

**IE2042 - Database Management Systems for Security**

Assignment 01: 2020 Regular Intake

Title: *Supermarket Database Management*

Group Members:

|  |  |
| --- | --- |
| IT Number | Name |
| IT19068374 | R M SASINDU SANKALPA |
| IT19026930 | DINUKA RANDIMA W |
|  |  |
| IT19054582 | S M U T SAMARAKOON |

**Contribution for the Project**

|  |  |
| --- | --- |
| IT Number | Contribution |
| IT19068374 | * ER Diagram * Table creation and data insertion * Access privileges |
| IT19026930 | * Relational schema * Partial contribution for web application security question |
|  | * Recovery mechanisms question * Presentation |
| IT19054582 | * Managing security mechanisms views, stored procedures, stored functions, and triggers. * Partial contribution to web application security section * Scenario * Report creation with proofreading |

# Supermarket Database Management System

## Scenario

There is only one store for the supermarket trademark with the owner working as the manager of the supermarket. The store gets the sales items from several suppliers and once the items are received and confirmed with the quantity and quality, the details of the shipment is entered to the database by the stock manager. Each item category has a stock ID and name. Each category is managed by separate stock managers. The customers can shop the items by their own. The sales persons assist customers with shopping such as weighing certain items for example, rice, sugar etc. Once the customers bring the shopped items to the cashier, the cashier enters the item codes, item names, quantity, and the prices to the database and prints the invoice. The customer then pays the invoice by cash, debit or credit cards. The cashier enters the payment details to the database as well. Each invoice and payment has unique ID for each and the dates and times of invoicing and payments are recorded in the database. Customers who are willing to get the promo codes and special promotions can provide their telephone numbers which are also added to the database. This information can only be reviewed and edited by the manager for security purposes. When a new employee is recruited to the supermarket, the Human Resource (HR) officer enters his/her details to the database. There is one HR officer for the supermarket as of now. If more HR officers are recruited, their details should be entered to the database as well. The access rights for the new employees are created and granted by the manager himself. Database functionalities such as views, stored functions, stored procedures, and triggers can only be created by the manager’s login credentials.

The manager is responsible for managing the database and all the privileges such as inserting, updating, deleting in the database is granted to him.

The supermarket store is run by the manager as mentioned above along with the support of a HR officer, Stock manager, Sales persons, and the Cashiers. The supermarket has neither pharmacy nor employees assigned for cleaning purposes.

The HR officer is responsible for managing employee details. He can enter, update or delete the records of the employees.

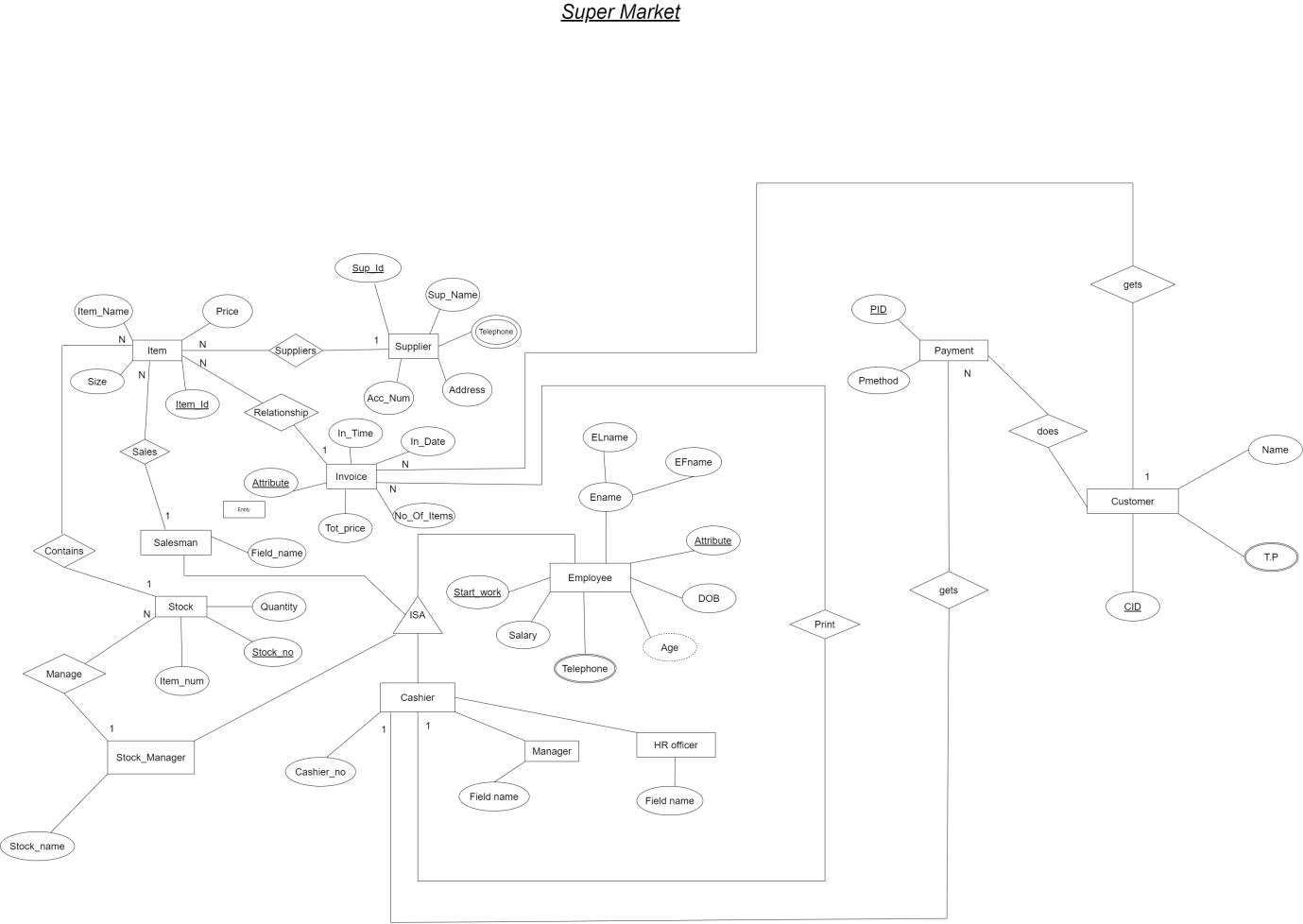
The Stock manager is responsible for managing the stock details. He enters details of the items which are bought from the suppliers with their respective details too.

The sales persons assist the customers to shop in the supermarket while the cashiers insert the invoice details and payment details of the customers to the database.

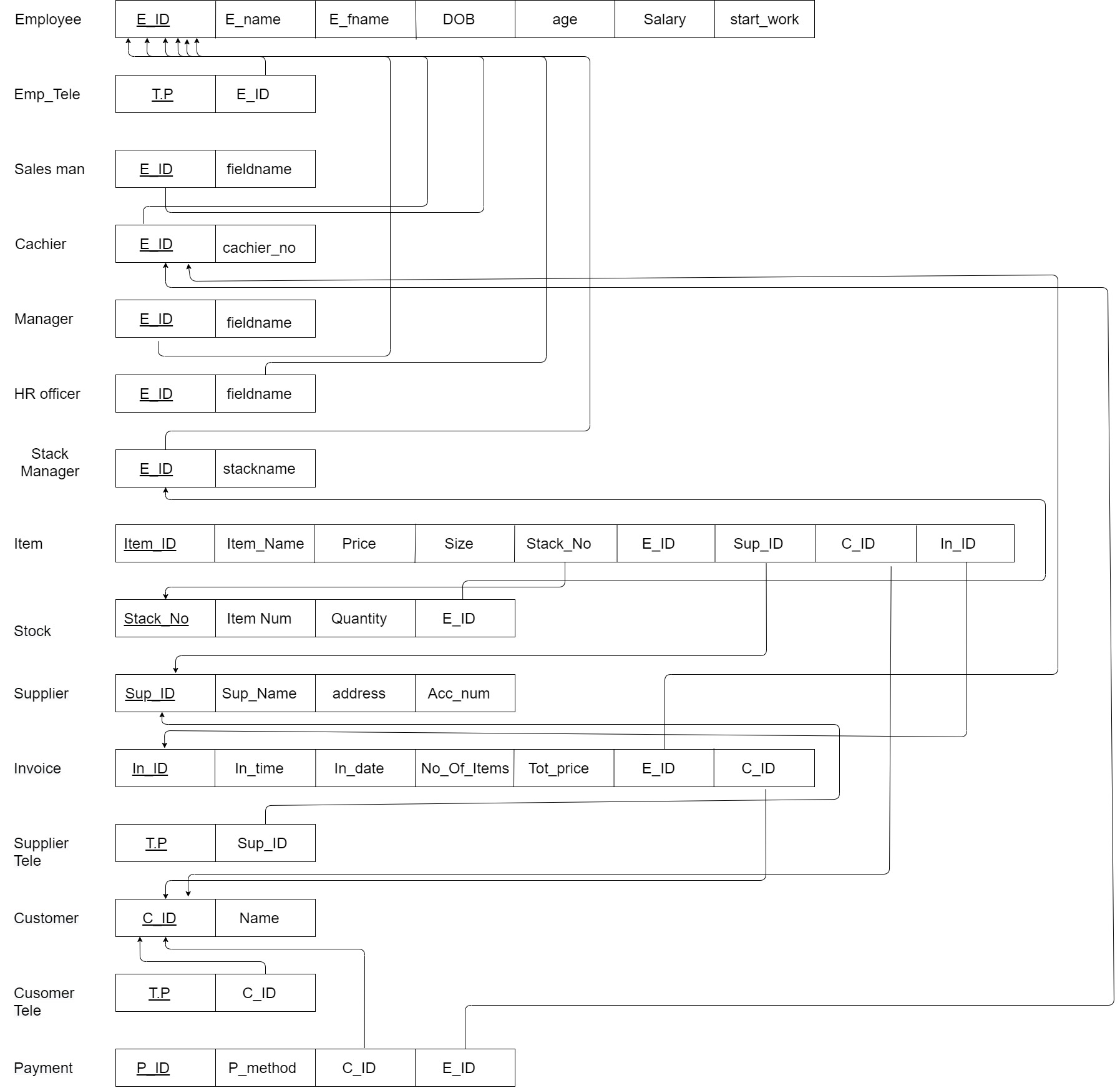
When talking about the access privileges given to these employees, the item table can be viewed by every employee but the editing privileges are not offered for them.

