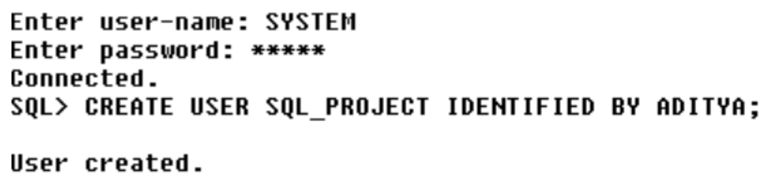
S9APLSQL Project Report: Hotel Management System

User Creation

To begin the project, a dedicated database user was created under the Oracle SYSTEM account. This ensures isolation, security, and better management of project-related objects.
Key Steps:
Connected to the database using the SYSTEM user.
Created a new application user with the syntax:



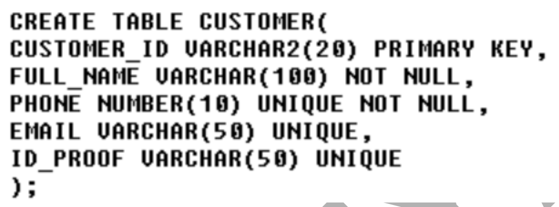
```
Enter user-name: SYSTEM
Enter password: *****
Connected.
SQL> CREATE USER SQL_PROJECT IDENTIFIED BY ADITYA;

User created.
```

To, create new user you have to follow the following syntax:

CREATE USER USER_NAME IDENTIFIED/UNLOCK BY PASSWORD

After creating the user, I have created my project table inside that, user

Tabel 1: Customer



```
CREATE TABLE CUSTOMER(
CUSTOMER_ID VARCHAR2(20) PRIMARY KEY,
FULL_NAME VARCHAR(100) NOT NULL,
PHONE NUMBER(10) UNIQUE NOT NULL,
EMAIL VARCHAR(50) UNIQUE,
ID_PROOF VARCHAR(50) UNIQUE
);
```

The **CUSTOMER** table stores profile details of hotel guests.
It ensures uniqueness and data validation using strongly defined constraints.

Columns & Description
CUSTOMER_ID (PK):
Unique identifier for each customer, used for referencing bookings.

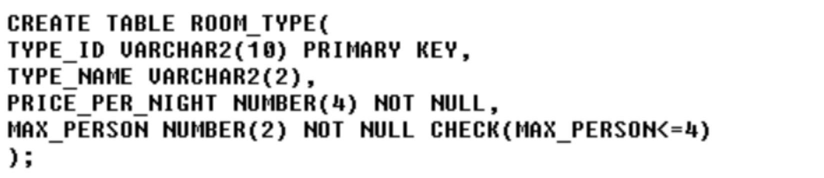
FULL_NAME (NOT NULL):
Stores the full name of the customer; cannot be left empty.

PHONE (UNIQUE, NOT NULL, CHECK):
Contains the customer's 10-digit mobile number.
The UNIQUE constraint avoids duplicates.
An optional CHECK(LENGTH(PHONE)=10) ensures correct length.

EMAIL (UNIQUE, CHECK):
Stores customer email.
UNIQUE avoids repetition;

CHECK(EMAIL LIKE '%.GMAIL.COM') ensures valid format (if applied).
ID_PROOF:
Stores customer verification details like Aadhaar, PAN, or Driving License.

Tabel 2: ROOM_TYPE



```
CREATE TABLE ROOM_TYPE(
TYPE_ID VARCHAR2(10) PRIMARY KEY,
TYPE_NAME VARCHAR2(2),
PRICE_PER_NIGHT NUMBER(4) NOT NULL,
MAX_PERSON NUMBER(2) NOT NULL CHECK(MAX_PERSON<=4)
);
```

Here the 2nd table is of ROOM_TYPE in which there are 4 column

The ROOM_TYPE table categorizes rooms based on size, capacity, and pricing.
It supports better management of room inventory.

Columns & Description

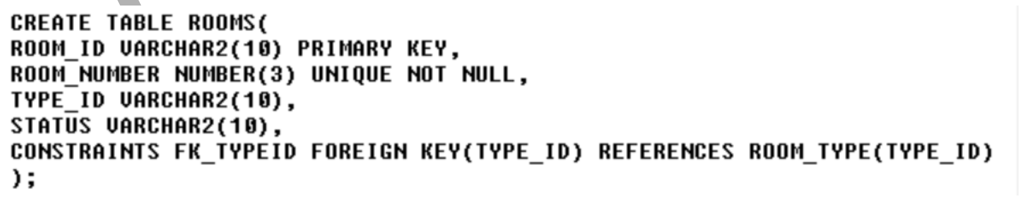
TYPE_ID (PK):
Unique room type identifier.

TYPE_NAME:
Defines category (e.g., Standard, Deluxe, Suite).

PRICE_PER_NIGHT (NOT NULL):
Rate of the room per night; mandatory for billing.

MAX_PERSON (NOT NULL, CHECK):
Maximum occupancy allowed per room type.
The constraint CHECK(MAX_PERSON <= 4) ensures safety and regulatory compliance.

Tabel 3: ROOMS



```
CREATE TABLE ROOMS(
ROOM_ID VARCHAR2(10) PRIMARY KEY,
ROOM_NUMBER NUMBER(3) UNIQUE NOT NULL,
TYPE_ID VARCHAR2(10),
STATUS VARCHAR2(10),
CONSTRAINTS FK_TYPEID FOREIGN KEY(TYPE_ID) REFERENCES ROOM_TYPE(TYPE_ID)
);
```

The ROOMS table represents the physical rooms available in the hotel.

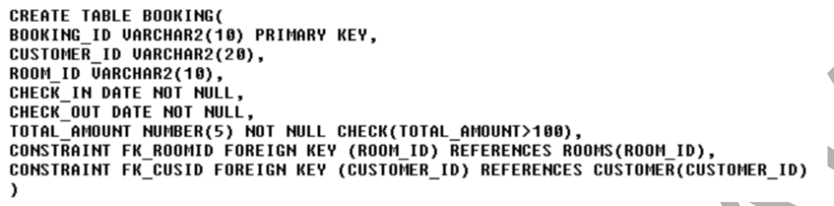
Columns & Description

ROOM_ID (PK):
Unique identifier for each room.

TYPE_ID (FK):
Links to *ROOM_TYPE*, establishing a many-to-one relationship.
ROOM_NUMBER (UNIQUE, NOT NULL):
Actual room number assigned in the hotel (101, 102, 103...).

