CST2355 – Database Systems Group Lab Assignment 2

# **Assignment Overview:**

Group Lab Assignment 2 will give you a chance to work in a team of 2 or 3 students to create a database application consisting of a **Microsoft Access application that uses a separate Oracle 19c database to store application data in linked tables** rather than in the Microsoft Access file itself.

You will add to three of your original tables from Group Lab Assignment 1 by creating a set of ‘views’ in Oracle that when queried have all the fields from your original tables. You will be doing this step for at least two of your original tables. This means that your previous Access application should work if the linked tables are updated to point to the new Oracle Views.

You will also be creating a set of triggers (and stored procedures if desired) to allow INSERTS, UPDATES, and DELETES to the views so that if a value of a field gets changed, a history of those changes is kept in a new set of tables to store the multi-valued fields. We will be putting timestamps on the updates, so that we have a history of when each value was valid.

# **Details of the Required Historical Data Framework Requirements:**

Let’s start with an example. If you have an original table with two multi-valued fields (e,g, person – containing an ID (primary key), and a FIRSTNAME and LASTNAME fields). That table could be replaced with a view (e.g., person\_view) that is defined as a SELECT statement from a join across a set of new tables. The table would get replaced with a table of the same name that has only the primary key (and any other single-valued fields that might have originally been in the table). We end up with: the original table (with just the key and the single-valued fields); a new view; a new table for each multi-valued field; and an association table that links the new table to each of the multi-valued fields. We could use a view defined as shown directly below. (using PERSON, FNAMES, LNAMES as the underlying tables, and the two association tables as PERSON\_FNAME and PERSON\_LNAME)

Example view (you need to REALLY understand this code….):

CREATE VIEW PERSON\_VIEW AS

SELECT PERSON.ID, FNAMES.FIRSTNAME, LNAMES.LASTNAME

FROM PERSON

LEFT JOIN PERSON\_FNAME

ON PERSON.ID = PERSON\_FNAME.PID

LEFT JOIN FNAMES

ON PERSON\_FNAME.FID = FNAMES.FID

LEFT JOIN PERSON\_LNAME

ON PERSON.ID = PERSON\_LNAME.PID

LEFT JOIN LNAMES

ON PERSON\_LNAME.LID = LNAMES.LID

WHERE

(person\_fname.ENDTIME is NULL)

AND

(person\_lname.ENDTIME is NULL);

Each multi-valued field was replaced with a new association table to store the one-to-many mappings, and a new table to hold the values as they are updated. In addition, the association table should have an additional field containing the “STARTTIME” (the timestamp from when the value was inserted or given the linked value) and an additional field containing the “ENDTIME” (the timestamp when the value was changed to a new value or DELETED.

OPTIONAL: You can add another field to the association tables hold a text field to store ‘Notes’ about the change – allow this field to be default NULL.

### Detailed Explanation with Sample Query for Historical Data Framework

#### Overview

To manage historical data for multi-valued fields in a table, we create separate tables to store the values and their changes over time. This approach involves:

1. Original table with single-valued fields.
2. New tables for multi-valued fields.
3. Association tables to link the original table with the new tables.
4. Views to present the combined data.
5. Triggers to handle INSERT, UPDATE, and DELETE operations and maintain historical records.

### Step-by-Step Example

#### Original Table

Let's start with an example of an original table PERSON with multi-valued fields FIRSTNAME and LASTNAME.

sql

Copy code

CREATE TABLE PERSON (

ID INT PRIMARY KEY,

DATE\_OF\_BIRTH DATE

);

#### New Tables for Multi-Valued Fields

We create new tables to store FIRSTNAME and LASTNAME values.

sql

Copy code

CREATE TABLE FNAMES (

FID INT PRIMARY KEY,

FIRSTNAME VARCHAR(50) NOT NULL

);

CREATE TABLE LNAMES (

LID INT PRIMARY KEY,

LASTNAME VARCHAR(50) NOT NULL

);

#### Association Tables

We create association tables to link PERSON with FNAMES and LNAMES, including STARTTIME and ENDTIME fields.

sql

Copy code

CREATE TABLE PERSON\_FNAME (

PID INT,

FID INT,

STARTTIME TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

ENDTIME TIMESTAMP,

PRIMARY KEY (PID, FID, STARTTIME),

FOREIGN KEY (PID) REFERENCES PERSON(ID),

FOREIGN KEY (FID) REFERENCES FNAMES(FID)

);

CREATE TABLE PERSON\_LNAME (

PID INT,

LID INT,

STARTTIME TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

ENDTIME TIMESTAMP,

PRIMARY KEY (PID, LID, STARTTIME),

FOREIGN KEY (PID) REFERENCES PERSON(ID),

FOREIGN KEY (LID) REFERENCES LNAMES(LID)

);

#### View to Combine Data

We create a view to present the combined data from PERSON, FNAMES, and LNAMES.

sql

Copy code

CREATE VIEW PERSON\_VIEW AS

SELECT PERSON.ID, FNAMES.FIRSTNAME, LNAMES.LASTNAME, PERSON.DATE\_OF\_BIRTH

FROM PERSON

LEFT JOIN PERSON\_FNAME

ON PERSON.ID = PERSON\_FNAME.PID

LEFT JOIN FNAMES

ON PERSON\_FNAME.FID = FNAMES.FID

LEFT JOIN PERSON\_LNAME

ON PERSON.ID = PERSON\_LNAME.PID

LEFT JOIN LNAMES

ON PERSON\_LNAME.LID = LNAMES.LID

WHERE

PERSON\_FNAME.ENDTIME IS NULL

AND

PERSON\_LNAME.ENDTIME IS NULL;

#### Triggers for Maintaining Historical Data

1. **INSERT Trigger:** Handles inserting new records into PERSON\_VIEW.

sql

Copy code

CREATE OR REPLACE TRIGGER PERSON\_INSERT

INSTEAD OF INSERT ON PERSON\_VIEW

FOR EACH ROW

BEGIN

INSERT INTO PERSON (ID, DATE\_OF\_BIRTH) VALUES (:NEW.ID, :NEW.DATE\_OF\_BIRTH);

INSERT INTO FNAMES (FID, FIRSTNAME) VALUES (:NEW.ID, :NEW.FIRSTNAME);

