL interface  -> SQL buffer messages retrieval into JDBC |
| Akash Munji | Project question 5 | -> Make a purchase  -> Purchase triggers  -> Java JDBC to SQL Interface |
| Manasa Devarabhatla | Project question 3, 7, project documentation | -> Monthly sale activities  -> Java, sql exception handling  -> Java JDBC to SQL interface  -> Project initial documentation draft |

d. Self-Assessment

Worked really well together. The team participated proactively.

# Project Execution Procedure

The project is executed using the following java commands:

1. compile: javac -cp /usr/lib/oracle/18.3/client64/lib/ojdbc8.jar RBMSInterface.java

2. execute: java -cp /usr/lib/oracle/18.3/client64/lib/ojdbc8.jar RBMSInterface.java

**The following images depict various scenarios of the project:**

**The menu driven interface:**

Text

Description automatically generated



­­

**Error Scenario 1 (Invalid username/password)**

Text

Description automatically generated

**Error Scenario 2 (Invalid eid to make a purchase)**

Text

Description automatically generated

**No Data Found Scenario**

Text

Description automatically generated

# Technologies used

-> Java, JDBC

-> Oracle PL/SQL