STATUS:
Indicates room availability (e.g., Available, Occupied, Under Maintenance).
This table helps track the operational status of each room

Tabel 4: BOOKING



```
CREATE TABLE BOOKING(
BOOKING_ID VARCHAR2(10) PRIMARY KEY,
CUSTOMER_ID VARCHAR2(20),
ROOM_ID VARCHAR2(10),
CHECK_IN DATE NOT NULL,
CHECK_OUT DATE NOT NULL,
TOTAL_AMOUNT NUMBER(5) NOT NULL CHECK(TOTAL_AMOUNT>100),
CONSTRAINT FK_ROOMID FOREIGN KEY (ROOM_ID) REFERENCES ROOMS(ROOM_ID),
CONSTRAINT FK_CUSID FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER(CUSTOMER_ID)
)
```

The **BOOKING** table manages all reservation details and connects customers with rooms.

Columns & Description

BOOKING_ID (PK):
Unique booking reference number.

CUSTOMER_ID (FK):
Connects booking to the registered customer.

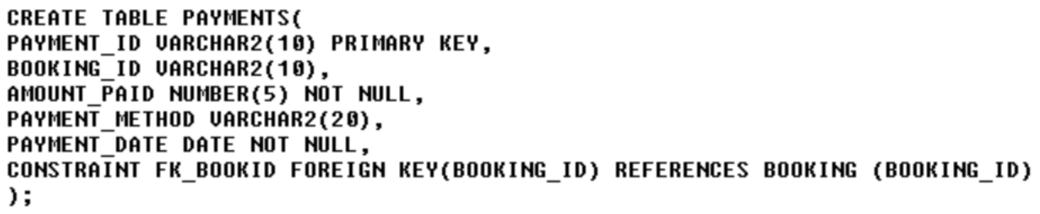
ROOM_ID (FK):
Links booking to the allocated room.

CHECK_IN:
Date on which the customer arrives.

CHECK_OUT:
Date on which the customer leaves.

TOTAL_AMOUNT (NOT NULL):
Final calculated amount including room charges and any additional services.
This table supports billing, availability management, and historical tracking.

Tabel 5: PAYMENTS



```
CREATE TABLE PAYMENTS(
PAYMENT_ID VARCHAR2(10) PRIMARY KEY,
BOOKING_ID VARCHAR2(10),
AMOUNT_PAID NUMBER(5) NOT NULL,
PAYMENT_METHOD VARCHAR2(20),
PAYMENT_DATE DATE NOT NULL,
CONSTRAINT FK_BOOKID FOREIGN KEY(BOOKING_ID) REFERENCES BOOKING (BOOKING_ID)
);
```

The **PAYMENTS** table records all financial transactions related to bookings.

Columns & Description

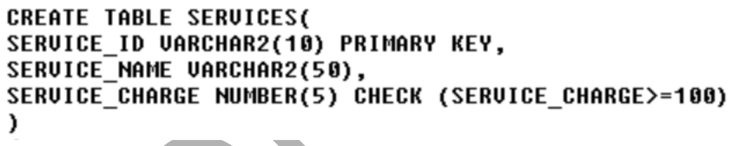
PAYMENT_ID (PK):
Unique identifier for each payment entry.

BOOKING_ID (FK):
Links payment to its corresponding booking.
AMOUNT:
Stores amount received from the customer.

PAYMENT_METHOD:
Includes UPI, Card, or Cash.

PAYMENT_DATE:
Date on which the payment was completed.
It ensures transparency, financial tracking, and auditability of transactions.

Tabel 6: SERVICES



```
CREATE TABLE SERVICES(
SERVICE_ID VARCHAR2(10) PRIMARY KEY,
SERVICE_NAME VARCHAR2(50),
SERVICE_CHARGE NUMBER(5) CHECK (SERVICE_CHARGE>=100)
)
```

The **SERVICES** table stores optional hotel services that customers may opt for.

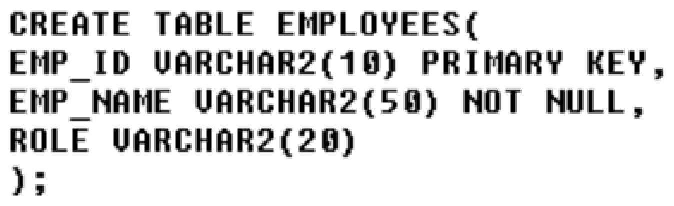
Columns & Description

SERVICE_ID (PK):
Unique identifier for each service.

SERVICE_NAME:
Description of the service (e.g., Breakfast, Spa, Laundry).

PRICE:
Charges associated with each service.
This table helps calculate additional service costs and supports billing integration.

Tabel 7: EMPLOYEE



```
CREATE TABLE EMPLOYEES(
EMP_ID VARCHAR2(10) PRIMARY KEY,
EMP_NAME VARCHAR2(50) NOT NULL,
ROLE VARCHAR2(20)
);
```

The **EMPLOYEES** table stores essential information about the staff working in the hotel.

It helps track employees responsible for managing bookings, services, payments, and customer interactions.

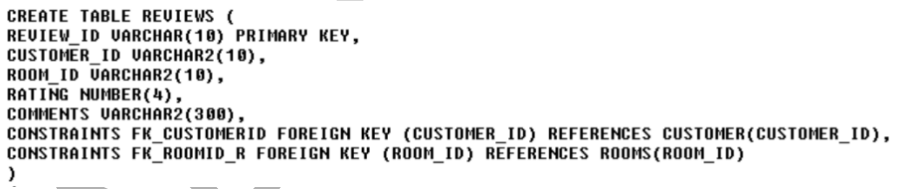
Purpose
To maintain a structured employee database for operational and audit purposes.

Columns & Description
EMP_ID (PK):
A unique identifier assigned to each employee.

EMP_NAME (NOT NULL):
Full name of the employee. This field is mandatory.

ROLE:
Specifies the employee's job role, such as Receptionist, Manager, Housekeeping, etc.
This table provides a foundation for linking employee activities with operational tables like **SERVICE_USAGE**.

Tabel 8: REVIEW



```
CREATE TABLE REVIEWS (
REVIEW_ID VARCHAR(10) PRIMARY KEY,
CUSTOMER_ID VARCHAR2(10),
ROOM_ID VARCHAR2(10),
RATING NUMBER(4),
COMMENTS VARCHAR2(300),
CONSTRAINTS FK_CUSTOMERID FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER(CUSTOMER_ID),
CONSTRAINTS FK_ROOMID_R FOREIGN KEY (ROOM_ID) REFERENCES ROOMS(ROOM_ID)
)
```

The **REVIEWS** table stores customer feedback regarding their stay and experience.
It helps the hotel analyse service quality and maintain customer satisfaction.

Purpose
To capture and manage customer feedback tied to specific rooms and bookings.

Columns & Description

REVIEW_ID (PK):
Unique identifier for each customer review.

CUSTOMER_ID (FK to CUSTOMER):
Links the review to the customer who submitted it.