The manager can access and edit all the data stored in the database. HR officer and the Stock manager can access and edit Employee table and Stock table respectively.

## ER diagram

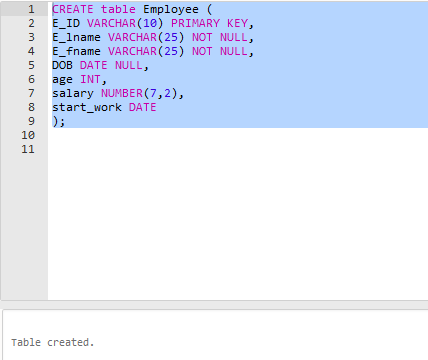


## Relational schema

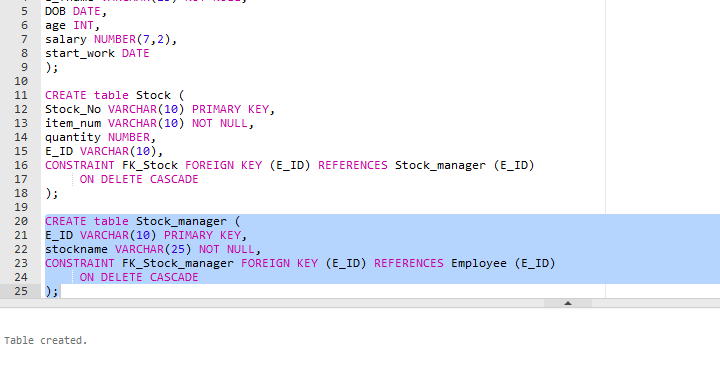


## Table creation

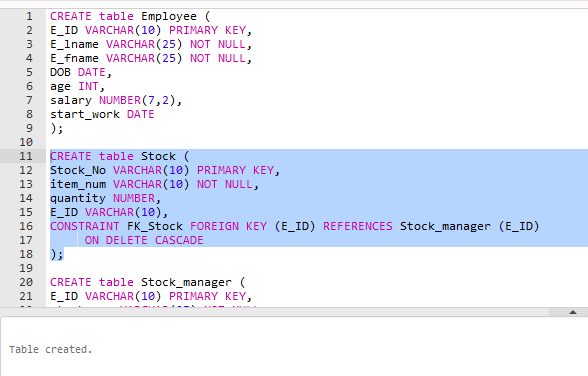
### Employee Table

****

### Stock manager table

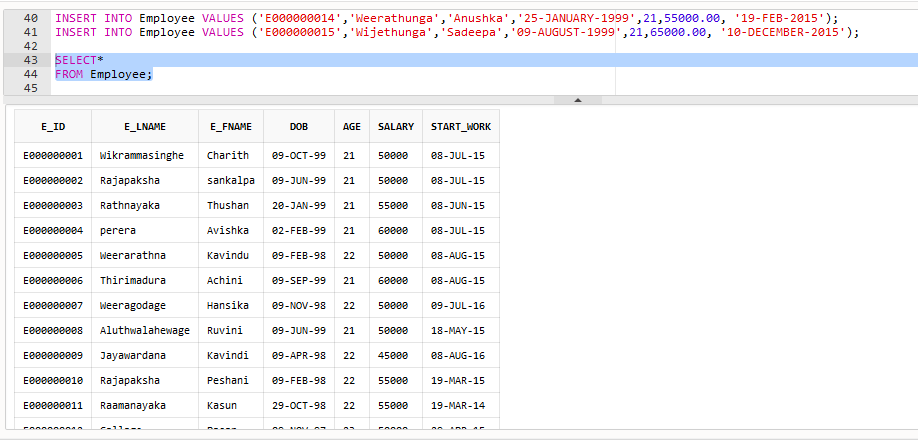
****

### Stock table

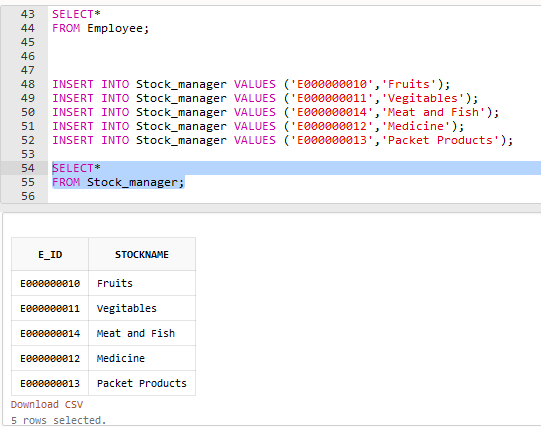
****

## Tables after inserting data

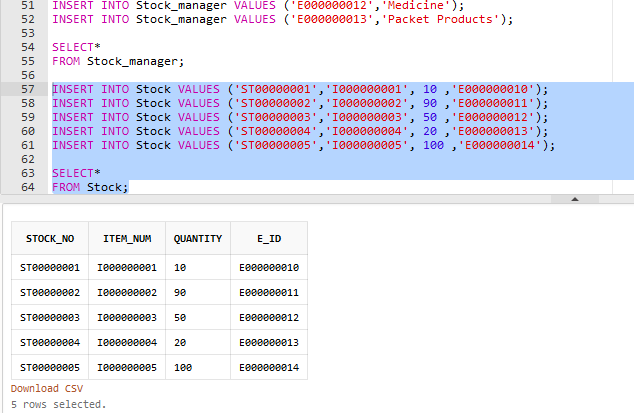
### Employee table

****

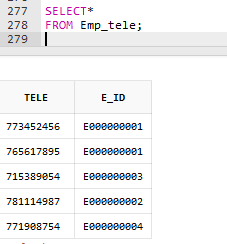
### 2. Stock manager table

****

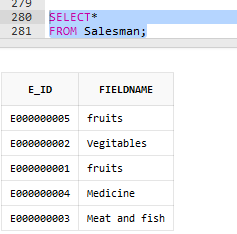
### Stock table

****

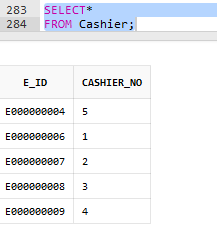
### Emp\_tele table



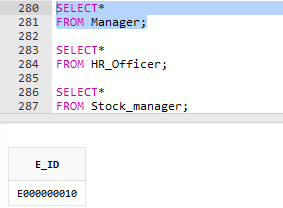
### Saleman table



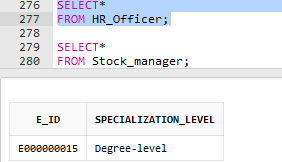
### Cashier table



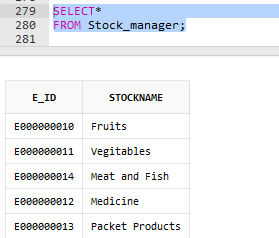
### Manager table



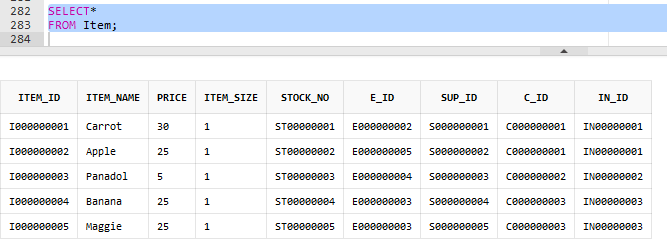
### HR\_Officer table



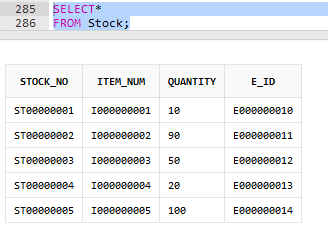
### Stock\_manager



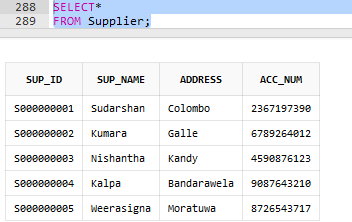
### Item table



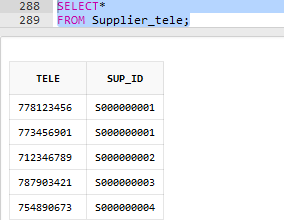
### Stock table



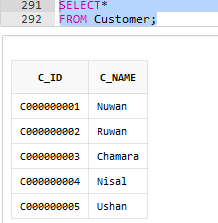
### Supplier table



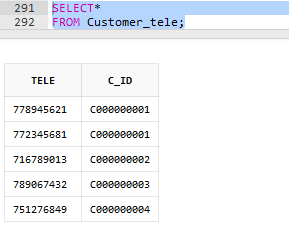
### Supplier\_tele table



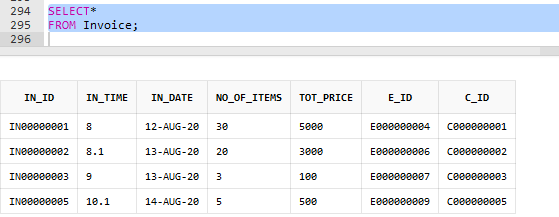
### Customer table



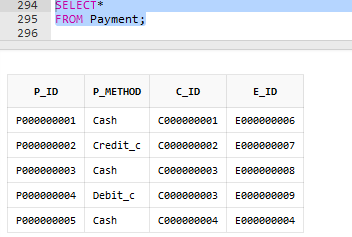
### Customer\_tele table



### Invoice table



### Payment table



## Transactions of our database

### Transaction 1 – Adding the payments once customer has paid the invoice

Suppose a customer has bought items worth 1000.00 by debit card or credit card. The transaction involves,

* Read the current balance of the customer card
* Reduce 1000.00 from the customer’s card
* Read the sum of the transactions taken place so far (current sum)
* Add 1000.00 to the current sum
* Update and read the balance of the supermarket database with the new balance

Further elaborating, this is what happens in the customer’s account.

Old\_Balance = C.balance

New\_Balance = Old\_Balance - 1000

C.balance = New\_Balance

And this is what happens in the supermarket store database sum.

Old\_Balance = S.balance

New\_Balance = Old\_Balance + 1000

S.balance = New\_Balance

This transaction is said to be a COMMIT transaction because all the changes or the payment transactions that were conducted successfully in this session would be saved once the application is closed.

### Transaction 2 – Adding new items to the stock table of the database

Suppose 200kg of rice has been purchased for the store and it has to be added to the stock database.

* Read the available amount of rice in the stock
* Add 100kg of newly purchased amount of rice to the database
* Update and read the new stock amount of rice in the database

Elaborating,

Old\_RiceStock = R.stock

New\_stock = Old\_RiceStock + 200

R.stock = New\_stock

This is a COMMITED transaction because all the changes of the session will be saved when the application is closed.

### **Transaction 3 – Viewing details from invoices which are not older than 13-AUG-2020**.

We actually do not want to permanently delete the invoice details older than the above mentioned data, but for analytic purposes, we want to delete them for the moment.

* Read all the rows of the invoice table
* Read the in\_Date column of the invoice table
* Delete any row which have their in\_Date column older than 13-AUG-2020.

Simply, we can run this code to view the required details and undo the modifications from next session onwards.

DELETE FROM Customers

WHERE in\_Date < ‘13-AUG-2020’;

ROLLBACK

## Access Privileges

ALTER SESSION SET "\_ORACLE\_SCRIPT"=TRUE;

--Creating roles--

CREATE ROLE Cashier;

CREATE ROLE Salesman;

CREATE ROLE Manager;

CREATE ROLE HR\_Officer;

CREATE ROLE Stock\_manager;

--Creating users--

CREATE USER Charith IDENTIFIED BY Charith123;

CREATE USER Sankalpa IDENTIFIED BY Sankalpa123;

CREATE USER Thushan IDENTIFIED BY Thushan123;

CREATE USER Avishka IDENTIFIED BY Avishka123;

CREATE USER Kavindu IDENTIFIED BY Kavindu123;

CREATE USER Achini IDENTIFIED BY Achini123;

CREATE USER Hansika IDENTIFIED BY Hansika123;

CREATE USER Ruvini IDENTIFIED BY Ruvini123;

CREATE USER Kavindi IDENTIFIED BY Kavindi123;

CREATE USER Peshani IDENTIFIED BY Peshani123;

CREATE USER Kasun IDENTIFIED BY Kasun123;

CREATE USER Pasan IDENTIFIED BY Kavindu123;

CREATE USER Ashini IDENTIFIED BY Asihini123;

CREATE USER Anushka IDENTIFIED BY Anushka123;

CREATE USER Sadeepa IDENTIFIED BY Sadeepa123;