INSERT INTO LNAMES (LID, LASTNAME) VALUES (:NEW.ID, :NEW.LASTNAME);

INSERT INTO PERSON\_FNAME (PID, FID) VALUES (:NEW.ID, :NEW.ID);

INSERT INTO PERSON\_LNAME (PID, LID) VALUES (:NEW.ID, :NEW.ID);

END;

/

1. **UPDATE Trigger:** Handles updating records in PERSON\_VIEW.

sql

Copy code

CREATE OR REPLACE TRIGGER PERSON\_UPDATE

INSTEAD OF UPDATE ON PERSON\_VIEW

FOR EACH ROW

BEGIN

UPDATE PERSON

SET DATE\_OF\_BIRTH = :NEW.DATE\_OF\_BIRTH

WHERE ID = :OLD.ID;

IF :OLD.FIRSTNAME != :NEW.FIRSTNAME THEN

UPDATE PERSON\_FNAME

SET ENDTIME = CURRENT\_TIMESTAMP

WHERE PID = :OLD.ID AND ENDTIME IS NULL;

INSERT INTO FNAMES (FID, FIRSTNAME) VALUES (:NEW.ID, :NEW.FIRSTNAME);

INSERT INTO PERSON\_FNAME (PID, FID) VALUES (:NEW.ID, :NEW.ID);

END IF;

IF :OLD.LASTNAME != :NEW.LASTNAME THEN

UPDATE PERSON\_LNAME

SET ENDTIME = CURRENT\_TIMESTAMP

WHERE PID = :OLD.ID AND ENDTIME IS NULL;

INSERT INTO LNAMES (LID, LASTNAME) VALUES (:NEW.ID, :NEW.LASTNAME);

INSERT INTO PERSON\_LNAME (PID, LID) VALUES (:NEW.ID, :NEW.ID);

END IF;

END;

/

1. **DELETE Trigger:** Handles deleting records from PERSON\_VIEW.

sql

Copy code

CREATE OR REPLACE TRIGGER PERSON\_DELETE

INSTEAD OF DELETE ON PERSON\_VIEW

FOR EACH ROW

BEGIN

DELETE FROM PERSON WHERE ID = :OLD.ID;

UPDATE PERSON\_FNAME SET ENDTIME = CURRENT\_TIMESTAMP WHERE PID = :OLD.ID AND ENDTIME IS NULL;

UPDATE PERSON\_LNAME SET ENDTIME = CURRENT\_TIMESTAMP WHERE PID = :OLD.ID AND ENDTIME IS NULL;

END;

/

### Sample Queries

1. **Insert a New Record:**

sql

Copy code

INSERT INTO PERSON\_VIEW (ID, FIRSTNAME, LASTNAME, DATE\_OF\_BIRTH)

VALUES (1, 'John', 'Doe', '1980-01-01');

1. **Update an Existing Record:**

sql

Copy code

UPDATE PERSON\_VIEW

SET FIRSTNAME = 'Jonathan', LASTNAME = 'Smith', DATE\_OF\_BIRTH = '1980-02-01'

WHERE ID = 1;

1. **Delete a Record:**

sql

Copy code

DELETE FROM PERSON\_VIEW WHERE ID = 1;

### Summary

In this example, the original PERSON table with multi-valued fields FIRSTNAME and LASTNAME is split into multiple tables to manage these fields and their historical data. Association tables link the new tables to the original table, and triggers ensure that changes to the data are recorded with timestamps. A view combines the data for easy access, ensuring that only the current values are visible. This approach allows for comprehensive tracking of historical data while maintaining data integrity and accessibility.

# **New Requirements to Extend Group Assignment 1 to Group Assignment 2:**

1. Introduce at least one multi-valued historical data field in a table that is part of each of the following 3 types of relationships: is-a, contains, is-related-to. This will mean that you must update the forms that use those 3 tables to use your new view instead of each of the original tables.
2. For each of the multi-valued field examples get your original group assignment 1 application still able to work using the new VIEWs. You will need to build a set of INSTEAD OF triggers for each of these new views to handle the SELECT, INSERT, UPDATE, and DELETE commands sent from your application to the database.
3. Update your forms for editing each of the three original tables to have buttons or some other forms so that you can look at the historical data for each field in a new pop-up form specific to that field (with start and end timestamps). You could do this by having a button such as “View Name History” which shows a form containing the name history data.

To meet the new requirements for extending your project, let's go through each step in more detail with comprehensive explanations and examples, including the SQL queries and steps needed to achieve the goals.

### Step 1: Introducing Multi-Valued Historical Data Fields

We'll work with the following relationships: is-a, contains, and is-related-to. For each relationship type, we’ll add a multi-valued historical data field.

#### Example Scenario:

We are working on a hotel booking system.

1. **is-a**: Employee is-a Person
2. **contains**: Room contains Amenities
3. **is-related-to**: Booking is related to Guests

#### 1.1 is-a Relationship: Employee is-a Person

**Person Table:**

sql

Copy code

CREATE TABLE Person (

PersonID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DateOfBirth DATE

);

**Employee Table:**

sql

Copy code

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY,

PersonID INT,

Position VARCHAR(50),

StartDate DATE,

FOREIGN KEY (PersonID) REFERENCES Person(PersonID)

);

**FirstName History Table:**

sql

Copy code

CREATE TABLE FirstNameHistory (

HistoryID INT PRIMARY KEY,

PersonID INT,

FirstName VARCHAR(50),

StartTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

EndTime TIMESTAMP,

FOREIGN KEY (PersonID) REFERENCES Person(PersonID)

);

**LastName History Table:**

sql

Copy code

CREATE TABLE LastNameHistory (

HistoryID INT PRIMARY KEY,

PersonID INT,

LastName VARCHAR(50),

StartTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

EndTime TIMESTAMP,

FOREIGN KEY (PersonID) REFERENCES Person(PersonID)

);

**View for Person with FirstName and LastName History:**

sql

Copy code

CREATE VIEW PersonView AS

SELECT p.PersonID, fn.FirstName, ln.LastName, p.DateOfBirth

FROM Person p

LEFT JOIN FirstNameHistory fn ON p.PersonID = fn.PersonID AND fn.EndTime IS NULL

LEFT JOIN LastNameHistory ln ON p.PersonID = ln.PersonID AND ln.EndTime IS NULL;