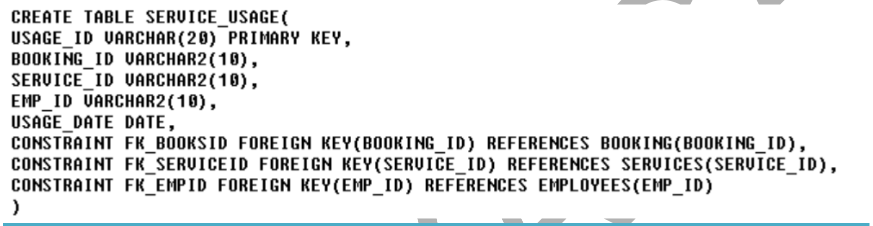
ROOM_ID (FK to ROOMS):
Identifies the room associated with the review.

RATING:
Numeric rating (e.g., 1–5 or 1–10) representing customer satisfaction.

COMMENTS:
Text feedback provided by the customer, up to 300 characters.

Key Constraints
FK_CUSTOMERID_R: Ensures the review belongs to a valid customer.
FK_ROOMID_R: Ensures the review is tied to a valid room.
This table supports service improvement, rating analytics, and quality monitoring.

Tabel 8: SERVICE_USAGE



```
CREATE TABLE SERVICE_USAGE(
USAGE_ID VARCHAR(20) PRIMARY KEY,
BOOKING_ID VARCHAR2(10),
SERVICE_ID VARCHAR2(10),
EMP_ID VARCHAR2(10),
USAGE_DATE DATE,
CONSTRAINT FK_BOOKSID FOREIGN KEY(BOOKING_ID) REFERENCES BOOKING(BOOKING_ID),
CONSTRAINT FK_SERVICEID FOREIGN KEY(SERVICE_ID) REFERENCES SERVICES(SERVICE_ID),
CONSTRAINT FK_EMPID FOREIGN KEY(EMP_ID) REFERENCES EMPLOYEES(EMP_ID)
)
```

The SERVICE_USAGE table records the usage of additional services (e.g., spa, food, laundry) during a customer's stay.
It provides detailed tracking for billing, reporting, and staff assignment.

Purpose
To maintain a transaction-level history of services availed by customers along with staff involvement.

Columns & Description

USAGE_ID (PK):
Unique identifier for each service usage transaction.

BOOKING_ID (FK to BOOKING):
Connects the service usage to a specific booking.

SERVICE_ID (FK to SERVICES):
Identifies the service that was used.

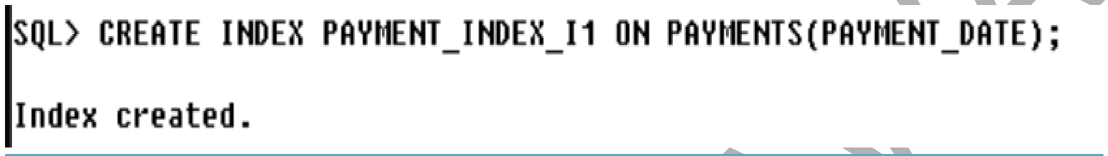
EMP_ID (FK to EMPLOYEES):
Identifies the employee who provided or managed the service.

USAGE_DATE:
The exact date when the service was provided.

Key Constraints
FK_BOOKSID: Ensures service usage is tied to a valid booking.
FK_SERVICEID: Validates that the referenced service exists.
FK_EMPID: Ensures that the recorded employee is valid and registered.

This table is essential for billing accuracy, employee workload tracking, and service analytics.

INDEXES:



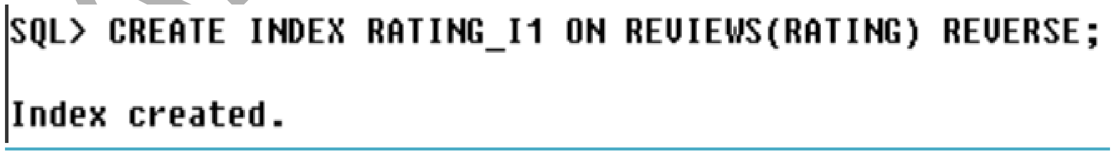
```
SQL> CREATE INDEX PAYMENT_INDEX_I1 ON PAYMENTS(PAYMENT_DATE);

Index created.
```

Purpose

A standard B-tree index was created on the PAYMENT_DATE column to speed up operations such as:
Searching payments made on a specific date
Generating date-wise financial reports
Filtering payments within date ranges (e.g., monthly reports)

Why this index is useful

PAYMENT_DATE is frequently used in reporting and sorting, making it a strong candidate for indexing.
A B-tree index is ideal for:
High-cardinality columns (many unique values)
Range queries (e.g., BETWEEN, >, <)



```
SQL> CREATE INDEX RATING_I1 ON REVIEWS(RATING) REVERSE;

Index created.
```

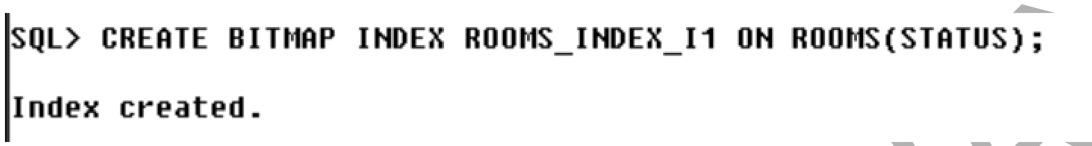
Purpose

A reverse key index was created on the RATING column to reduce index block contention during inserts.
When reverse indexes help

Reverse indexes reverse the byte order of index entries.

This is useful when:

Many inserts happen sequentially
Index block contention might occur (multiple sessions inserting at same key range)
Why used on RATING
Ratings often fall within a small numeric range (1–5 or 1–10).
Reverse indexing spreads inserts across multiple index blocks, improving performance.



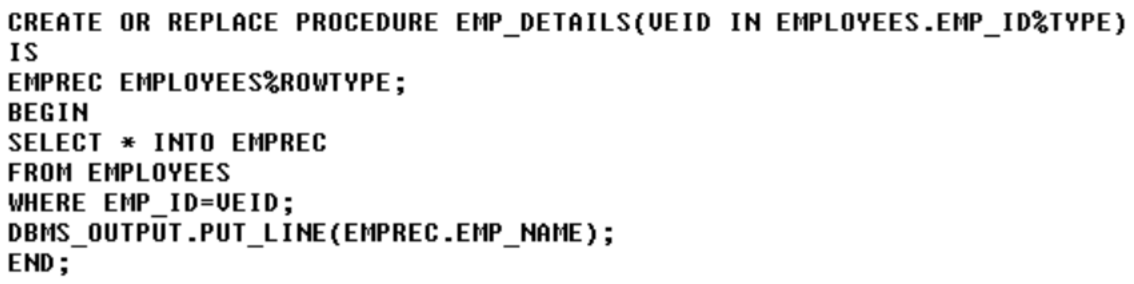
```
SQL> CREATE BITMAP INDEX ROOMS_INDEX_I1 ON ROOMS(STATUS);

Index created.
```

Purpose

A bitmap index was created for the STATUS column because it has low cardinality (few distinct values), typically:
AVAILABLE
OCCUPIED
MAINTENANCE
Bitmap indexes are extremely efficient for:
Columns with small number of unique values
Queries that perform filtering + aggregation
Reports that combine multiple conditions (e.g., available rooms by type)
Why bitmap index is ideal
Bitmap indexes use bitmaps instead of row pointers, allowing faster logical operations.
Great for OLAP-style queries like:
"How many rooms are available?"
"How many rooms are under maintenance?"