--global\_name retrieves the database name--

SELECT \* FROM global\_name;

--Granting specific access privilige rights--

GRANT CONNECT TO Cashier;

GRANT select ON Item TO Cashier;

GRANT CONNECT TO Salesman;

GRANT select ON Item TO Salesman;

GRANT ALL Privileges TO Manager;

GRANT CONNECT TO HR\_Officer;

GRANT select, insert, update, delete ON Employee, Employee\_tele TO HR\_Officer;

GRANT select, insert, update, delete ON Employee\_tele TO HR\_Officer;

GRANT select ON Item TO HR\_Officer;

GRANT CONNECT TO Stock\_manager;

GRANT select, insert, update, delete ON Stock TO Stock\_manager;

GRANT select ON Item TO Stock\_manager;

GRANT select ON Supplier TO Stock\_manager;

GRANT select ON Supplier\_tele TO Stock\_manager;

--Grant users for roles--

GRANT Cashier TO Achini;

GRANT Cashier TO Avishka;

GRANT Cashier TO Hansika;

GRANT Cashier TO Ruvini;

GRANT Cashier TO Kavindi;

GRANT Salesman TO Avishka;

GRANT Salesman TO Charith;

GRANT Salesman TO Sankalpa;

GRANT Salesman TO Thushan;

GRANT Salesman TO Kavindu;

GRANT Manager TO Peshani;

GRANT Manager TO Kasun;

GRANT Manager TO Pasan;

GRANT Manager TO Ashini;

GRANT Manager TO Anushka;

GRANT HR\_Officer TO Sadeepa;

GRANT HR\_Officer TO Anushka;

GRANT HR\_Officer TO Ashini;

GRANT HR\_Officer TO Peshani;

GRANT HR\_Officer TO Kasun;

GRANT Stock\_manager TO Peshani;

GRANT Stock\_manager TO Kasun;

GRANT Stock\_manager TO Pasan;

GRANT Stock\_manager TO Ashini;

GRANT Stock\_manager TO Anushka;

--Granting the rights to create stored functions and procedures--

GRANT CREATE PROCEDURE TO Manager;

GRANT CREATE FUNCTION TO Manager;

## Attacks and countermeasures for the database if connected with an web application

Databases-by definition-contain details and details including sensitive information such as credit card details are always of interest to the hackers. This implies databases are indeed an enticing honeypot, which is why database protection is fundamentally important.

Our database is ideally not meant to be connected to a web application because it is intended to be used offline in the supermarket store only. We are supposing that it is connected to a web application and suggesting these security mechanisms for better service.

The supermarket store contains employee details, item details, stock details, invoice details, payment details, customer details, etc. which most of them are sensitive information especially the payment details, customer details, and the employee details. Below are the possible attacks for databases connected to a web application.

### Cross-site scripts (XSS)

This is one of the most common attacks for a database that uses a web application to respond to user requests. The attacks embed malicious codes into the client-side scripts when requesting data. Once the web page is loaded, the code becomes activated and the attacker can retrieve any information stored in the database as per their preferences.

XSS is of 3 types:

* Reflected
* Stored
* DOM-based

Techniques to prevent XSS attacks

* Escaping user inputs

This method avoids users sending requests typed maliciously. For example, receiving user requests containing < or > are not rendered.

* Input validation

This is also somewhat similar to the above mentioned but is preventing users to input specific characters that might contain malicious codes.

* Sanitizing user inputs

In this technique, the user inputs suspected of having malicious codes (identified by special characters) are allowed to be sent to the client’s end by converting them into a valid form such as enclosing the user inputs within double-quotes.

* Output encoding and escaping untrusted characters.
* Enabling Content-Security-policy (CSP)

### SQL injection

SQL injection attacks allow attackers to spoof identities, exploit existing data, cancel transactions and user balance, and require full disclosure of all data on the server that would otherwise be open only to the database administrator.

In SQL injections, the attackers enter malicious SQL codes to the input fields to be executed at the backend. The attacker then can modify, insert, or delete the existing data of the database also can act as an administrator of the database.

SQL injections can be avoided by installing a firewall to validate the user requests and to filter the network traffic. Small privilege access to the database server, passwords, bank account information, credit card details, branch routing number will all be redacted. This information should be encrypted to prevent identity fraud before saving it to the database.