**Triggers for PersonView:**

**Insert Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER PersonInsert

INSTEAD OF INSERT ON PersonView

FOR EACH ROW

BEGIN

INSERT INTO Person (PersonID, DateOfBirth) VALUES (:NEW.PersonID, :NEW.DateOfBirth);

INSERT INTO FirstNameHistory (PersonID, FirstName) VALUES (:NEW.PersonID, :NEW.FirstName);

INSERT INTO LastNameHistory (PersonID, LastName) VALUES (:NEW.PersonID, :NEW.LastName);

END;

/

**Update Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER PersonUpdate

INSTEAD OF UPDATE ON PersonView

FOR EACH ROW

BEGIN

UPDATE Person SET DateOfBirth = :NEW.DateOfBirth WHERE PersonID = :OLD.PersonID;

IF :OLD.FirstName != :NEW.FirstName THEN

UPDATE FirstNameHistory SET EndTime = CURRENT\_TIMESTAMP WHERE PersonID = :OLD.PersonID AND EndTime IS NULL;

INSERT INTO FirstNameHistory (PersonID, FirstName) VALUES (:NEW.PersonID, :NEW.FirstName);

END IF;

IF :OLD.LastName != :NEW.LastName THEN

UPDATE LastNameHistory SET EndTime = CURRENT\_TIMESTAMP WHERE PersonID = :OLD.PersonID AND EndTime IS NULL;

INSERT INTO LastNameHistory (PersonID, LastName) VALUES (:NEW.PersonID, :NEW.LastName);

END IF;

END;

/

**Delete Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER PersonDelete

INSTEAD OF DELETE ON PersonView

FOR EACH ROW

BEGIN

DELETE FROM Person WHERE PersonID = :OLD.PersonID;

UPDATE FirstNameHistory SET EndTime = CURRENT\_TIMESTAMP WHERE PersonID = :OLD.PersonID AND EndTime IS NULL;

UPDATE LastNameHistory SET EndTime = CURRENT\_TIMESTAMP WHERE PersonID = :OLD.PersonID AND EndTime IS NULL;

END;

/

### 1.2 contains Relationship: Room contains Amenities

**Room Table:**

sql

Copy code

CREATE TABLE Room (

RoomID INT PRIMARY KEY,

RoomNumber VARCHAR(10) UNIQUE,

RoomType VARCHAR(50),

PricePerNight DECIMAL(10, 2)

);

**Amenity Table:**

sql

Copy code

CREATE TABLE Amenity (

AmenityID INT PRIMARY KEY,

AmenityName VARCHAR(50)

);

**RoomAmenity Association Table:**

sql

Copy code

CREATE TABLE RoomAmenity (

RoomID INT,

AmenityID INT,

StartTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

EndTime TIMESTAMP,

PRIMARY KEY (RoomID, AmenityID, StartTime),

FOREIGN KEY (RoomID) REFERENCES Room(RoomID),

FOREIGN KEY (AmenityID) REFERENCES Amenity(AmenityID)

);

**View for Room with Amenities:**

sql

Copy code

CREATE VIEW RoomView AS

SELECT r.RoomID, r.RoomNumber, r.RoomType, r.PricePerNight, a.AmenityName

FROM Room r

LEFT JOIN RoomAmenity ra ON r.RoomID = ra.RoomID AND ra.EndTime IS NULL

LEFT JOIN Amenity a ON ra.AmenityID = a.AmenityID;

**Triggers for RoomView:**

**Insert Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER RoomInsert

INSTEAD OF INSERT ON RoomView

FOR EACH ROW

BEGIN

INSERT INTO Room (RoomID, RoomNumber, RoomType, PricePerNight) VALUES (:NEW.RoomID, :NEW.RoomNumber, :NEW.RoomType, :NEW.PricePerNight);

INSERT INTO Amenity (AmenityID, AmenityName) VALUES (:NEW.RoomID, :NEW.AmenityName);

INSERT INTO RoomAmenity (RoomID, AmenityID) VALUES (:NEW.RoomID, :NEW.RoomID);

END;

/

**Update Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER RoomUpdate

INSTEAD OF UPDATE ON RoomView

FOR EACH ROW

BEGIN

UPDATE Room SET RoomNumber = :NEW.RoomNumber, RoomType = :NEW.RoomType, PricePerNight = :NEW.PricePerNight WHERE RoomID = :OLD.RoomID;

IF :OLD.AmenityName != :NEW.AmenityName THEN

UPDATE RoomAmenity SET EndTime = CURRENT\_TIMESTAMP WHERE RoomID = :OLD.RoomID AND EndTime IS NULL;

INSERT INTO Amenity (AmenityID, AmenityName) VALUES (:NEW.RoomID, :NEW.AmenityName);

INSERT INTO RoomAmenity (RoomID, AmenityID) VALUES (:NEW.RoomID, :NEW.RoomID);

END IF;

END;

/

**Delete Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER RoomDelete

INSTEAD OF DELETE ON RoomView

FOR EACH ROW

BEGIN

DELETE FROM Room WHERE RoomID = :OLD.RoomID;

UPDATE RoomAmenity SET EndTime = CURRENT\_TIMESTAMP WHERE RoomID = :OLD.RoomID AND EndTime IS NULL;

END;

/

### 1.3 is-related-to Relationship: Booking is related to Guests

**Booking Table:**

sql

Copy code

CREATE TABLE Booking (

BookingID INT PRIMARY KEY,

GuestID INT,

RoomID INT,

CheckInDate DATE,

CheckOutDate DATE,

FOREIGN KEY (GuestID) REFERENCES Guests(GuestID),

FOREIGN KEY (RoomID) REFERENCES Room(RoomID)

);

**Guest Table:**

sql

Copy code

CREATE TABLE Guest (

GuestID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100) UNIQUE,

PhoneNumber VARCHAR(15)

);

**BookingGuest Association Table:**

sql

Copy code

CREATE TABLE BookingGuest (

BookingID INT,

GuestID INT,

StartTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

EndTime TIMESTAMP,

PRIMARY KEY (BookingID, GuestID, StartTime),

FOREIGN KEY (BookingID) REFERENCES Booking(BookingID),

FOREIGN KEY (GuestID) REFERENCES Guest(GuestID)