STORED PROCEDURENBG

PROCEDURE 1:EMP_DETAILS

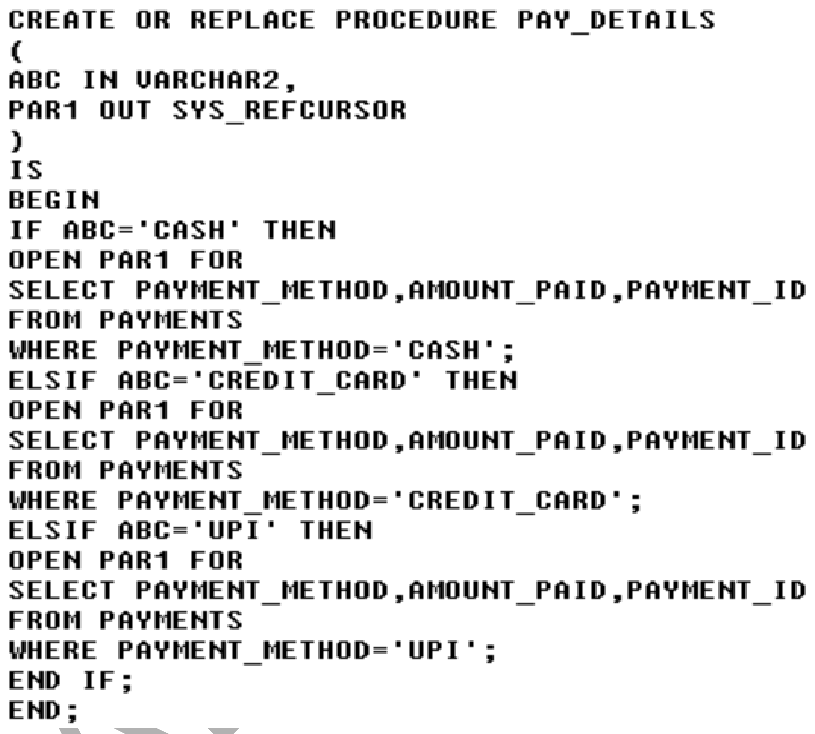


```
CREATE OR REPLACE PROCEDURE EMP_DETAILS(VEID IN EMPLOYEES.EMP_ID%TYPE)
IS
EMPREC EMPLOYEES%ROWTYPE;
BEGIN
SELECT * INTO EMPREC
FROM EMPLOYEES
WHERE EMP_ID=VEID;
DBMS_OUTPUT.PUT_LINE(EMPREC.EMP_NAME);
END;
```

Objective of the Procedure

The **EMP_DETAILS** procedure is designed to retrieve and display the name of an employee based on the employee ID provided as input.
It demonstrates how PL/SQL can be used to fetch complete row data using %ROWTYPE and process values from the result.

PROCEDURE 2: PAY_DETAILS



```
CREATE OR REPLACE PROCEDURE PAY_DETAILS
(
ABC IN VARCHAR2,
PAR1 OUT SYS_REFCURSOR
)
IS
BEGIN
IF ABC='CASH' THEN
OPEN PAR1 FOR
SELECT PAYMENT_METHOD,AMOUNT_PAID,PAYMENT_ID
FROM PAYMENTS
WHERE PAYMENT_METHOD='CASH';
ELSIF ABC='CREDIT_CARD' THEN
OPEN PAR1 FOR
SELECT PAYMENT_METHOD,AMOUNT_PAID,PAYMENT_ID
FROM PAYMENTS
WHERE PAYMENT_METHOD='CREDIT_CARD';
ELSIF ABC='UPI' THEN
OPEN PAR1 FOR
SELECT PAYMENT_METHOD,AMOUNT_PAID,PAYMENT_ID
FROM PAYMENTS
WHERE PAYMENT_METHOD='UPI';
END IF;
END;
```

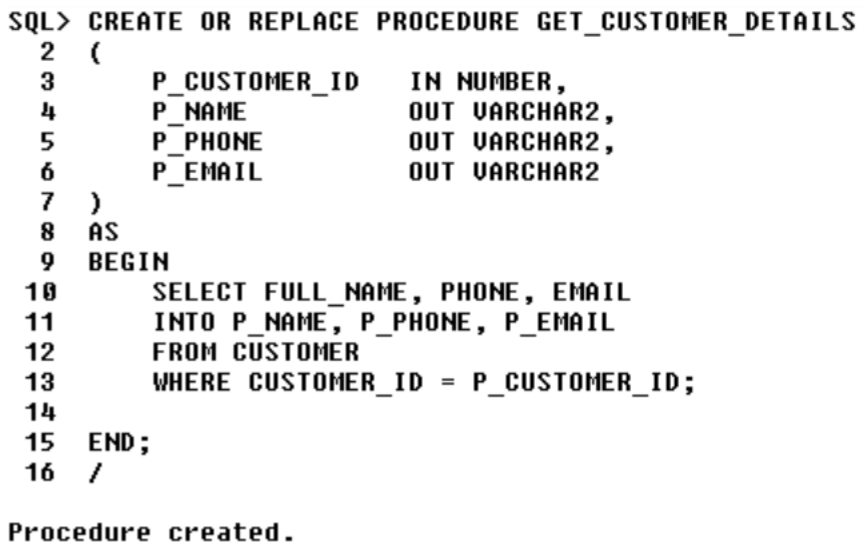
Name of the Procedure: PAY_DETAILS

Objective of the Procedure

The purpose of the PAY_DETAILS procedure is to retrieve payment-related information based on the payment method chosen by the user.

It uses a SYS_REFCURSOR to return dynamic query results for different payment modes such as CASH, CREDIT_CARD, or UPI.
This enhances flexibility in report generation and allows applications to request filtered data from a single stored procedure.