### Automated threats

These attacks are caused by using internet bots. The hackers allow the web pages to do large amounts of repetitive work which causes many types of automated threats such as Account aggregation, carding, scraping, Denial-of-Service (DoS) are few automated threats to happen.

We can use,

* Security Event Manager (SEM) – SEM can track the network traffic and notify when there is unusual behavior in the network before they become overwhelming so the admin of the database can disable the privileges. The SEM also kills those programs and disables the relevant IP or USBs from sending requests to the database or from accessing the web application.
* Cloudflare – This is a scalable tool that integrates multiple DoS mitigation techniques into one single tool.
* Imperva – This shields and protects the database and the web application from getting heavy network.

### Brute Force

Username and password attacks are common nowadays for most of the databases. Oracle provides default passwords for the databases which makes the database vulnerable for attacks. The newer versions do not provide default passwords and if it provides default passwords, we should make sure that default passwords are changed and we are using our own strong passwords. We can use password cracking tools for this too.

### Privilege escalation

This is the most common type of attack for the databases. It occurs when the users are granted more privileges than they are intended to. This can cause leakage of sensitive data from the employees already working in the store. The employees should be undergone with privacy protection training and necessary policies should be established in addition to granting only the needed privileges to the users. Also, mechanisms can be implemented to lock the accounts after three unsuccessful attempts to log in to the user accounts. Some other countermeasures are,

• Invalidate tokens and cookies after logout.

• Forced login/logout after a password change.

• Server-side resource restriction e.g. directories.

• Restrict access to all resources basis roles.

### Stolen backup files

Backup files stored in hardware devices can be stolen by someone who needs to access sensitive information of the database. Therefore, the backup files must be encrypted using standard encryption methods when storing them.

### Unrestricted File Upload

Uploading files in an application poses tremendous danger for applications. The first step to launch a device attack is to insert the malicious code block into the device that is to be targeted. Then attack only must find a way to execute the code. Using the uploader file lets the attacker start the first step to insert the block of malicious code into the network.

Countermeasures can be suggested as validating that the data content meets the intended types for all approved forms for uploaded file content and reject attempts for all those file extensions that can be executed on the server side or that can be unsafe on the client side. Assign a filename created by the program to any uploaded file, and do not include any portion of the original filename. Do not serve or process files uploaded from web server directories by the user that may have script mapping enabled.

### Broken Authentication

"Lost authentication happens when session dependent information is mismanaged by the client, so that the user's identity is compromised. The details may be in the form of session cookies, passwords, hidden keys, etc."

The object here is either to break into someone else's session or to use a session that has been terminated by the user or to capture information relevant to the session.

Countermeasures are,

* Use of multifactor authentication
* Session isolation
* Idle session timeouts
* Using secured cookies

### XML External Entities (XXE)

"An application is susceptible to XXE attacks because it allowed users to upload a malicious XML that further exploits insecure functionality and/or dependencies," which may be used to execute functionality, steal data and perform other malicious tasks.

Countermeasures are,

* Avoid serialization of sensitive data
* Implement whitelisting approach at server side to prevent malicious XML upload.
* Use of WAF to detect and block XXE.
* Code review

### Security misconfigurations

Developers and IT workers maintain accessibility and not safety. The settings on the application server, MySQL server, proxy, software, and other devices need to be in accordance with the specifications for security. Most protection criteria are ignored until the experts or hackers recognize them.

Examples of these protection misconfigurations include insecure passwords, default keys, server-save file files, file folders, default error messages etc.

Countermeasures are,

* Have a hardening process in place for both hardware and applications. Do ensure that defaults are changed.
* Install only the required features from a framework.
* Review the security of the configurations at fixed intervals.

## Implementing security countermeasures

### Views

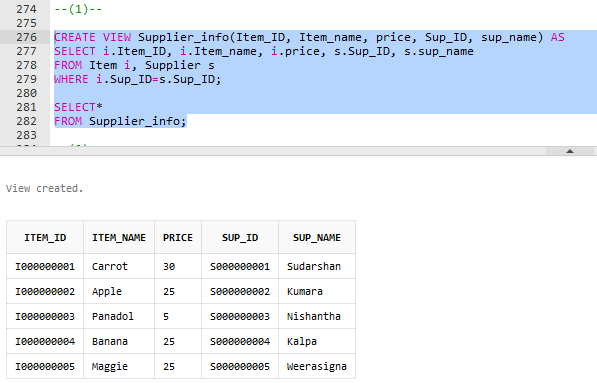
1. **View to select data from both Item and Supplier tables**

CREATE VIEW Supplier\_info(Item\_ID, Item\_name, price, Sup\_ID, sup\_name) AS

SELECT i.Item\_ID, i.Item\_name, i.price, s.Sup\_ID, s.sup\_name

FROM Item i, Supplier s

WHERE i.Sup\_ID=s.Sup\_ID;



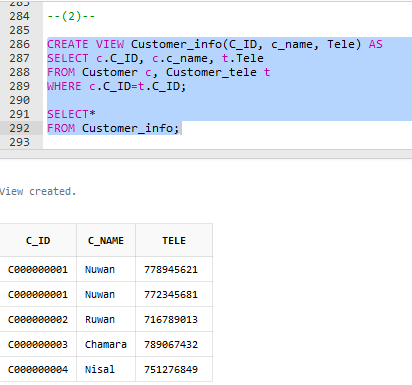
1. **View to select data from both Customer and Customer\_tele tables**

CREATE VIEW Customer\_info(C\_ID, c\_name, Tele) AS

SELECT c.C\_ID, c.c\_name, t.Tele

FROM Customer c, Customer\_tele t

WHERE c.C\_ID=t.C\_ID;



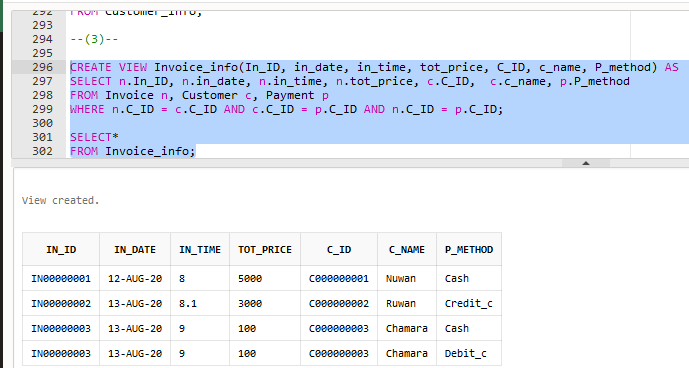
1. **View to select data from all Invoice, Customer, and Payment tables**

CREATE VIEW Invoice\_info(In\_ID, in\_date, in\_time, tot\_price, C\_ID, c\_name, P\_method) AS

SELECT n.In\_ID, n.in\_date, n.in\_time, n.tot\_price, c.C\_ID, c.c\_name, p.P\_method

FROM Invoice n, Customer c, Payment p

WHERE n.C\_ID = c.C\_ID AND c.C\_ID = p.C\_ID AND n.C\_ID = p.C\_ID



### Stored Functions

1. **Function to return the total revenue for a given day**

CREATE OR REPLACE FUNCTION TotRevenue(Date1 DATE) RETURN NUMBER

AS

revenue NUMBER;

BEGIN

SELECT SUM(tot\_price) INTO revenue

FROM Invoice i

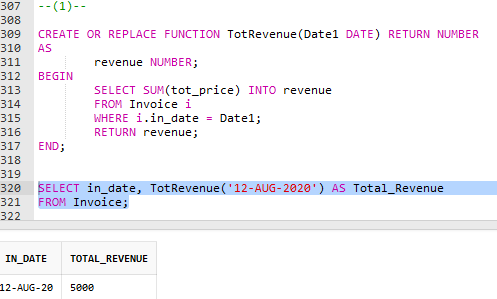
WHERE i.in\_date = Date1;

RETURN revenue;

END;

SELECT in\_date, TotRevenue(’12-AUG-2020’) AS Total\_Revenue

FROM Invoice;



1. **Function to return the minimum invoice amount for a given date**

CREATE OR REPLACE FUNCTION MinPrice(inDate DATE) RETURN NUMBER

AS

Minimum NUMBER;

BEGIN

SELECT Min(tot\_price) INTO Minimum

FROM Invoice

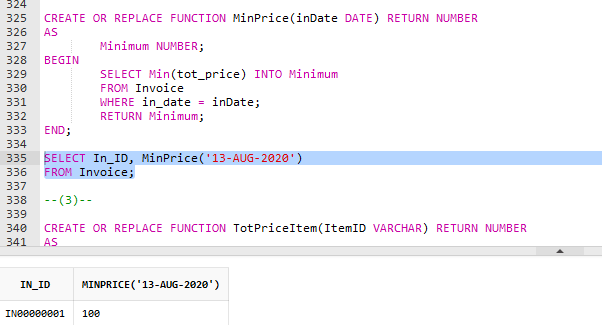
WHERE in\_date = inDate;

RETURN Minimum;

END;

SELECT In\_ID, MinPrice('13-AUG-2020')

FROM Invoice;