);

**View for Booking with Guests:**

sql

Copy code

CREATE VIEW BookingView AS

SELECT b.BookingID, g.FirstName, g.LastName, g.Email, g.PhoneNumber, b.CheckInDate, b.CheckOutDate

FROM Booking b

LEFT JOIN BookingGuest bg ON b.BookingID = bg.BookingID AND bg.EndTime IS NULL

LEFT JOIN Guest g ON bg.GuestID = g.GuestID;

**Triggers for BookingView:**

**Insert Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER BookingInsert

INSTEAD OF INSERT ON BookingView

FOR EACH ROW

BEGIN

INSERT INTO Booking (BookingID, GuestID, RoomID, CheckInDate, CheckOutDate) VALUES (:NEW.BookingID, :NEW.GuestID, :NEW.RoomID, :NEW.CheckInDate, :NEW.CheckOutDate);

INSERT INTO BookingGuest (BookingID, GuestID) VALUES (:NEW.BookingID, :NEW.GuestID);

END;

/

**Update Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER BookingUpdate

INSTEAD OF UPDATE ON BookingView

FOR EACH ROW

BEGIN

UPDATE Booking SET GuestID = :NEW.GuestID, RoomID = :NEW.RoomID, CheckInDate = :NEW.CheckInDate, CheckOutDate = :NEW.CheckOutDate WHERE BookingID = :OLD.BookingID;

IF :OLD.GuestID != :NEW.GuestID THEN

UPDATE BookingGuest SET EndTime = CURRENT\_TIMESTAMP WHERE BookingID = :OLD.BookingID AND EndTime IS NULL;

INSERT INTO BookingGuest (BookingID, GuestID) VALUES (:NEW.BookingID, :NEW.GuestID);

END IF;

END;

/

**Delete Trigger:**

sql

Copy code

CREATE OR REPLACE TRIGGER BookingDelete

INSTEAD OF DELETE ON BookingView

FOR EACH ROW

BEGIN

DELETE FROM Booking WHERE BookingID = :OLD.BookingID;

UPDATE BookingGuest SET EndTime = CURRENT\_TIMESTAMP WHERE BookingID = :OLD.BookingID AND EndTime IS NULL;

END;

/

### Step 2: Building Views and Triggers

We have already created the views and triggers in the above examples. These views and triggers ensure that our application works with the historical data framework, keeping track of changes and maintaining data integrity.

### Step 3: Updating Forms in MS Access

In MS Access, update the forms to use the new views and add buttons or other form elements to show historical data.

**Steps:**

1. **Linking Views:**
   * Create linked tables in MS Access for PersonView, RoomView, and BookingView using ODBC connections to the Oracle database.
2. **Updating Forms:**
   * Update the data source of your existing forms to use the linked views instead of the original tables.
   * Ensure that all fields in the forms are correctly mapped to the columns in the views.
3. **Adding Historical Data Buttons:**
   * Add buttons to the forms to open new pop-up forms showing the historical data.
   * Example button for viewing name history in Person form:

vba

Copy code

Private Sub btnViewNameHistory\_Click()

DoCmd.OpenForm "frmNameHistory", , , "PersonID = " & Me.PersonID

End Sub

* + Create new forms (frmNameHistory, frmAmenityHistory, frmBookingHistory) to display historical data using queries that select from the history tables.

**Example Query for Name History:**

sql

Copy code

SELECT FirstName, StartTime, EndTime

FROM FirstNameHistory

WHERE PersonID = [Forms]![YourFormName]![PersonID];

### Summary

In this detailed elaboration, we have covered:

1. **Introducing multi-valued historical data fields:**
   * Created new tables for historical data.
   * Set up views combining current data with historical data.
   * Implemented triggers to handle changes and maintain historical records.
2. **Ensuring application compatibility with new views:**
   * Updated existing forms to use the new views.
   * Created triggers to manage INSERT, UPDATE, and DELETE operations.
3. **Adding functionality to view historical data:**
   * Added buttons in forms to open new pop-up forms.
   * Created new forms and queries to display historical data.

By following these steps, you can extend your application to manage historical data effectively while maintaining the integrity and functionality of your original application.

# **Steps (Similar to Group Assignment 1):**

1. The first step is to confirm the membership of your group. An update to your one-page proposal describing your assignment topic **is required**. It should clearly state your group membership details and who is responsible for which components of the solution and describe the data in some detail. It will be edited to become the application description and usage instruction documents in the final submission of the assignment.
   1. Group Membership: **YES** you can change group members, but this will need to be discussed with your lab professor in advance.
      1. All group members MUST be in the same lab section.
      2. Discuss topic choice in the lab and then after agreeing to work together, all members enroll in the same group # in Brightspace.
   2. List of topic suggestions: (as before)
      1. Vinyl recording collection management.
      2. Spotify streaming data (XML download for individuals, import data, …)
      3. Knitting supplies management
      4. Tool rental management
      5. Doll collection management
      6. Guitar collection management
      7. Appointment scheduling for piano tuners
      8. Appointment scheduling for appliance repair
      9. Safety inspection scheduling in apartment buildings
      10. Any interesting example that has the required types of relationships….
2. Proposal Template: (1 or more pages)

Section 1: Introduction:

* Describe the topic; for example, an application to manage a personal vinyl recording collection. Some details of the information to be stored and how ‘hierarchies’, ‘is-a’, ‘contains’, and ‘related-to’ relationships are dealt with in the data. Try to limit the number of tables but cover all the required relationship types.

e.g. For a vinyl recording management application: a list of artists, each artist is-a ‘type’ of artist (e.g., songwriter, musician, singer, …), artists are related-to vinyl albums, each vinyl album contains recordings, individual musicians, singers, etc. are related-to each recording. A song can be recorded by several musicians on different vinyl recordings, etc. You will need to build a precise model later, but at least give the professor an idea of what you are planning to store and manage,

Section 2: Group Membership and Tasks:

* List of members and email addresses.
* Table showing who will do which pieces of the assignment. You need to have only one individual per piece.