PROCEDURE 3: GET_CUSTOMER_DETAILS



```
SQL> CREATE OR REPLACE PROCEDURE GET_CUSTOMER_DETAILS
     (
         P_CUSTOMER_ID   IN NUMBER,
         P_NAME          OUT VARCHAR2,
         P_PHONE         OUT VARCHAR2,
         P_EMAIL         OUT VARCHAR2
     )
     AS
     BEGIN
         SELECT FULL_NAME, PHONE, EMAIL
         INTO P_NAME, P_PHONE, P_EMAIL
         FROM CUSTOMER
         WHERE CUSTOMER_ID = P_CUSTOMER_ID;

     END;
     /

Procedure created.
```

Objective

The purpose of the BOOKING_INFO procedure is to retrieve complete booking details based on the booking ID provided by the user. This procedure simplifies data retrieval and supports modular programming by encapsulating booking-related logic within a reusable PL/SQL block.

Input Parameter

P_BOOKING_ID (IN NUMBER):
Accepts the Booking ID for which details are required.

Output Parameters

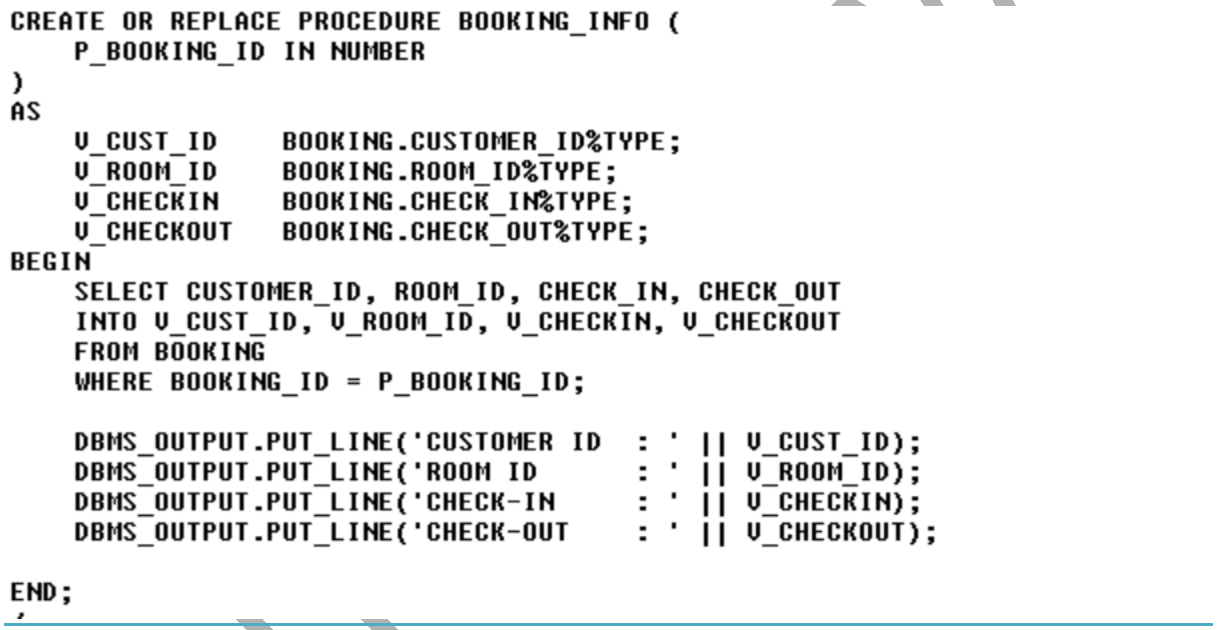
P_CUST_ID (OUT NUMBER): Returns the Customer ID associated with the booking.
P_ROOM_ID (OUT NUMBER): Returns the Room ID booked by the customer.
P_CHECKIN (OUT DATE): Returns the check-in date of the booking.

P_CHECKOUT (OUT DATE): Returns the check-out date of the booking.

Functionality

When executed, the procedure performs a SELECT query on the BOOKING table and fetches the corresponding booking details. These values are then returned through OUT parameters. This method ensures secure access to booking information and avoids repetitive SQL queries across the application.

PROCEDURE 4: BOOKING_INFO



```
CREATE OR REPLACE PROCEDURE BOOKING_INFO (
    P_BOOKING_ID IN NUMBER
)
AS
    V_CUST_ID     BOOKING.CUSTOMER_ID%TYPE;
    V_ROOM_ID     BOOKING.ROOM_ID%TYPE;
    V_CHECKIN     BOOKING.CHECK_IN%TYPE;
    V_CHECKOUT    BOOKING.CHECK_OUT%TYPE;
BEGIN
    SELECT CUSTOMER_ID, ROOM_ID, CHECK_IN, CHECK_OUT
    INTO V_CUST_ID, V_ROOM_ID, V_CHECKIN, V_CHECKOUT
    FROM BOOKING
    WHERE BOOKING_ID = P_BOOKING_ID;

    DBMS_OUTPUT.PUT_LINE('CUSTOMER ID   : ' || V_CUST_ID);
    DBMS_OUTPUT.PUT_LINE('ROOM ID       : ' || V_ROOM_ID);
    DBMS_OUTPUT.PUT_LINE('CHECK-IN      : ' || V_CHECKIN);
    DBMS_OUTPUT.PUT_LINE('CHECK-OUT     : ' || V_CHECKOUT);

END;
```

Objective

The GET_CUSTOMER_DETAILS procedure is designed to fetch personal details of a customer using their Customer ID. It enhances usability by returning customer information through OUT parameters, making the data easily accessible to other application modules.

Input Parameter

P_CUSTOMER_ID (IN NUMBER):
Accepts the Customer ID for retrieving customer information.

Output Parameters

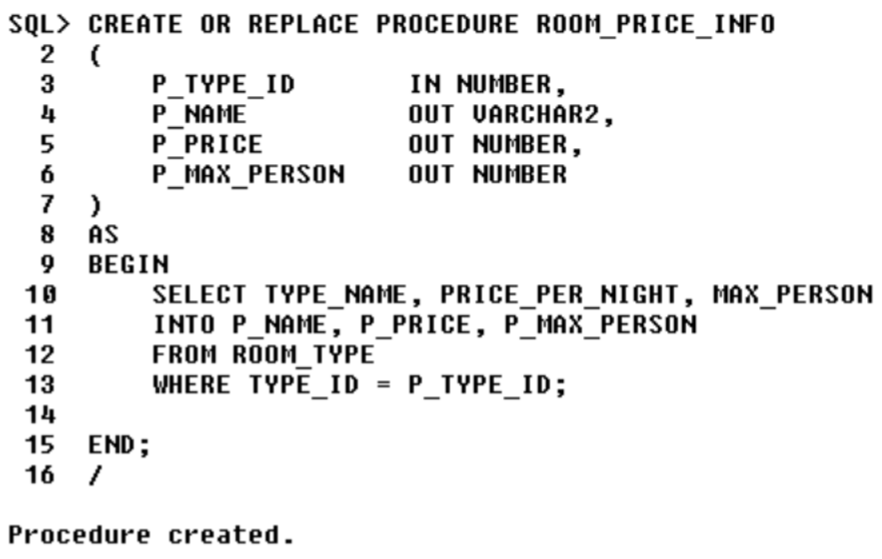
P_NAME (OUT VARCHAR2): Returns the full name of the customer.