1. **Function to return the maximum invoice amount for a given date**

CREATE OR REPLACE FUNCTION MaxPrice(inDate1 DATE) RETURN NUMBER

AS

Maximum NUMBER;

BEGIN

SELECT Max(tot\_price) INTO Maximum

FROM Invoice

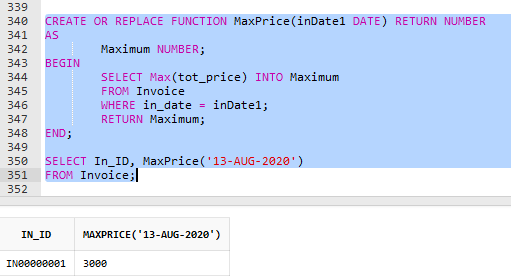
WHERE in\_date = inDate1;

RETURN Maximum;

END;

SELECT In\_ID, MaxPrice('13-AUG-2020')

FROM Invoice;



### Stored Procedures

1. **Procedure to update Employee table salary fields by a certain bonus amount**

CREATE OR REPLACE PROCEDURE BONUS(amount NUMBER)

AS

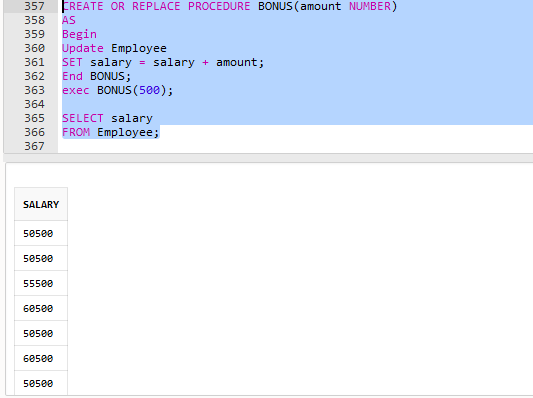
Begin

Update Employee

SET salary = salary + amount;

End BONUS;

exec BONUS(500);



1. **Increase salary field of the Employee table by a certain percentage**

CREATE OR REPLACE PROCEDURE IncreaseSalary (incr DECIMAL)

AS

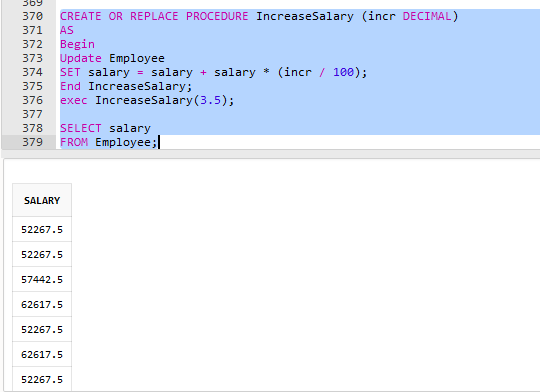
Begin

Update Employee

SET salary = salary + salary \* (incr / 100);

End IncreaseSalary;

exec IncreaseSalary(3.5);



1. **Updating the price of an item for a given item code in the Item table**

CREATE OR REPLACE PROCEDURE SetPrice(increaseP DECIMAL, ItemID VARCHAR)

AS

BEGIN

UPDATE Item

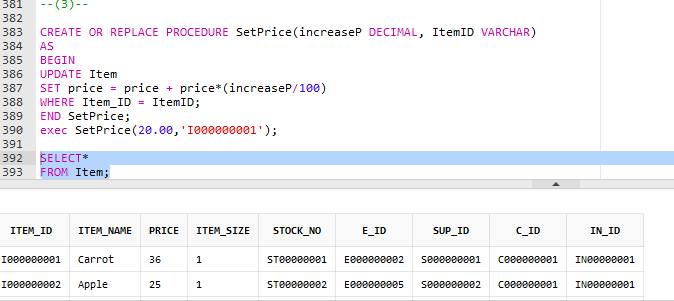
SET price = price + price\*(increaseP/100)

WHERE Item\_ID = ItemID;

END SetPrice;

Execution

exec SetPrice(20.00,‘I000000001’)



### Triggers

1. **A trigger for the Employee table. If the DOB field of a new insert is NULL, the reminders table would display a message informing to enter data to the date of birth field.**

CREATE TABLE reminders (

memberId VARCHAR(10) PRIMARY KEY,

message VARCHAR(255) NOT NULL

);

CREATE TRIGGER after\_members\_insert

AFTER INSERT

ON Employee FOR EACH ROW

BEGIN

IF :new.DOB IS NULL THEN

INSERT INTO reminders(memberId, message)

VALUES(:new.E\_ID,'Hi, please update your date of birth.');

END IF;

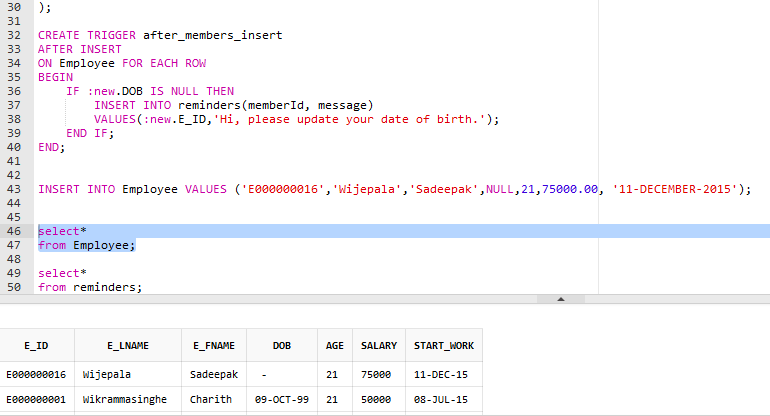
END;

--TESTING--

INSERT INTO Employee VALUES ('E000000016','Wijepala','Sadeepak',NULL,21,75000.00, '11-DECEMBER-2015');

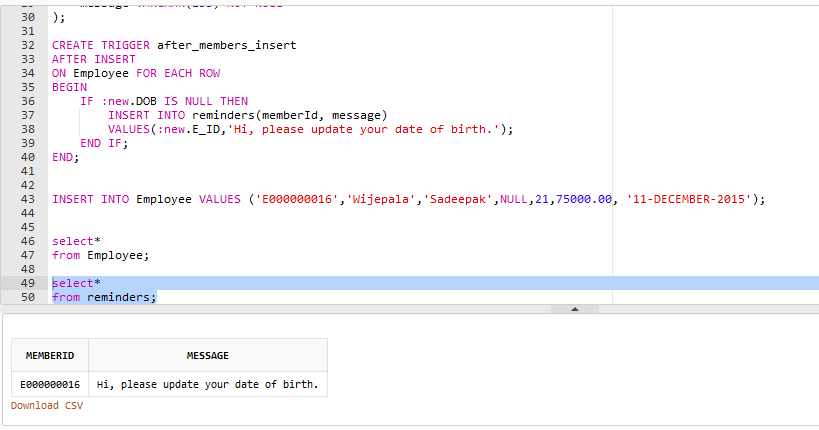
SELECT\*

FROM Employee;



SELECT\*

FROM reminders;



1. **When data from the salary field of the Employee is deleted, the total salary field of the SalaryDel table changes after reducing that deleted amount from the total salary of all the employees.**

CREATE TABLE SalaryDe(

salary\_D number(15,8) NOT NULL

);

CREATE TRIGGER after\_salaries\_delete

AFTER DELETE

ON Employee FOR EACH ROW

BEGIN

UPDATE SalaryDe

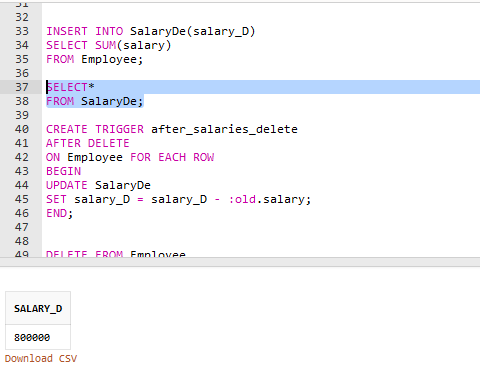
SET salary\_D = salary\_D - :old.salary;

END;

INSERT INTO SalaryDe(salary\_D)

SELECT SUM(salary)

FROM Employee;



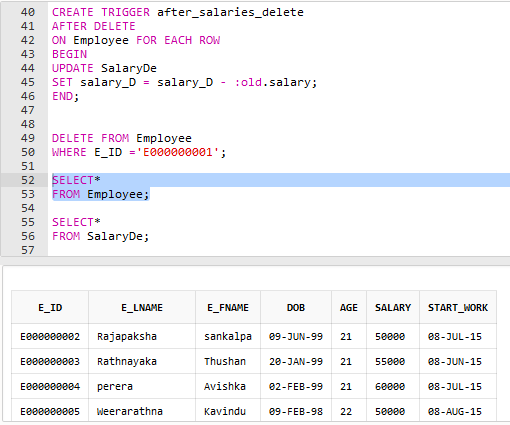
--TESTING--

DELETE FROM Employee

WHERE E\_ID = ‘E000000001’;