1. The demo will be during the last lab session of the course. The lab professor will spend 5 or 10 minutes to have a quick look at the application and ask you any questions they might have about the application and how you implemented it. PLEASE plan your demo carefully so that you are able to show the deliverables quickly and demonstrate that you should receive full marks. After the demo, one of your group members should submit the 6 files to Brightspace for grading.
2. You will need to plan your time carefully, and not leave too much for the last couple of weeks. In the end, there will be 6 files to hand in (see below). The files should be submitted using Brightspace by one of the group members, in a single submission .zip format. Normally, all team members should have a chance to review the files before they are submitted, and all team members will receive the same grade,

Marking Scheme:

The assignment will be graded out of 15 points, using the following breakdown:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TASK | | | | | | MAX |
| Front End | |  |  |  |  |  |
|  | • 2 points for set of forms to update ALL underlying tables – including new multi-valued | | | | | 2 |
|  | • 2 points for set of reports to navigate and display relevant data (esp. hierarchies) | | | | | 2 |
|  | • 1 point for set of reports &/or forms for convenient updates/inserts/imports | | | | | 1 |
|  |  |  |  | Task Total |  | 5 |
| Physical model | |  |  |  |  |  |
|  | o 2 points – Covered all the minimum required entities; hierarchies, is-a, contains, related-to | | | | | 2 |
|  | o 1 point – appropriate use of surrogate keys | | | | | 1 |
|  | o 1 point - Relationships match business rules; with correct cardinality | | | | | 1 |
|  |  |  |  | Task Total |  | 4 |
| Data logistics | |  |  |  |  |  |
|  | • 1 point - test data complete, useful | | | | | 1 |
|  | • 1 point – Test Data and database creation script supplied | | | | | 1 |
|  | • 1 point – Database creation steps identified, concise, clear (Users, Instance names…) | | | | | 1 |
|  |  |  |  | Task Total |  | 3 |
| Demo/Documentation | | |  |  |  |  |
|  | • 1 point - planning - 6 Files: | | | | | 1 |
|  |  | MS-Access, Oracle database creation script, Data Model Diagram | | | |  |
|  |  | 1 pager database creation instructions, 1 pager application description, 1 pager usage instructions; | | | |  |
|  | • 2 points - execution | | | | | 2 |
|  |  |  |  | Task Total |  | 3 |

Creating a hotel booking management system involves designing a database schema that includes tables for guests, rooms, bookings, payments, and additional services. Here’s a detailed outline of the database schema with the necessary tables and their relationships.

### Database Schema

#### 1. Guests Table

This table stores information about the guests.

sql

Copy code

CREATE TABLE Guests (

GuestID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

Address TEXT,

City VARCHAR(50),

Country VARCHAR(50),

DateOfBirth DATE

);

#### 2. Rooms Table

This table stores information about the rooms in the hotel.

sql

Copy code

CREATE TABLE Rooms (

RoomID INT PRIMARY KEY AUTO\_INCREMENT,

RoomNumber VARCHAR(10) UNIQUE NOT NULL,

RoomType VARCHAR(50) NOT NULL,

PricePerNight DECIMAL(10, 2) NOT NULL,

MaxOccupancy INT NOT NULL,

Description TEXT

);

#### 3. Bookings Table

This table stores information about the bookings made by guests.

sql

Copy code

CREATE TABLE Bookings (

BookingID INT PRIMARY KEY AUTO\_INCREMENT,

GuestID INT,

RoomID INT,

CheckInDate DATE NOT NULL,

CheckOutDate DATE NOT NULL,

NumberOfGuests INT NOT NULL,

TotalAmount DECIMAL(10, 2) NOT NULL,

BookingStatus VARCHAR(20) NOT NULL,

BookingDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (GuestID) REFERENCES Guests(GuestID),

FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)

);

#### 4. Payments Table

This table stores information about the payments made for the bookings.

sql

Copy code

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY AUTO\_INCREMENT,

BookingID INT,

PaymentDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Amount DECIMAL(10, 2) NOT NULL,

PaymentMethod VARCHAR(50) NOT NULL,

PaymentStatus VARCHAR(20) NOT NULL,

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)

);

#### 5. Services Table

This table stores information about additional services offered by the hotel.

sql

Copy code

CREATE TABLE Services (

ServiceID INT PRIMARY KEY AUTO\_INCREMENT,

ServiceName VARCHAR(50) NOT NULL,

Description TEXT,

Price DECIMAL(10, 2) NOT NULL

);

#### 6. BookingServices Table

This table stores information about the services availed by guests during their stay.

sql

Copy code

CREATE TABLE BookingServices (

BookingServiceID INT PRIMARY KEY AUTO\_INCREMENT,

BookingID INT,

ServiceID INT,

Quantity INT NOT NULL,

TotalPrice DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID),

FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID)

);

### Relationships

### Detailed Report for Hotel Booking Management System

#### 1. MS-Access Application File

**Overview:** The MS-Access application will serve as the front-end for managing hotel bookings. It will connect to the Oracle 19c database where the data is stored. The application will include forms for data entry, queries for data retrieval, and reports for data presentation.

**Components:**

1. **Forms:**
   * **Guest Form:** Allows users to add, update, and delete guest information.
   * **Room Form:** Allows users to add, update, and delete room information.
   * **Booking Form:** Allows users to create new bookings, update existing ones, and view booking details.
   * **Payment Form:** Allows users to record payments for bookings.
   * **Service Form:** Allows users to manage additional services offered by the hotel.
   * **Historical Data Pop-Up Forms:** Allows users to view the history of changes for guest information, room details, and bookings.
2. **Queries:**
   * **Guest Queries:** Retrieve guest information.
   * **Room Queries:** Retrieve room information.
   * **Booking Queries:** Retrieve booking information, including join queries with guests and rooms.
   * **Payment Queries:** Retrieve payment information for bookings.
   * **Service Queries:** Retrieve information about additional services.
3. **Reports:**
   * **Guest Report:** Displays a list of all guests.
   * **Room Report:** Displays a list of all rooms.
   * **Booking Report:** Displays booking details along with guest and room information.
   * **Payment Report:** Displays payment details for bookings.
   * **Service Report:** Displays information about additional services.