P_PHONE (OUT VARCHAR2): Returns the registered phone number.
P_EMAIL (OUT VARCHAR2): Returns the email address of the customer.

Functionality

Upon receiving a valid Customer ID, the procedure queries the CUSTOMER table and extracts the customer's name, phone number, and email address. These retrieved values are passed back to the calling environment via OUT parameters.

PROCEDURE 5: ROOM_PRICE_INFO



```
SQL> CREATE OR REPLACE PROCEDURE ROOM_PRICE_INFO
     (
         P_TYPE_ID        IN NUMBER,
         P_NAME           OUT VARCHAR2,
         P_PRICE          OUT NUMBER,
         P_MAX_PERSON     OUT NUMBER
     )
     AS
     BEGIN
         SELECT TYPE_NAME, PRICE_PER_NIGHT, MAX_PERSON
         INTO P_NAME, P_PRICE, P_MAX_PERSON
         FROM ROOM_TYPE
         WHERE TYPE_ID = P_TYPE_ID;

     END;
     /

Procedure created.
```

Objective

The ROOM_PRICE_INFO procedure is used to retrieve room type details and pricing information based on a specific room type ID. It is highly beneficial for hotel management systems where room categories and prices must be dynamically displayed.

Input Parameter

P_TYPE_ID (IN NUMBER):
Accepts the Room Type ID for which details are required.

Output Parameters

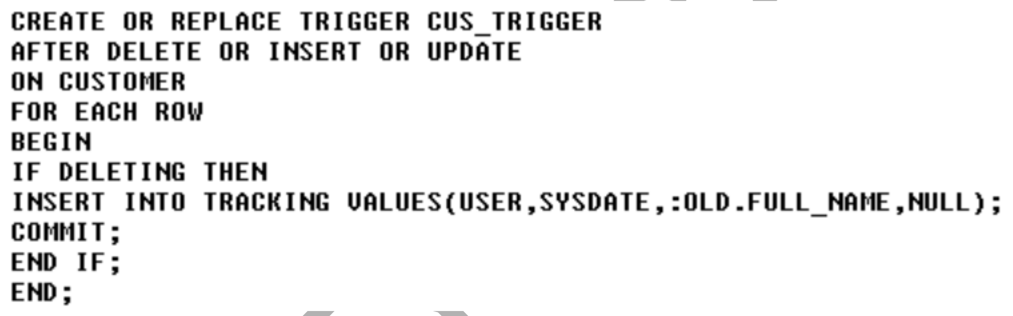
P_NAME (OUT VARCHAR2): Returns the name of the room type.
P_PRICE (OUT NUMBER): Returns the price per night of the room type.
P_MAX_PERSON (OUT NUMBER): Returns the maximum number of persons allowed.

Functionality

The procedure queries the ROOM_TYPE table and retrieves room type attributes, including price and maximum occupancy. The values are stored into OUT parameters for easy use in billing systems, room selection modules, and UI components.

DML LEVEL TRIGGERS:

1.Trigger Name: CUS_TRIGGER

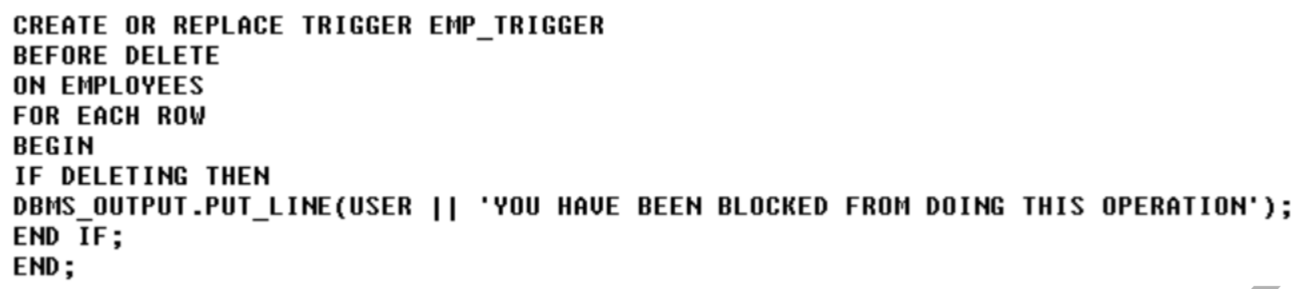


```
CREATE OR REPLACE TRIGGER CUS_TRIGGER
AFTER DELETE OR INSERT OR UPDATE
ON CUSTOMER
FOR EACH ROW
BEGIN
IF DELETING THEN
INSERT INTO TRACKING VALUES(USER,SYSDATE,:OLD.FULL_NAME,NULL);
COMMIT;
END IF;
END;
```

Benefits of This Trigger

1. Ensures automatic audit logging of customer deletions
2. Improves data accountability and traceability
3. Supports internal audits and incident investigation
4. Prevents loss of important customer identity during deletion
5. Helps maintain a secure and compliant database system

1.Trigger Name: EMP_TRIGGER



```
CREATE OR REPLACE TRIGGER EMP_TRIGGER
BEFORE DELETE
ON EMPLOYEES
FOR EACH ROW
BEGIN
IF DELETING THEN
DBMS_OUTPUT.PUT_LINE(USER || 'YOU HAVE BEEN BLOCKED FROM DOING THIS OPERATION');
END IF;
END;
```

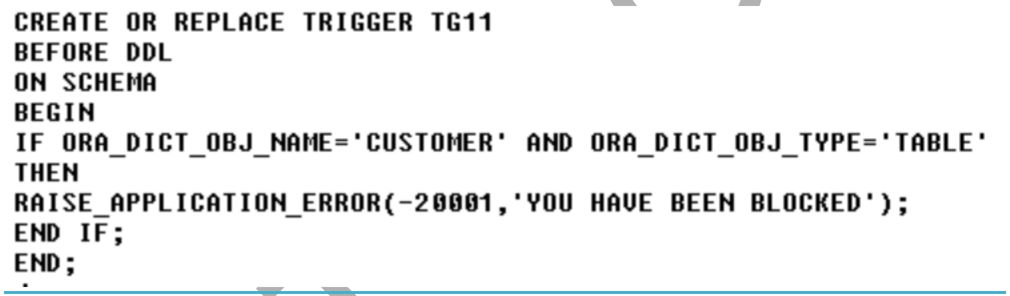
Purpose of the Trigger

The main purpose of this trigger is to prevent deletion of employee records and display a message to the user stating that the operation is not allowed.