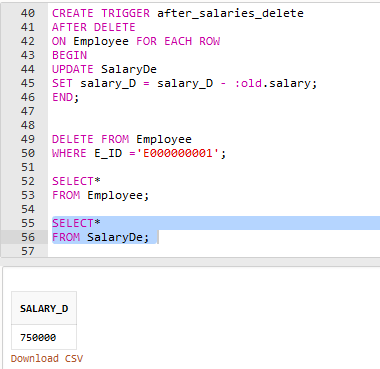
SELECT\*

FROM Employee;



SELECT\*

FROM SalaryDe;



1. **When the quantity field of the Stock table is updated, the SalaryARCHIVES table of the**

CREATE TABLE changeQuantity(

ItemID VARCHAR(10) PRIMARY KEY,

afterQuantity NUMBER

);

CREATE TRIGGER after\_sales\_update

AFTER UPDATE

ON Stock FOR EACH ROW

BEGIN

IF :OLD.quantity <> :NEW.quantity THEN

INSERT INTO changeQuantity(ItemID,afterQuantity)

VALUES(:OLD.item\_num,:NEW.quantity);

END IF;

END;

--TESTING--

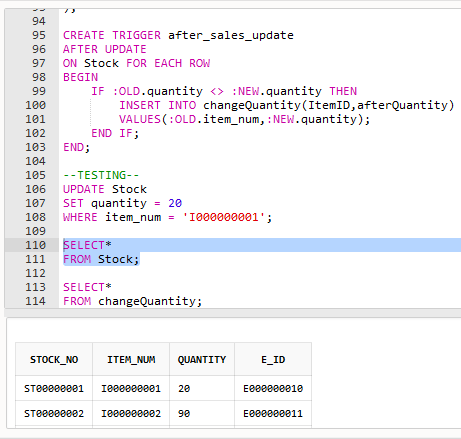
UPDATE Stock

SET quantity = 20

WHERE item\_num = ' I000000001';

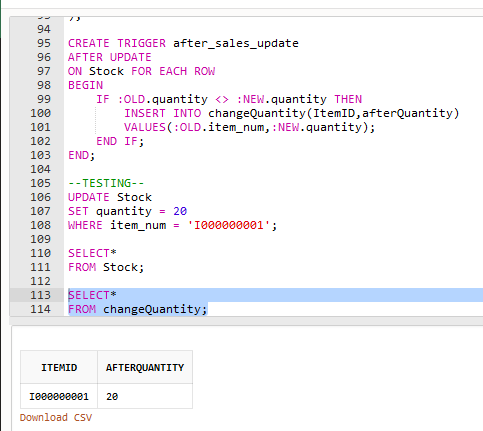
SELECT\*

FROM Stock;



SELECT\*

FROM changeQuantity;



## Recovery mechanisms

Shadow paging is a database recovery technique in which all the transactions are executed in the primary memory or shadow copy of the database. Using this kind of recovery technique will provide atomicity and durability in the proposed database system. Once all the transactions are completely executed, this will be updated to the database. If there is any failure in the middle of the transaction, it will not reflect in the database.

A copy of the database is used in a transaction to update, which was pointed by the database pointer to the consistent copy of the database. The old copy is deleted when the DB pointer is modified to point to a new copy of the Database. But if there is any failure during the transaction, the pointer will be still pointing to the old copy of the database, and the shadow database will be deleted.

When a transaction starts both page tables are identical, the shadow page table is never changed throughout the transaction. And, the current page table may be changed when a transaction performs a write operation.

**And, it is important to know how this recovery technique is going to work in our database.**

The database is partitioned into several fixed-length blocks, which known as pages, there are several pages, numbered from 1. These pages do not need to be stored in any order on disk. If we want to find a relevant page on the disk, there must be a way to find it. So, there are page tables that consist of several entries-one for each database.

Each of these entries contains a pointer to a page in the disk. Suppose that there is a transaction to perform a write operation that resides on the selected page. If the page is not already in the main memory, then the system issues an input. And, if the write first performed on the page which transaction is to be performed, the system modifies the current page to find the unused page on disk, then it deletes the page (unused) found previously from the list of the free page frames, and it copies the content to the relevant page to the unused page. Then it modifies the current page table and assigning values in the buffer page.

**Using this kind of recovery technique will give us advantages**, storing the shadow page table in nonvolatile storage, so the execution of a transaction in a database can be recovered in the event of a crash, or a transaction commits happened. When the system comes back up a crash, it will copy the shadow page table into the main memory, and it can be used for subsequent transaction processing. In some cases, we do not care whether the current page table is lost in a crash since the system recovers by using the shadow page table.

Using this kind of database recovery technique will recover the databases from issues very fast.

## Complete Code

--Table creation--

CREATE table Employee (

E\_ID VARCHAR(10) PRIMARY KEY,

E\_lname VARCHAR(25) NOT NULL,

E\_fname VARCHAR(25) NOT NULL,

DOB DATE NULL,

age INT,

salary NUMBER(7,2),

start\_work DATE

);

CREATE table Emp\_tele (

Tele CHAR(10) PRIMARY KEY,

E\_ID VARCHAR(10) NOT NULL,

CONSTRAINT FK\_Emp\_tele FOREIGN KEY(E\_ID) REFERENCES Employee (E\_ID)

ON DELETE CASCADE

);

CREATE table Salesman (

E\_ID VARCHAR(10) PRIMARY KEY,

fieldname VARCHAR(25) NOT NULL,

CONSTRAINT FK\_Salesman FOREIGN KEY (E\_ID) REFERENCES Employee (E\_ID)

ON DELETE CASCADE

);

CREATE table Cashier (

E\_ID VARCHAR(10) PRIMARY KEY,

cashier\_no VARCHAR(10) NOT NULL,

CONSTRAINT FK\_Cachier FOREIGN KEY (E\_ID) REFERENCES Employee (E\_ID)

ON DELETE CASCADE

);

CREATE table Manager (

E\_ID VARCHAR(10) PRIMARY KEY,

CONSTRAINT FK\_Manager FOREIGN KEY (E\_ID) REFERENCES Employee (E\_ID)

ON DELETE CASCADE

);

CREATE table HR\_Officer (

E\_ID VARCHAR(10) PRIMARY KEY,

Specialization\_level VARCHAR(25),

CONSTRAINT FK\_HR\_officer FOREIGN KEY (E\_ID) REFERENCES Employee (E\_ID)

ON DELETE CASCADE

);

CREATE table Stock\_manager (

E\_ID VARCHAR(10) PRIMARY KEY,

stockname VARCHAR(25) NOT NULL,

CONSTRAINT FK\_Stock\_manager FOREIGN KEY (E\_ID) REFERENCES Employee (E\_ID)

ON DELETE CASCADE

);

CREATE table Item (

Item\_ID VARCHAR(10) PRIMARY KEY,

Item\_name VARCHAR(25) NOT NULL,

price NUMBER(7,2) NOT NULL,

item\_size NUMBER,

Stock\_No VARCHAR(10) NOT NULL,

E\_ID VARCHAR(10),

Sup\_ID VARCHAR(10),

C\_ID VARCHAR(10),

In\_ID VARCHAR(10),

CONSTRAINT FK\_Item FOREIGN KEY (Stock\_No) REFERENCES Stock (Stock\_No)

ON DELETE CASCADE,

CONSTRAINT FK\_Item1 FOREIGN KEY (E\_ID) REFERENCES Salesman (E\_ID)

ON DELETE CASCADE,

CONSTRAINT FK\_Item2 FOREIGN KEY (Sup\_ID) REFERENCES Supplier (Sup\_ID)

ON DELETE CASCADE,

CONSTRAINT FK\_Item3 FOREIGN KEY (C\_ID) REFERENCES Customer (C\_ID)

ON DELETE CASCADE,

CONSTRAINT FK\_Item4 FOREIGN KEY (In\_ID) REFERENCES Invoice (In\_ID)

ON DELETE CASCADE

);

CREATE table Stock (

Stock\_No VARCHAR(10) PRIMARY KEY,

item\_num VARCHAR(10) NOT NULL,

quantity NUMBER,

E\_ID VARCHAR(10),

CONSTRAINT FK\_Stock FOREIGN KEY (E\_ID) REFERENCES Stock\_manager (E\_ID)

ON DELETE CASCADE

);

CREATE table Supplier (

Sup\_ID VARCHAR(10) PRIMARY KEY,

sup\_name VARCHAR(25) NOT NULL,

address VARCHAR(25),

acc\_num NUMBER(10)

);

CREATE table Supplier\_tele (

Tele CHAR(10) PRIMARY KEY,

sup\_ID VARCHAR(10) NOT NULL,

CONSTRAINT FK\_Supplier\_tele FOREIGN KEY (Sup\_ID) REFERENCES Supplier (Sup\_ID)

ON DELETE CASCADE

);

CREATE table Invoice (

In\_ID VARCHAR(10) PRIMARY KEY,

in\_time NUMBER(4,2),

in\_date DATE NOT NULL,

no\_of\_items NUMBER NOT NULL,

tot\_price NUMBER(7,2) NOT NULL,

E\_ID VARCHAR(10),

C\_ID VARCHAR(10),

CONSTRAINT FK\_Invoice FOREIGN KEY (E\_ID) REFERENCES Cashier (E\_ID)