**Steps to Create MS-Access Application:**

1. **Create Linked Tables:**
   * Link the tables in MS-Access to the Oracle views using ODBC connections.
   * Ensure that the linked tables point to the Oracle views created for the guests, rooms, bookings, payments, and services.
2. **Design Forms:**
   * Use the Form Design tool to create forms for guests, rooms, bookings, payments, and services.
   * Add buttons for adding, updating, and deleting records.
   * Add buttons for viewing historical data, which will open pop-up forms.
3. **Create Queries:**
   * Use the Query Design tool to create the necessary queries for retrieving data.
   * Use SQL View to write custom queries if needed.
4. **Design Reports:**
   * Use the Report Design tool to create reports based on the queries.
   * Format the reports to display data in a readable manner.
5. **Test the Application:**
   * Test all forms, queries, and reports to ensure they work correctly with the Oracle database.

#### 2. Oracle Database Creation Script

**Overview:** The Oracle database will store all the data for the hotel booking management system. It will include tables for guests, rooms, bookings, payments, and services, as well as views and triggers to manage historical data.

**Oracle Database Creation Script:**

sql

Copy code

-- Create Guests Table

CREATE TABLE Guests (

GuestID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

Address TEXT,

City VARCHAR(50),

Country VARCHAR(50),

DateOfBirth DATE

);

-- Create Rooms Table

CREATE TABLE Rooms (

RoomID INT PRIMARY KEY,

RoomNumber VARCHAR(10) UNIQUE NOT NULL,

RoomType VARCHAR(50) NOT NULL,

PricePerNight DECIMAL(10, 2) NOT NULL,

MaxOccupancy INT NOT NULL,

Description TEXT

);

-- Create Bookings Table

CREATE TABLE Bookings (

BookingID INT PRIMARY KEY,

GuestID INT,

RoomID INT,

CheckInDate DATE NOT NULL,

CheckOutDate DATE NOT NULL,

NumberOfGuests INT NOT NULL,

TotalAmount DECIMAL(10, 2) NOT NULL,

BookingStatus VARCHAR(20) NOT NULL,

BookingDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (GuestID) REFERENCES Guests(GuestID),

FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)

);

-- Create Payments Table

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY,

BookingID INT,

PaymentDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Amount DECIMAL(10, 2) NOT NULL,

PaymentMethod VARCHAR(50) NOT NULL,

PaymentStatus VARCHAR(20) NOT NULL,

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)

);

-- Create Services Table

CREATE TABLE Services (

ServiceID INT PRIMARY KEY,

ServiceName VARCHAR(50) NOT NULL,

Description TEXT,

Price DECIMAL(10, 2) NOT NULL

);

-- Create BookingServices Table

CREATE TABLE BookingServices (

BookingServiceID INT PRIMARY KEY,

BookingID INT,

ServiceID INT,

Quantity INT NOT NULL,

TotalPrice DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID),

FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID)

);

-- Create Historical Tables

CREATE TABLE GuestsHistory (

HistoryID INT PRIMARY KEY,

GuestID INT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

PhoneNumber VARCHAR(15),

Address TEXT,

City VARCHAR(50),

Country VARCHAR(50),

DateOfBirth DATE,

StartTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

EndTime TIMESTAMP

);

-- Create Views

CREATE VIEW GuestsView AS

SELECT Guests.GuestID,

Guests.FirstName,

Guests.LastName,

Guests.Email,

Guests.PhoneNumber,

Guests.Address,

Guests.City,

Guests.Country,

Guests.DateOfBirth

FROM Guests;

-- Create Triggers for Historical Data

CREATE OR REPLACE TRIGGER LogGuestsChanges

AFTER INSERT OR UPDATE OR DELETE ON Guests

FOR EACH ROW

BEGIN

IF INSERTING THEN

INSERT INTO GuestsHistory (GuestID, FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth, EndTime)

VALUES (:NEW.GuestID, :NEW.FirstName, :NEW.LastName, :NEW.Email, :NEW.PhoneNumber, :NEW.Address, :NEW.City, :NEW.Country, :NEW.DateOfBirth, NULL);

ELSIF UPDATING THEN

UPDATE GuestsHistory

SET EndTime = CURRENT\_TIMESTAMP

WHERE GuestID = :OLD.GuestID AND EndTime IS NULL;

INSERT INTO GuestsHistory (GuestID, FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth, EndTime)

VALUES (:NEW.GuestID, :NEW.FirstName, :NEW.LastName, :NEW.Email, :NEW.PhoneNumber, :NEW.Address, :NEW.City, :NEW.Country, :NEW.DateOfBirth, NULL);

ELSIF DELETING THEN

UPDATE GuestsHistory

SET EndTime = CURRENT\_TIMESTAMP

WHERE GuestID = :OLD.GuestID AND EndTime IS NULL;

END IF;

END;

/

-- Triggers for Views

CREATE OR REPLACE TRIGGER GuestsInsertTrigger

INSTEAD OF INSERT ON GuestsView

FOR EACH ROW

BEGIN

INSERT INTO Guests (GuestID, FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth)

VALUES (:NEW.GuestID, :NEW.FirstName, :NEW.LastName, :NEW.Email, :NEW.PhoneNumber, :NEW.Address, :NEW.City, :NEW.Country, :NEW.DateOfBirth);

END;

/

CREATE OR REPLACE TRIGGER GuestsUpdateTrigger

INSTEAD OF UPDATE ON GuestsView

FOR EACH ROW

BEGIN

UPDATE Guests

SET FirstName = :NEW.FirstName,

LastName = :NEW.LastName,

Email = :NEW.Email,

PhoneNumber = :NEW.PhoneNumber,

Address = :NEW.Address,

City = :NEW.City,

Country = :NEW.Country,

DateOfBirth = :NEW.DateOfBirth

WHERE GuestID = :OLD.GuestID;

END;

/

CREATE OR REPLACE TRIGGER GuestsDeleteTrigger

INSTEAD OF DELETE ON GuestsView

FOR EACH ROW

BEGIN

DELETE FROM Guests WHERE GuestID = :OLD.GuestID;

END;