This helps ensure:

1. Protection of critical employee data
2. Prevention of accidental or unauthorized deletions
3. Enforcement of business rules inside the database layer
4. Although the trigger prints a warning message, it does not explicitly raise an exception—however, since it's a BEFORE trigger, you can later enhance it to block deletion completely.

DML LEVEL TRIGGERS ON SCHEMA:



```
CREATE OR REPLACE TRIGGER TG11
BEFORE DDL
ON SCHEMA
BEGIN
IF ORA_DICT_OBJ_NAME='CUSTOMER' AND ORA_DICT_OBJ_TYPE='TABLE'
THEN
RAISE_APPLICATION_ERROR(-20001,'YOU HAVE BEEN BLOCKED');
END IF;
END;
```

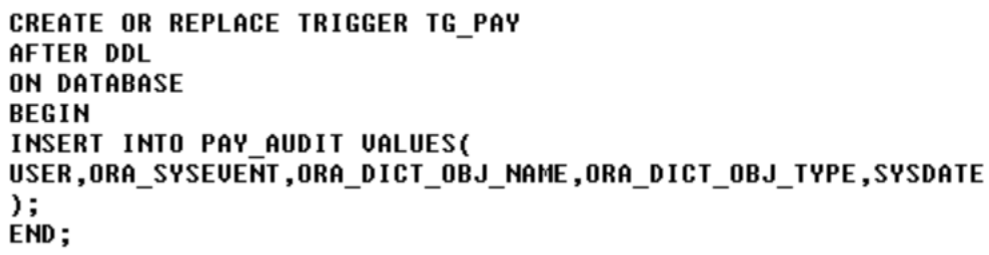
Purpose of the Trigger

This trigger is designed to protect the CUSTOMER table from unwanted structural changes.
It prevents any user from performing DDL operations such as:

DROP TABLE CUSTOMER
ALTER TABLE CUSTOMER
RENAME CUSTOMER
TRUNCATE CUSTOMER

If anyone tries to run a DDL command on the CUSTOMER table, the trigger throws a custom error and blocks the action.
This ensures the table remains safe from accidental or unauthorized modifications.

DML LEVEL TRIGGERS ON DATABSE:

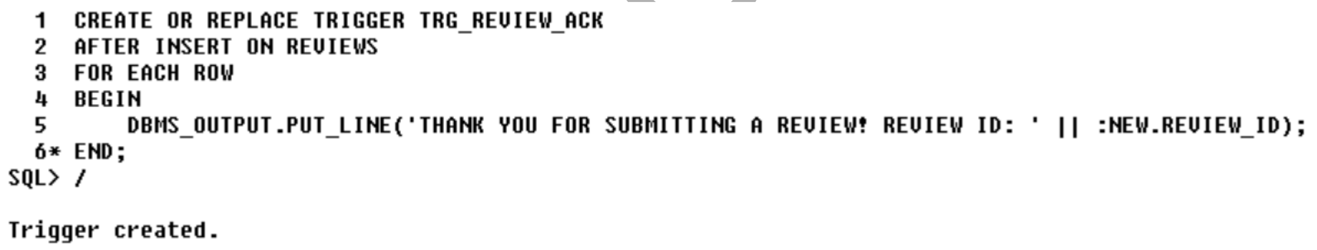


```
CREATE OR REPLACE TRIGGER TG_PAY
AFTER DDL
ON DATABASE
BEGIN
INSERT INTO PAY_AUDIT VALUES(
USER,ORA_SYSEVENT,ORA_DICT_OBJ_NAME,ORA_DICT_OBJ_TYPE,SYSDATE
);
END;
```

Purpose of the Trigger:

The purpose of the TG_PAY trigger is to capture all DDL operations happening in the database and store them in the PAY_AUDIT table.
This creates a complete audit trail of database structure changes, including:
Who performed the operation
What DDL event occurred
Which object was affected
What type of object it was
When the event occurred
This is extremely valuable for security, monitoring, compliance, and troubleshooting.

DML LEVEL TRIGGERS ON DATABSE:



```
CREATE OR REPLACE TRIGGER TRG_REVIEW_ACK
AFTER INSERT ON REVIEWS
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('THANK YOU FOR SUBMITTING A REVIEW! REVIEW ID: ' || :NEW.REVIEW_ID);
END;
SQL> /

Trigger created.
```

Purpose of the Trigger:

purpose of this trigger is to **acknowledge and notify** the user whenever a new review is submitted. It displays a message containing the **Review ID** of the newly inserted review using DBMS_OUTPUT.This enhances user interaction and provides confirmation that the review was successfully recorded.

Conclusion

The PL/SQL-based Hotel Management System implemented in this project successfully demonstrates the use of advanced database concepts required for building a secure, reliable, and efficient backend system. Throughout the development process, various database objects—including tables, views, indexes, stored procedures, functions, and multiple types of triggers—were designed and implemented to ensure structured data handling, automation of repetitive tasks, enforcement of business rules, and maintenance of data integrity.

The project incorporates essential features such as audit logging, deletion control, DDL monitoring, employee and customer management, payment tracking, room management, and service usage analysis. The strategic use of **row-level and schema-level triggers**, along with **DDL auditing mechanisms**, strengthens the system by preventing unauthorized modifications and ensuring complete traceability of critical operations. Additionally, the use of **SYS_REFCURSOR** in stored procedures offers flexible and dynamic data retrieval, making the system adaptable for integration with real-world applications.

Overall, this project not only fulfils the requirements of a functional hotel database system but also showcases strong understanding and practical application of PL/SQL programming techniques. It reflects how database programming can be used to automate operations, enforce business logic at the database layer, and build secure enterprise-grade solutions. The concepts demonstrated here can be extended to larger, more complex systems, making this project a valuable foundation for real-time database development.

Future Scope

The current PL/SQL-based Hotel Management System provides a stable and functional database foundation; however, it can be significantly expanded to meet the needs of modern, large-scale hospitality operations. Several enhancements can be incorporated in the future to increase usability, performance, and integration capability.

Integration with Front-End Applications:

The system can be connected to web or mobile applications using technologies such as Java, Python, .NET, or PHP. This would allow real-time booking, customer management, and payment processing through a user-friendly interface.

Enhanced Security and Role Management:

Implementing advanced security features such as role-based access control (RBAC), encryption of sensitive data, and multi-level authentication can offer stronger protection for customer and financial information.

Advanced Reporting and Analytics:

Business intelligence modules and analytic dashboards can be integrated to generate insights on occupancy rates, revenue trends, customer behaviour, and service usage, supporting better managerial decision-making.