ON DELETE CASCADE,

CONSTRAINT FK\_Invoice1 FOREIGN KEY (C\_ID) REFERENCES Customer (C\_ID)

ON DELETE CASCADE

);

CREATE table Customer (

C\_ID VARCHAR(10) PRIMARY KEY,

c\_name VARCHAR(25)

);

CREATE table Customer\_tele (

Tele CHAR(10) PRIMARY KEY,

C\_ID VARCHAR(10) NOT NULL,

CONSTRAINT FK\_Customer\_tele FOREIGN KEY (c\_ID) REFERENCES Customer (C\_ID)

ON DELETE CASCADE

);

CREATE table Payment (

P\_ID VARCHAR(10) PRIMARY KEY,

P\_method VARCHAR(10) NOT NULL,

C\_ID VARCHAR(10),

E\_ID VARCHAR(10),

CONSTRAINT FK\_Payment FOREIGN KEY (C\_ID) REFERENCES Customer (C\_ID)

ON DELETE CASCADE,

CONSTRAINT FK\_Payment1 FOREIGN KEY (E\_ID) REFERENCES Cashier (E\_ID)

ON DELETE CASCADE

);

--Data insertion--

INSERT INTO Employee VALUES ('E000000001','Wikrammasinghe','Charith','09-OCT-1999',21,50000.00, '08-JULY-2015');

INSERT INTO Employee VALUES ('E000000002','Rajapaksha','sankalpa','09-JUNE-1999',21,50000.00, '08-JULY-2015');

INSERT INTO Employee VALUES ('E000000003','Rathnayaka','Thushan','20-JANUARY-1999',21,55000.00, '08-JUNE-2015');

INSERT INTO Employee VALUES ('E000000004','perera','Avishka','02-FEB-1999',21,60000.00, '08-JULY-2015');

INSERT INTO Employee VALUES ('E000000005','Weerarathna','Kavindu','09-FEB-1998',22,50000.00, '08-August-2015');

INSERT INTO Employee VALUES ('E000000006','Thirimadura','Achini','09-SEP-1999',21,60000.00, '08-August-2015');

INSERT INTO Employee VALUES ('E000000007','Weeragodage','Hansika','09-NOVEMBER-1998',22,50000.00, '09-JULY-2016');

INSERT INTO Employee VALUES ('E000000008','Aluthwalahewage','Ruvini','09-JUNE-1999',21,50000.00, '18-MAY-2015');

INSERT INTO Employee VALUES ('E000000009','Jayawardana','Kavindi','09-APRIL-1998',22,45000.00, '08-August-2016');

INSERT INTO Employee VALUES ('E000000010','Rajapaksha','Peshani','09-FEB-1998',22,55000.00, '19-MARCH-2015');

INSERT INTO Employee VALUES ('E000000011','Raamanayaka','Kasun','29-OCT-1998',22,55000.00, '19-MARCH-2014');

INSERT INTO Employee VALUES ('E000000012','Gallage','Pasan','09-NOV-1997',23,50000.00, '29-APRIL-2015');

INSERT INTO Employee VALUES ('E000000013','Premadasa','Ashini','20-FEB-1998',22,50000.00, '30-MAY-2017');

INSERT INTO Employee VALUES ('E000000014','Weerathunga','Anushka','25-JANUARY-1999',21,55000.00, '19-FEB-2015');

INSERT INTO Employee VALUES ('E000000015','Wijethunga','Sadeepa','09-AUGUST-1999',21,65000.00, '10-DECEMBER-2015');

INSERT INTO Emp\_tele VALUES (0773452456,'E000000001');

INSERT INTO Emp\_tele VALUES (0765617895,'E000000001');

INSERT INTO Emp\_tele VALUES (0715389054,'E000000003');

INSERT INTO Emp\_tele VALUES (0781114987,'E000000002');

INSERT INTO Emp\_tele VALUES (0771908754,'E000000004');

INSERT INTO Salesman VALUES ('E000000005','fruits');

INSERT INTO Salesman VALUES ('E000000002','Vegitables');

INSERT INTO Salesman VALUES ('E000000001','fruits');

INSERT INTO Salesman VALUES ('E000000004','Medicine');

INSERT INTO Salesman VALUES ('E000000003','Meat and fish');

INSERT INTO cashier VALUES ('E000000004',05);

INSERT INTO cashier VALUES ('E000000006',01);

INSERT INTO cashier VALUES ('E000000007',02);

INSERT INTO cashier VALUES ('E000000008',03);

INSERT INTO cashier VALUES ('E000000009',04);

INSERT INTO Manager VALUES ('E000000010’);

INSERT INTO HR\_Officer VALUES ('E000000015','Degree-level');

INSERT INTO Stock\_manager VALUES ('E000000010','Fruits');

INSERT INTO Stock\_manager VALUES ('E000000011','Vegitables');

INSERT INTO Stock\_manager VALUES ('E000000014','Meat and Fish');

INSERT INTO Stock\_manager VALUES ('E000000012','Medicine');

INSERT INTO Stock\_manager VALUES ('E000000013','Packet Products');

INSERT INTO Item VALUES ('I000000001','Carrot',30.00,1,'ST00000001','E000000002','S000000001','C000000001','IN00000001');

INSERT INTO Item VALUES ('I000000002','Apple',25.00,1,'ST00000002','E000000005','S000000002','C000000001','IN00000001');

INSERT INTO Item VALUES ('I000000003','Panadol',5.00,1,'ST00000003','E000000004','S000000003','C000000002','IN00000002');

INSERT INTO Item VALUES ('I000000004','Banana',25.00,1,'ST00000004','E000000003','S000000004','C000000003','IN00000003');

INSERT INTO Item VALUES ('I000000005','Maggie',25.00,1,'ST00000005','E000000003','S000000005','C000000003','IN00000003');

INSERT INTO Stock VALUES ('ST00000001','I000000001', 10 ,'E000000010');

INSERT INTO Stock VALUES ('ST00000002','I000000002', 90 ,'E000000011');

INSERT INTO Stock VALUES ('ST00000003','I000000003', 50 ,'E000000012');

INSERT INTO Stock VALUES ('ST00000004','I000000004', 20 ,'E000000013');

INSERT INTO Stock VALUES ('ST00000005','I000000005', 100 ,'E000000014');

INSERT INTO Supplier VALUES ('S000000001', 'Sudarshan', 'Colombo', 2367197390);

INSERT INTO Supplier VALUES ('S000000002', 'Kumara', 'Galle', 6789264012);

INSERT INTO Supplier VALUES ('S000000003', 'Nishantha', 'Kandy', 4590876123);

INSERT INTO Supplier VALUES ('S000000004', 'Kalpa', 'Bandarawela', 9087643210);

INSERT INTO Supplier VALUES ('S000000005', 'Weerasigna', 'Moratuwa', 8726543717);

INSERT INTO Supplier\_tele VALUES (0778123456, 'S000000001');

INSERT INTO Supplier\_tele VALUES (0773456901, 'S000000001');

INSERT INTO Supplier\_tele VALUES (0712346789, 'S000000002');

INSERT INTO Supplier\_tele VALUES (0787903421, 'S000000003');

INSERT INTO Supplier\_tele VALUES (0754890673, 'S000000004');

INSERT INTO Invoice VALUES ('IN00000001', 8.00, '12-AUG-2020', 30, 5000.00, 'E000000004', 'C000000001');

INSERT INTO Invoice VALUES ('IN00000002', 8.10, '13-AUG-2020', 20, 3000.00, 'E000000006', 'C000000002');

INSERT INTO Invoice VALUES ('IN00000003', 9.00, '13-AUG-2020', 3, 100.00, 'E000000007', 'C000000003');

INSERT INTO Invoice VALUES ('IN00000004', 9.40, '14-AUG-2020', 10, 2000.00, 'E000000008', 'C00000004');

INSERT INTO Invoice VALUES ('IN00000005', 10.10, '14-AUG-2020', 5, 500.00, 'E000000009', 'C000000005');

INSERT INTO Customer VALUES ('C000000001', 'Nuwan');

INSERT INTO Customer VALUES ('C000000002', 'Ruwan');

INSERT INTO Customer VALUES ('C000000003', 'Chamara');

INSERT INTO Customer VALUES ('C000000004', 'Nisal');

INSERT INTO Customer VALUES ('C000000005', 'Ushan');

INSERT INTO Customer\_tele VALUES (0778945621, 'C000000001');

INSERT INTO Customer\_tele VALUES (0772345681, 'C000000001');

INSERT INTO Customer\_tele VALUES (0716789013, 'C000000002');

INSERT INTO Customer\_tele VALUES (0789067432, 'C000000003');

INSERT INTO Customer\_tele VALUES (0751276849, 'C000000004');

INSERT INTO Payment VALUES ('P000000001', 'Cash', 'C000000001', 'E000000006');

INSERT INTO Payment VALUES ('P000000002', 'Credit\_c', 'C000000002', 'E000000007');