/

#### 3. Data Model Diagram

**ER Diagram:** Here is a detailed ER diagram showing the relationships between tables.

plaintext

Copy code

Guests

| GuestID | FirstName | LastName | Email | PhoneNumber | Address | City | Country | DateOfBirth |

|

|----< Bookings

| | BookingID | GuestID | RoomID | CheckInDate | CheckOutDate | NumberOfGuests | TotalAmount | BookingStatus | BookingDate |

| |

| |----< Rooms

| | | RoomID | RoomNumber | RoomType | PricePerNight | MaxOccupancy | Description |

|

|----< Payments

| | PaymentID | BookingID | PaymentDate | Amount | PaymentMethod | PaymentStatus |

|

|----< BookingServices

| BookingServiceID | BookingID | ServiceID | Quantity | TotalPrice |

|

|----< Services

| ServiceID | ServiceName | Description | Price |

**Tables and Relationships:**

1. **Guests Table:**
   * Primary Key: GuestID
   * Relationships:
     + One-to-Many with Bookings (GuestID)
     + One-to-Many with Payments (GuestID via Bookings)
     + One-to-Many with BookingServices (GuestID via Bookings)
2. **Rooms Table:**
   * Primary Key: RoomID
   * Relationships:
     + One-to-Many with Bookings (RoomID)
3. **Bookings Table:**
   * Primary Key: BookingID
   * Foreign Keys: GuestID, RoomID
   * Relationships:
     + Many-to-One with Guests (GuestID)
     + Many-to-One with Rooms (RoomID)
     + One-to-Many with Payments (BookingID)
     + One-to-Many with BookingServices (BookingID)
4. **Payments Table:**
   * Primary Key: PaymentID
   * Foreign Key: BookingID
   * Relationships:
     + Many-to-One with Bookings (BookingID)
5. **Services Table:**
   * Primary Key: ServiceID
   * Relationships:
     + One-to-Many with BookingServices (ServiceID)
6. **BookingServices Table:**
   * Primary Key: BookingServiceID
   * Foreign Keys: BookingID, ServiceID
   * Relationships:
     + Many-to-One with Bookings (BookingID)
     + Many-to-One with Services (ServiceID)

### Summary

The hotel booking management system consists of a comprehensive set of tables for managing guests, rooms, bookings, payments, and services. It includes views and triggers to handle historical data and ensure data integrity. The MS-Access application serves as the front-end interface, connecting to the Oracle database via linked tables, and providing forms, queries, and reports for efficient data management.

The detailed Oracle database creation script and data model diagram help in understanding the relationships and dependencies between various entities in the system, ensuring a robust and scalable solution for hotel booking management.

### ER Diagram

Here is a simplified description of the ER diagram:

* **Guests** (GuestID, FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth)
  + One-to-many relationship with **Bookings**
* **Rooms** (RoomID, RoomNumber, RoomType, PricePerNight, MaxOccupancy, Description)
  + One-to-many relationship with **Bookings**
* **Bookings** (BookingID, GuestID, RoomID, CheckInDate, CheckOutDate, NumberOfGuests, TotalAmount, BookingStatus, BookingDate)
  + Many-to-one relationship with **Guests**
  + Many-to-one relationship with **Rooms**
  + One-to-many relationship with **Payments**
  + Many-to-many relationship with **Services** through **BookingServices**
* **Payments** (PaymentID, BookingID, PaymentDate, Amount, PaymentMethod, PaymentStatus)
  + Many-to-one relationship with **Bookings**
* **Services** (ServiceID, ServiceName, Description, Price)
  + Many-to-many relationship with **Bookings** through **BookingServices**
* **BookingServices** (BookingServiceID, BookingID, ServiceID, Quantity, TotalPrice)
  + Many-to-one relationship with **Bookings**
  + Many-to-one relationship with **Services**

### Example Queries

#### 1. Adding a New Guest

sql

Copy code

INSERT INTO Guests (FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth)

VALUES ('John', 'Doe', 'john.doe@example.com', '1234567890', '123 Main St', 'Springfield', 'USA', '1985-06-15');

#### 2. Adding a New Room

sql

Copy code

INSERT INTO Rooms (RoomNumber, RoomType, PricePerNight, MaxOccupancy, Description)

VALUES ('101', 'Deluxe', 150.00, 2, 'A deluxe room with all amenities.');

#### 3. Booking a Room

sql

Copy code

INSERT INTO Bookings (GuestID, RoomID, CheckInDate, CheckOutDate, NumberOfGuests, TotalAmount, BookingStatus)

VALUES (1, 1, '2024-07-20', '2024-07-25', 2, 750.00, 'Confirmed');

#### 4. Recording a Payment

sql

Copy code

INSERT INTO Payments (BookingID, Amount, PaymentMethod, PaymentStatus)

VALUES (1, 750.00, 'Credit Card', 'Completed');

#### 5. Adding a Service to a Booking

sql

Copy code

INSERT INTO BookingServices (BookingID, ServiceID, Quantity, TotalPrice)

VALUES (1, 1, 2, 50.00);

This schema and these queries should provide a solid foundation for your hotel booking management system. Adjust the details as needed to fit your specific requirements.

Creating a hotel booking management system that adheres to the assignment requirements involves several steps, including database design, implementation of views, triggers, and the integration of Microsoft Access with Oracle 19c. Here’s a detailed approach to help you complete the project:

### Database Design

#### 1. Tables

**Guests Table:** Stores information about guests.

sql

Copy code

CREATE TABLE Guests (

GuestID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

PhoneNumber VARCHAR(15),

Address TEXT,

City VARCHAR(50),

Country VARCHAR(50),

DateOfBirth DATE

);

**Rooms Table:** Stores information about rooms.

sql

Copy code

CREATE TABLE Rooms (

RoomID INT PRIMARY KEY,

RoomNumber VARCHAR(10) UNIQUE NOT NULL,

RoomType VARCHAR(50) NOT NULL,

PricePerNight DECIMAL(10, 2) NOT NULL,

MaxOccupancy INT NOT NULL,

Description TEXT

);

**Bookings Table:** Stores information about bookings.

sql

Copy code

CREATE TABLE Bookings (

BookingID INT PRIMARY KEY,

GuestID INT,

RoomID INT,

CheckInDate DATE NOT NULL,

CheckOutDate DATE NOT NULL,

NumberOfGuests INT NOT NULL,

TotalAmount DECIMAL(10, 2) NOT NULL,

BookingStatus VARCHAR(20) NOT NULL,

BookingDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (GuestID) REFERENCES Guests(GuestID),

FOREIGN KEY (RoomID) REFERENCES Rooms(RoomID)

);

**Payments Table:** Stores information about payments.

sql

Copy code

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY,

BookingID INT,

PaymentDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Amount DECIMAL(10, 2) NOT NULL,