Automation Using Advanced PL/SQL Features

Future enhancements may include automated billing generation, room assignment logic, dynamic pricing algorithms, and automated cancellation or reminder notifications.

Integration with Third-Party Services

The system can be extended to interact with external APIs such as online payment gateways, SMS/email notification systems, and third-party hotel reservation platforms like Booking.com or Airbnb.

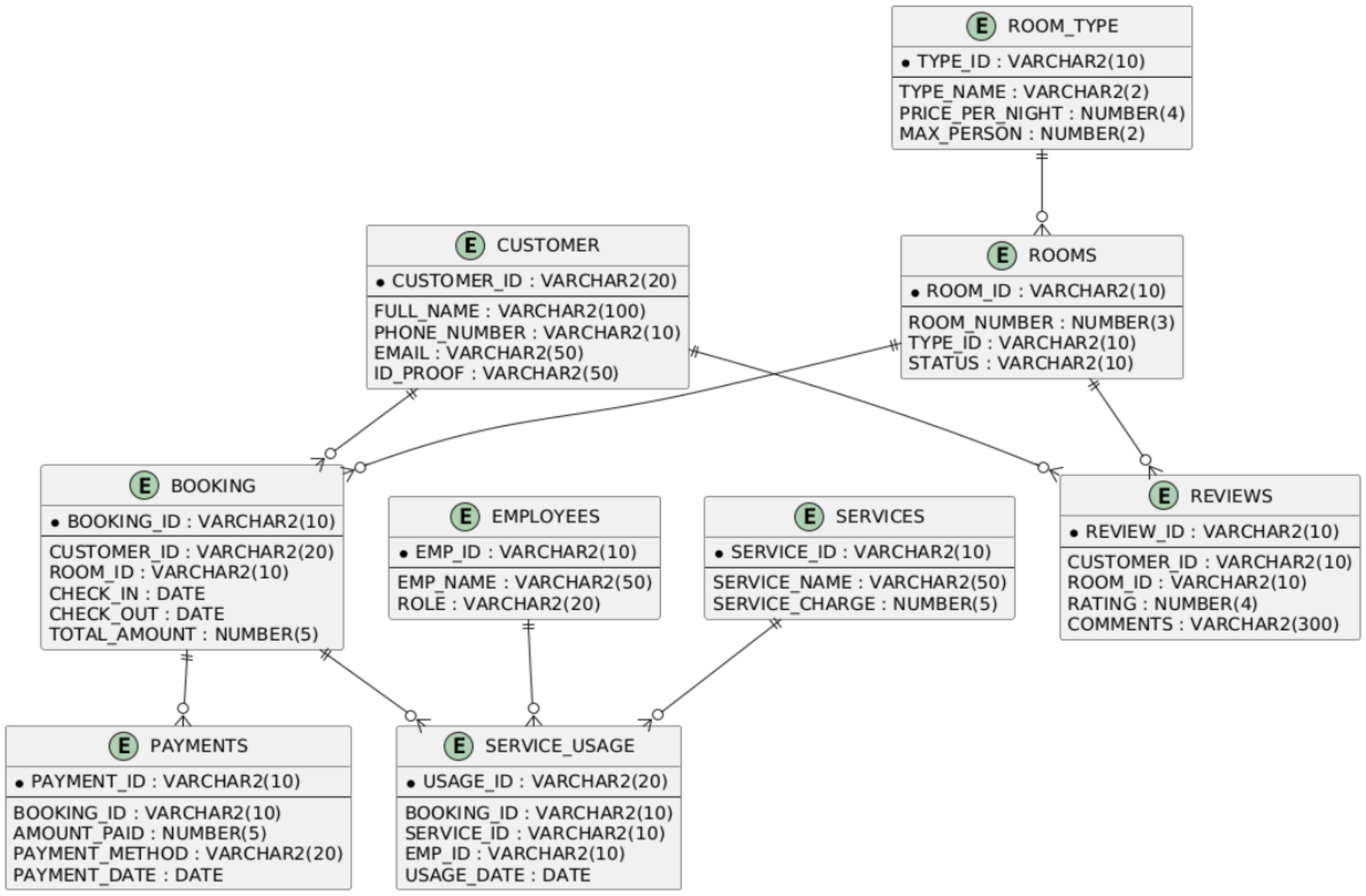
Cloud Deployment and Scalability

Migrating the database to cloud platforms such as Oracle Cloud, AWS RDS, or Azure SQL can improve scalability, availability, and disaster recovery.
Machine Learning for Recommendations
Customer behaviour analysis using ML models can help recommend rooms, predict peak seasons, forecast occupancy, and personalize service offerings.

Multi-Hotel Support

The existing system can evolve into a centralized solution capable of managing multiple hotel branches through one unified database.

Schema Diagram:



ROOM_TYPE
TYPE_ID : VARCHAR2(10)
TYPE_NAME : VARCHAR2(2)
PRICE_PER_NIGHT : NUMBER(4)
MAX_PERSON : NUMBER(2)
CUSTOMER
CUSTOMER_ID : VARCHAR2(20)
FULL_NAME : VARCHAR2(100)
PHONE_NUMBER : VARCHAR2(10)
EMAIL : VARCHAR2(50)
ID_PROOF : VARCHAR2(50)
ROOMS
ROOM_ID : VARCHAR2(10)
ROOM_NUMBER : NUMBER(3)
TYPE_ID : VARCHAR2(10)
STATUS : VARCHAR2(10)
BOOKING
BOOKING_ID : VARCHAR2(10)
CUSTOMER_ID : VARCHAR2(20)
ROOM_ID : VARCHAR2(10)
CHECK_IN : DATE
CHECK_OUT : DATE
TOTAL_AMOUNT : NUMBER(5)
EMPLOYEES
EMP_ID : VARCHAR2(10)
EMP_NAME : VARCHAR2(50)
ROLE : VARCHAR2(20)
SERVICES
SERVICE_ID : VARCHAR2(10)
SERVICE_NAME : VARCHAR2(50)
SERVICE_CHARGE : NUMBER(5)
REVIEWS
REVIEW_ID : VARCHAR2(10)
CUSTOMER_ID : VARCHAR2(10)
ROOM_ID : VARCHAR2(10)
RATING : NUMBER(4)
COMMENTS : VARCHAR2(300)
PAYMENTS
PAYMENT_ID : VARCHAR2(10)
BOOKING_ID : VARCHAR2(10)
AMOUNT_PAID : NUMBER(5)
PAYMENT_METHOD : VARCHAR2(20)
PAYMENT_DATE : DATE
SERVICE_USAGE
USAGE_ID : VARCHAR2(20)
BOOKING_ID : VARCHAR2(10)
SERVICE_ID : VARCHAR2(10)
EMP_ID : VARCHAR2(10)
USAGE_DATE : DATE