INSERT INTO Payment VALUES ('P000000003', 'Cash', 'C000000003', 'E000000008');

INSERT INTO Payment VALUES ('P000000004', 'Debit\_c', 'C000000003', 'E000000009');

INSERT INTO Payment VALUES ('P000000005', 'Cash', 'C000000004', 'E000000004');

--ACCESS PRIVILIGES--

ALTER SESSION SET "\_ORACLE\_SCRIPT"=TRUE;

--Creating roles--

CREATE ROLE Cashier;

CREATE ROLE Salesman;

CREATE ROLE Manager;

CREATE ROLE HR\_Officer;

CREATE ROLE Stock\_manager;

--creating users--

CREATE USER Charith IDENTIFIED BY Charith123;

CREATE USER Sankalpa IDENTIFIED BY Sankalpa123;

CREATE USER Thushan IDENTIFIED BY Thushan123;

CREATE USER Avishka IDENTIFIED BY Avishka123;

CREATE USER Kavindu IDENTIFIED BY Kavindu123;

CREATE USER Achini IDENTIFIED BY Achini123;

CREATE USER Hansika IDENTIFIED BY Hansika123;

CREATE USER Ruvini IDENTIFIED BY Ruvini123;

CREATE USER Kavindi IDENTIFIED BY Kavindi123;

CREATE USER Peshani IDENTIFIED BY Peshani123;

CREATE USER Kasun IDENTIFIED BY Kasun123;

CREATE USER Pasan IDENTIFIED BY Kavindu123;

CREATE USER Ashini IDENTIFIED BY Asihini123;

CREATE USER Anushka IDENTIFIED BY Anushka123;

CREATE USER Sadeepa IDENTIFIED BY Sadeepa123;

--global\_name retrive the database name--

SELECT \* FROM global\_name;

--Granting specific access privilige rights--

GRANT CONNECT TO Cashier;

GRANT select ON Item TO Cashier;

GRANT CONNECT TO Salesman;

GRANT select ON Item TO Salesman;

GRANT ALL Privileges TO Manager;

GRANT CONNECT TO HR\_Officer;

GRANT select, insert, update, delete ON Employee, Employee\_tele TO HR\_Officer;

GRANT select, insert, update, delete ON Employee\_tele TO HR\_Officer;

GRANT select ON Item TO HR\_Officer;

GRANT CONNECT TO Stock\_manager;

GRANT select, insert, update, delete ON Stock TO Stock\_manager;

GRANT select ON Item TO Stock\_manager;

GRANT select ON Supplier TO Stock\_manager;

GRANT select ON Supplier\_tele TO Stock\_manager;

--Grat users for roles--

GRANT Cashier TO Achini;

GRANT Cashier TO Avishka;

GRANT Cashier TO Hansika;

GRANT Cashier TO Ruvini;

GRANT Cashier TO Kavindi;

GRANT Salesman TO Avishka;

GRANT Salesman TO Charith;

GRANT Salesman TO Sankalpa;

GRANT Salesman TO Thushan;

GRANT Salesman TO Kavindu;

GRANT Manager TO Peshani;

GRANT Manager TO Kasun;

GRANT Manager TO Pasan;

GRANT Manager TO Ashini;

GRANT Manager TO Anushka;

GRANT HR\_Officer TO Sadeepa;

GRANT HR\_Officer TO Anushka;

GRANT HR\_Officer TO Ashini;

GRANT HR\_Officer TO Peshani;

GRANT HR\_Officer TO Kasun;

GRANT Stock\_manager TO Peshani;

GRANT Stock\_manager TO Kasun;

GRANT Stock\_manager TO Pasan;

GRANT Stock\_manager TO Ashini;

GRANT Stock\_manager TO Anushka;

--Granting the rights to create stored functions and procedures--

GRANT CREATE PROCEDURE TO Manager;

GRANT CREATE FUNCTION TO Manager;

--VIEWS--

--(1)--

CREATE VIEW Supplier\_info(Item\_ID, Item\_name, price, Sup\_ID, sup\_name) AS

SELECT i.Item\_ID, i.Item\_name, i.price, s.Sup\_ID, s.sup\_name

FROM Item i, Supplier s

WHERE i.Sup\_ID=s.Sup\_ID;

SELECT\*

FROM Supplier\_info;

--(2)--

CREATE VIEW Customer\_info(C\_ID, c\_name, Tele) AS

SELECT c.C\_ID, c.c\_name, t.Tele

FROM Customer c, Customer\_tele t

WHERE c.C\_ID=t.C\_ID;

SELECT\*

FROM Customer\_info;

--(3)--

CREATE VIEW Invoice\_info(In\_ID, in\_date, in\_time, tot\_price, C\_ID, c\_name, P\_method) AS

SELECT n.In\_ID, n.in\_date, n.in\_time, n.tot\_price, c.C\_ID, c.c\_name, p.P\_method

FROM Invoice n, Customer c, Payment p

WHERE n.C\_ID = c.C\_ID AND c.C\_ID = p.C\_ID AND n.C\_ID = p.C\_ID;

SELECT\*

FROM Invoice\_info;

--STORED FUNCTIONS--

--(1)--

CREATE OR REPLACE FUNCTION TotRevenue(Date1 DATE) RETURN NUMBER

AS

revenue NUMBER;

BEGIN

SELECT SUM(tot\_price) INTO revenue

FROM Invoice i

WHERE i.in\_date = Date1;

RETURN revenue;

END;

SELECT in\_date, TotRevenue('12-AUG-2020') AS Total\_Revenue

FROM Invoice;

--(2)--

CREATE OR REPLACE FUNCTION MinPrice(inDate DATE) RETURN NUMBER

AS

Minimum NUMBER;

BEGIN

SELECT Min(tot\_price) INTO Minimum

FROM Invoice

WHERE in\_date = inDate;

RETURN Minimum;

END;

SELECT In\_ID, MinPrice('13-AUG-2020')

FROM Invoice;

--(3)--

CREATE OR REPLACE FUNCTION MaxPrice(inDate1 DATE) RETURN NUMBER

AS

Maximum NUMBER;

BEGIN

SELECT Max(tot\_price) INTO Maximum

FROM Invoice

WHERE in\_date = inDate1;

RETURN Maximum;

END;

SELECT In\_ID, MaxPrice('13-AUG-2020')

FROM Invoice;

--PROCEDURES--

--(1)--

CREATE OR REPLACE PROCEDURE BONUS(amount NUMBER)

AS

Begin

Update Employee

SET salary = salary + amount;

End BONUS;

exec BONUS(500);

SELECT salary

FROM Employee;

--(2)--

CREATE OR REPLACE PROCEDURE IncreaseSalary (incr DECIMAL)

AS

Begin

Update Employee

SET salary = salary + salary \* (incr / 100);

End IncreaseSalary;

exec IncreaseSalary(3.5);

SELECT salary

FROM Employee;

--(3)--

CREATE OR REPLACE PROCEDURE SetPrice(increaseP DECIMAL, ItemID VARCHAR)

AS

BEGIN

UPDATE Item

SET price = price + price\*(increaseP/100)

WHERE Item\_ID = ItemID;

END SetPrice;

exec SetPrice(20.00,'I000000001');

SELECT\*

FROM Item;

--TRIGGERS--

--(1)

CREATE TABLE reminders (

memberId VARCHAR(10) PRIMARY KEY,

message VARCHAR(255) NOT NULL

);

CREATE TRIGGER after\_members\_insert

AFTER INSERT

ON Employee FOR EACH ROW

BEGIN

IF :new.DOB IS NULL THEN

INSERT INTO reminders(memberId, message)

VALUES(:new.E\_ID,'Hi, please update your date of birth.');

END IF;

END;

INSERT INTO Employee VALUES ('E000000016','Wijepala','Sadeepak',NULL,21,75000.00, '11-DECEMBER-2015');

SELECT\*

FROM Employee;

SELECT\*

FROM reminders;

--(2)--

CREATE TABLE SalaryDe(

salary\_D number(15,8) NOT NULL

);

CREATE TRIGGER after\_salaries\_delete

AFTER DELETE

ON Employee FOR EACH ROW

BEGIN

UPDATE SalaryDe

SET salary\_D = salary\_D - :old.salary;

END;

INSERT INTO SalaryDe(salary\_D)

SELECT SUM(salary)

FROM Employee;

DELETE FROM Employee

WHERE E\_ID = 'E000000001';

SELECT\*

FROM SalaryDe;

--(3)--

CREATE TABLE changeQuantity(

ItemID VARCHAR(10) PRIMARY KEY,

afterQuantity NUMBER

);

CREATE TRIGGER after\_sales\_update

AFTER UPDATE

ON Stock FOR EACH ROW

BEGIN

IF :OLD.quantity <> :NEW.quantity THEN

INSERT INTO changeQuantity(ItemID,afterQuantity)

VALUES(:OLD.item\_num,:NEW.quantity);

END IF;

END;

--TESTING--

UPDATE Stock

SET quantity = 20

WHERE item\_num = 'I000000001';

SELECT\*

FROM Stock;

SELECT\*

FROM changeQuantity;