Project Title: Birthdate Record Management System

Course: Advanced Database Management System

Section: B

Submitted to

Juena Ahmed
Noshin

GROUP DETAILS

NAME	ID	Contribution
ADEEPTA SHUSHIL SHOVO	20-43092-1	25%
MD. YEASIN RAHAT	20-43097-1	25%
MD.ASIF AL MAMUN	20-43132-1	25%
A.S.M RIFAT	20-43767-2	25%

Contents

Topic Name	Page Number
Introduction	3
Project proposal	3-5
Diagram	5-7
User interface	8-10
Scenario Description	11-12
ER Diagram	13
Normalization	14-20
Schema Diagram	21
User Create	22
Table Crate	22-25
Data Insertion	26-31
Query Writing (SQL)	32-41
Query Writing (PL/SQL)	42-58
Relational Algebra	59
Conclusion	59-61

Introduction:

The Birth Record Management System is a project that is being considered with the goal of developing a thorough and effective system for managing birth records. Birth-related data will be stored, retrieved, and managed more easily with the help of the system, which will also ensure accuracy, security, and convenience for authorized users. By putting this system into place, we hope to increase data accuracy, expedite the birth registration procedure, and improve overall record management within the designated organization or jurisdiction.

Project Proposal:

Scope:

The following attributes will be present in the birth record management system:

Objectives:

a. Efficient Data Management: Create a centralized database to securely store and maintain birth records, doing away with the need for paper forms and lowering the likelihood of data loss or errors.

b. Simplified Birth Registration Process: Create a user-friendly interface for newborn registration that will make it simple for parents, medical staff, and authorized individuals to submit and confirm birth details.

c. Secure Data Access and Authorization: Implement strong authentication and access control measures to guarantee that only people with the proper authorization can access, read, or extract birth records, safeguarding the confidentiality and security of sensitive data.

d. Enhancing reporting and analytics tools will help with decision-making and resource allocation by generating statistical reports, demographic analyses, and trends connected to births.

e. Integration with Existing Systems: To guarantee data consistency and interoperability, seamlessly interface the Birth Record Management System with other pertinent systems, such as hospital information systems or vital statistics databases.

Features:

a. Birth Registration: Create a user-friendly interface for registering newborns that collects necessary data, including the child's name, birthplace, parents' names, and other mandatory fields.

b. Record management: Provide efficient search, retrieval, updating, and archiving of birth records for authorized staff, protecting data integrity and reducing duplication.

c. Document submit: To ensure thorough record-keeping, permit users to submit supporting papers such birth certificates, medical records, and identification documents.

d. Data validation: Use validation checks to guarantee that entered data is accurate and full, lowering errors and raising the standard of birth records as a whole.

e. Reporting and Analytics: Create individualized reports and statistical evaluations using birth records to reveal insights into birth patterns, population expansion, and other pertinent data.
f. Security and privacy: To protect sensitive data from unauthorized access or manipulation, use strong security measures, such as encryption, access limits, and audit trails.
g. Integration: Establishing data interfaces and integration protocols will allow for easy data sharing with other systems while assuring data consistency and preventing duplication..

Class Diagram, Use Case Diagram, Activity Diagram:

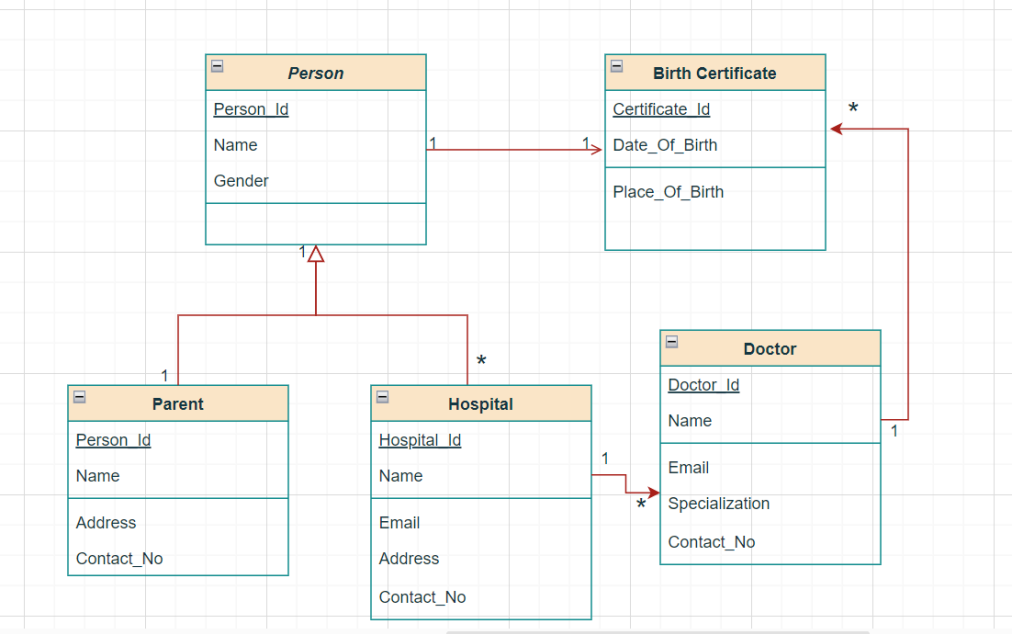


Figure 1: Class Diagram

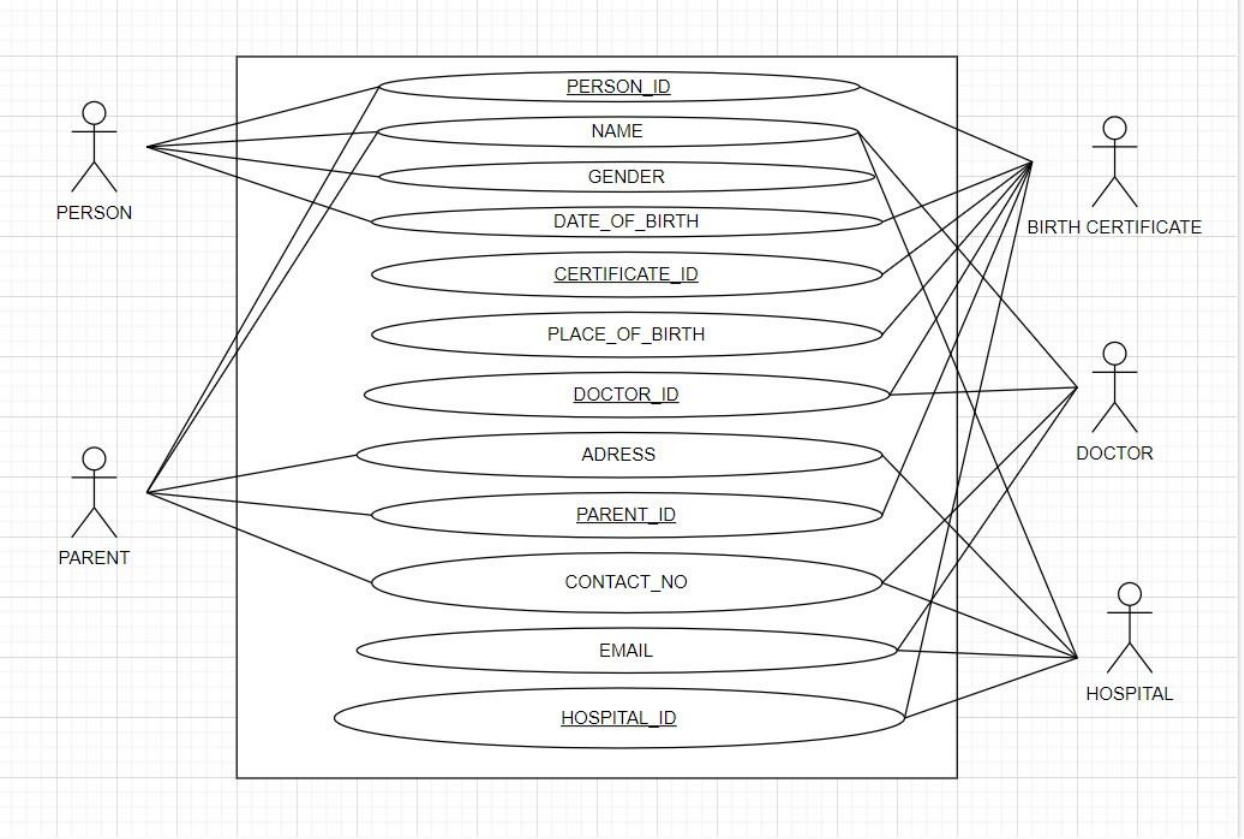