PaymentMethod VARCHAR(50) NOT NULL,

PaymentStatus VARCHAR(20) NOT NULL,

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID)

);

**Services Table:** Stores information about additional services.

sql

Copy code

CREATE TABLE Services (

ServiceID INT PRIMARY KEY,

ServiceName VARCHAR(50) NOT NULL,

Description TEXT,

Price DECIMAL(10, 2) NOT NULL

);

**BookingServices Table:** Stores information about services availed by guests.

sql

Copy code

CREATE TABLE BookingServices (

BookingServiceID INT PRIMARY KEY,

BookingID INT,

ServiceID INT,

Quantity INT NOT NULL,

TotalPrice DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (BookingID) REFERENCES Bookings(BookingID),

FOREIGN KEY (ServiceID) REFERENCES Services(ServiceID)

);

### Historical Data Framework

#### Views and Triggers

**Guests View:** Create a view for the Guests table.

sql

Copy code

CREATE VIEW GuestsView AS

SELECT Guests.GuestID,

Guests.FirstName,

Guests.LastName,

Guests.Email,

Guests.PhoneNumber,

Guests.Address,

Guests.City,

Guests.Country,

Guests.DateOfBirth

FROM Guests;

**Trigger for INSERT, UPDATE, DELETE:** Create triggers to maintain historical data.

sql

Copy code

CREATE OR REPLACE TRIGGER GuestsInsertTrigger

INSTEAD OF INSERT ON GuestsView

FOR EACH ROW

BEGIN

INSERT INTO Guests (GuestID, FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth)

VALUES (:NEW.GuestID, :NEW.FirstName, :NEW.LastName, :NEW.Email, :NEW.PhoneNumber, :NEW.Address, :NEW.City, :NEW.Country, :NEW.DateOfBirth);

END;

/

CREATE OR REPLACE TRIGGER GuestsUpdateTrigger

INSTEAD OF UPDATE ON GuestsView

FOR EACH ROW

BEGIN

UPDATE Guests

SET FirstName = :NEW.FirstName,

LastName = :NEW.LastName,

Email = :NEW.Email,

PhoneNumber = :NEW.PhoneNumber,

Address = :NEW.Address,

City = :NEW.City,

Country = :NEW.Country,

DateOfBirth = :NEW.DateOfBirth

WHERE GuestID = :OLD.GuestID;

END;

/

CREATE OR REPLACE TRIGGER GuestsDeleteTrigger

INSTEAD OF DELETE ON GuestsView

FOR EACH ROW

BEGIN

DELETE FROM Guests WHERE GuestID = :OLD.GuestID;

END;

/

**Historical Data Table:** Create tables to store historical data for Guests.

sql

Copy code

CREATE TABLE GuestsHistory (

HistoryID INT PRIMARY KEY,

GuestID INT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

PhoneNumber VARCHAR(15),

Address TEXT,

City VARCHAR(50),

Country VARCHAR(50),

DateOfBirth DATE,

StartTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

EndTime TIMESTAMP

);

**Trigger to Log Changes:** Create triggers to log changes to the Guests table.

sql

Copy code

CREATE OR REPLACE TRIGGER LogGuestsChanges

AFTER INSERT OR UPDATE OR DELETE ON Guests

FOR EACH ROW

BEGIN

IF INSERTING THEN

INSERT INTO GuestsHistory (GuestID, FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth, EndTime)

VALUES (:NEW.GuestID, :NEW.FirstName, :NEW.LastName, :NEW.Email, :NEW.PhoneNumber, :NEW.Address, :NEW.City, :NEW.Country, :NEW.DateOfBirth, NULL);

ELSIF UPDATING THEN

UPDATE GuestsHistory

SET EndTime = CURRENT\_TIMESTAMP

WHERE GuestID = :OLD.GuestID AND EndTime IS NULL;

INSERT INTO GuestsHistory (GuestID, FirstName, LastName, Email, PhoneNumber, Address, City, Country, DateOfBirth, EndTime)

VALUES (:NEW.GuestID, :NEW.FirstName, :NEW.LastName, :NEW.Email, :NEW.PhoneNumber, :NEW.Address, :NEW.City, :NEW.Country, :NEW.DateOfBirth, NULL);

ELSIF DELETING THEN

UPDATE GuestsHistory

SET EndTime = CURRENT\_TIMESTAMP

WHERE GuestID = :OLD.GuestID AND EndTime IS NULL;

END IF;

END;

/

### Updating Forms and Creating Pop-Up Forms

**Updating Forms:** Update your Access forms to point to the new views.

**Creating Pop-Up Forms:** Create pop-up forms to display historical data for each field.

### Proposal Template

#### Section 1: Introduction

The project is a hotel booking management system. This system will manage guest information, room details, bookings, payments, and additional services. The system will maintain historical data for guest information, room details, and booking details.

#### Section 2: Group Membership and Tasks

| **Member Name** | **Email** | **Task** |
| --- | --- | --- |
| Member 1 | email1@example.com | Guests table and forms |
| Member 2 | email2@example.com | Rooms table and forms |
| Member 3 | email3@example.com | Bookings table and forms |

### Final Submission Files

1. MS-Access application file
2. Oracle database creation script
3. Data Model Diagram
4. One-page database creation instructions
5. One-page application description
6. One-page usage instructions

### Example ER Diagram

plaintext

Copy code

Guests

| GuestID | FirstName | LastName | Email | PhoneNumber | Address | City | Country | DateOfBirth |

|

|----> Bookings

| | BookingID | GuestID | RoomID | CheckInDate | CheckOutDate | NumberOfGuests | TotalAmount | BookingStatus | BookingDate |

| |

| |----> Rooms

| | | RoomID | RoomNumber | RoomType | PricePerNight | MaxOccupancy | Description |

|

|----> Payments

| | PaymentID | BookingID | PaymentDate | Amount | PaymentMethod | PaymentStatus |

|

|----> BookingServices

| BookingServiceID | BookingID | ServiceID | Quantity | TotalPrice |

|

|----> Services

| ServiceID | ServiceName | Description | Price |

This structure should help you set up your hotel booking management system while meeting the assignment requirements for historical data, triggers, and integration with Microsoft Access and Oracle 19c.