Figure 2: Use Case diagram

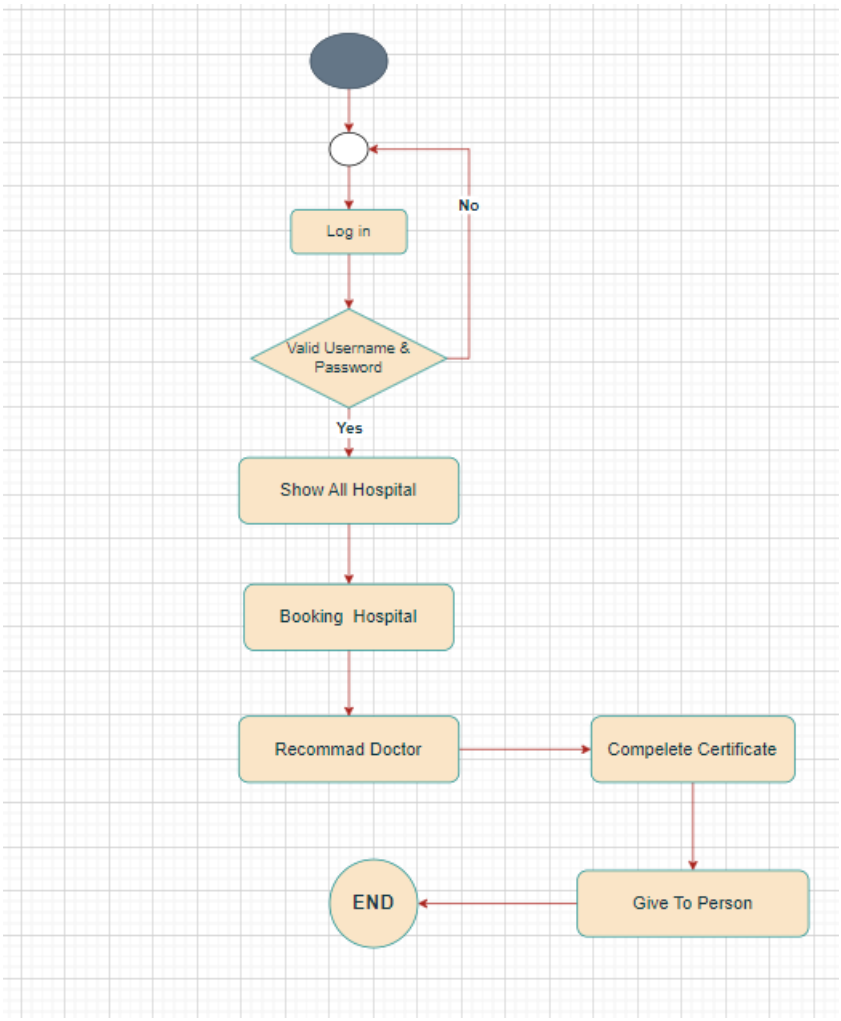


Figure 3: Activity Diagram

User Interface:

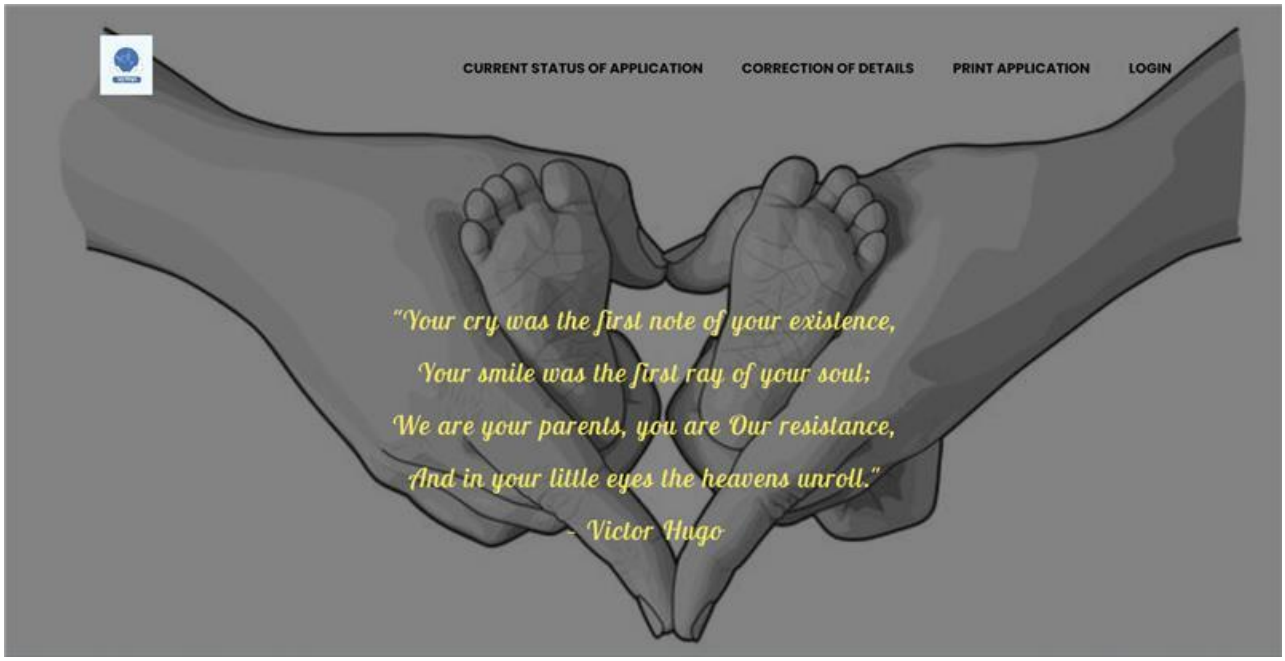


Fig: Home page

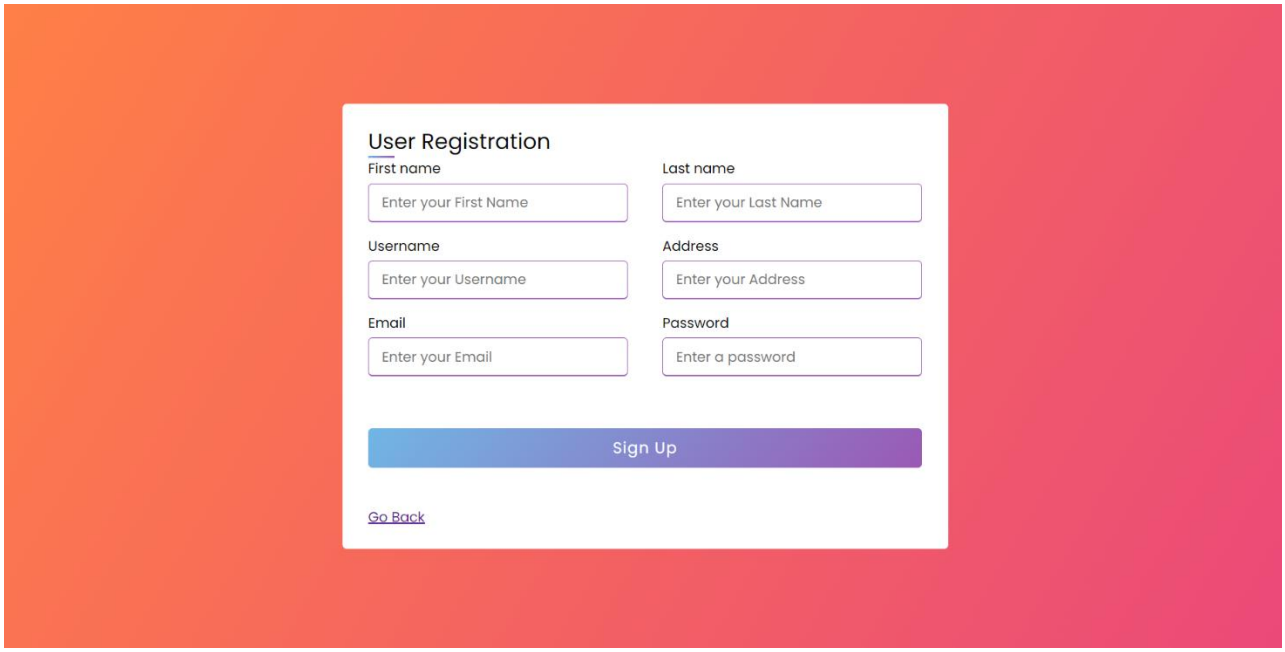


Fig: Registration page

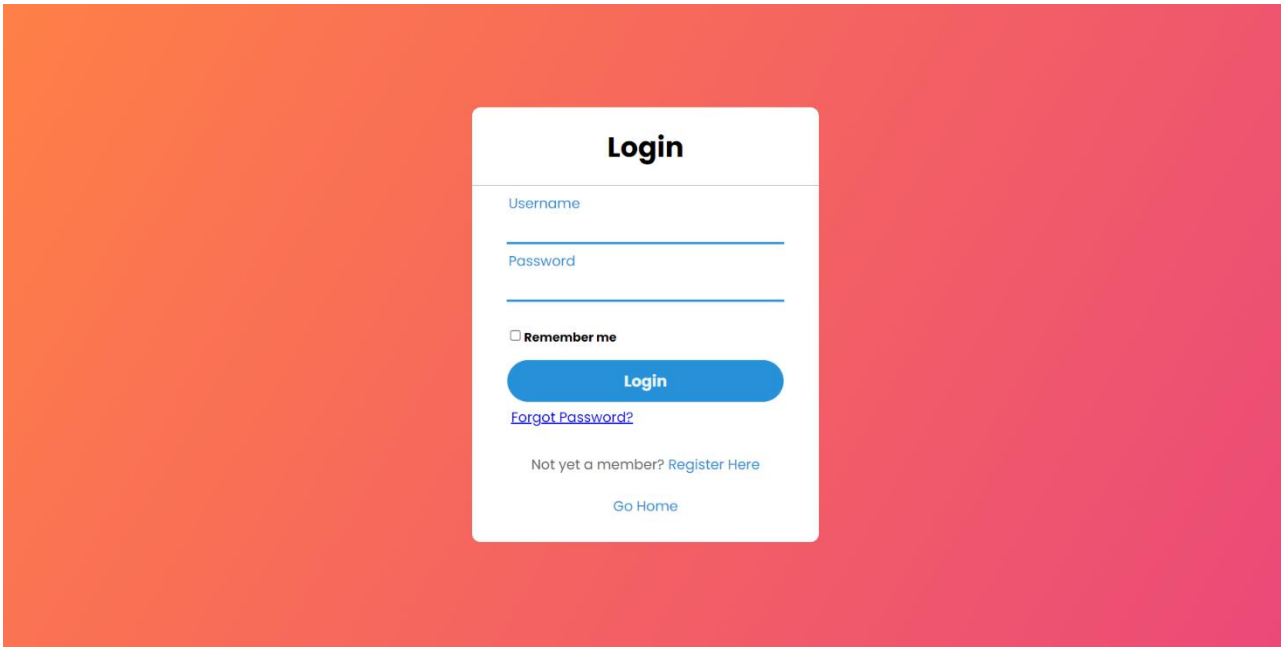


Fig: Log in page

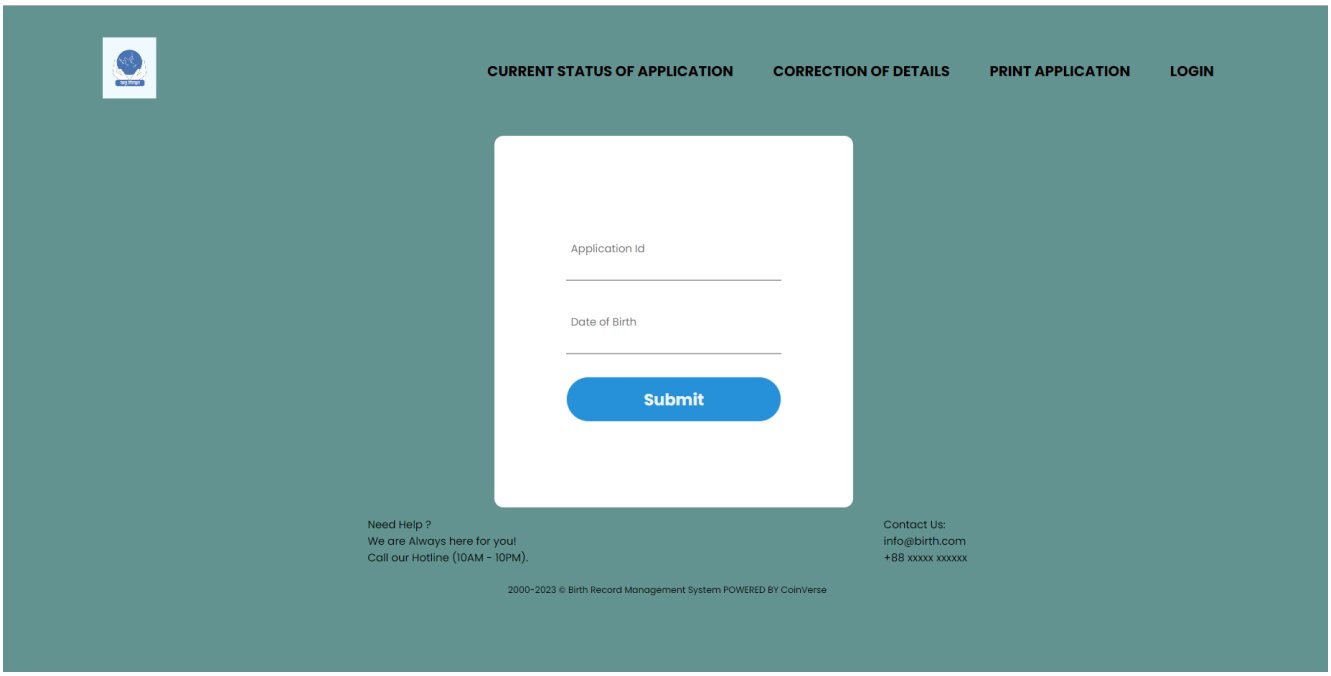


Fig: Application page

Scenario Description:

User Registration and Login:
Administrators and other designated hospital staff members would have their user accounts registered in the system. Using their login information, they could access the birth record management system.

Recording a Birth:

A hospital employee would go into the system and go to the "Birth Certificate" area whenever a new baby was born. They would start the record-creation process by entering pertinent information, such as the newborn's name, birthdate, gender, and any other necessary facts. The method would create a special identification number (certificate ID) for the birth certificate and link it to the infant.

Adding Parent Information:

The staff person would next input the parents' information, including their names, residences, phone numbers, and any other pertinent information. The system would save this data in the "Parent" table and give each parent a special ID (parent ID). Through foreign key links, the parents' person IDs would be connected to the birth certificate.

Doctor and Hospital Details:

The staff person would next note the information about the doctor or other healthcare provider who assisted with the delivery of the child. Name, area of expertise, contact information, and a special identification number (doctor ID) would all be included. The staff person would also input facts about the hospital or healthcare facility where the birth occurred, including its name, address, phone number, and other contact information, as well as a special identification code (hospital ID). Through foreign key restrictions, these tables would connect to the birth certificate table.

Accessing and Retrieving Records:

By using the search capabilities provided by the Birth Record Management System, authorized users will be able to locate birth records using a variety of parameters. For instance, hospital workers might look up data using the baby's name, birthdate, parents' names, or even the doctor's name. The system would do the searches and show the pertinent birth records and related information.

Updating and Modifying Records:

Authorized users can access the individual birth certificate and amend the appropriate data in the event that the birth record has to be changed or updated, such as to fix a misspelled name or update contact information. By updating the corresponding tables and upholding the linkages between entities, the system would guarantee data integrity.

Reporting and Analytics:

In order to provide statistical reports and analytics on birth records, the birth record management system would have reporting capabilities. Authorized users might produce reports on demographics, birth patterns, or any other pertinent information needed by the hospital management or governmental organizations.

ER Diagram:

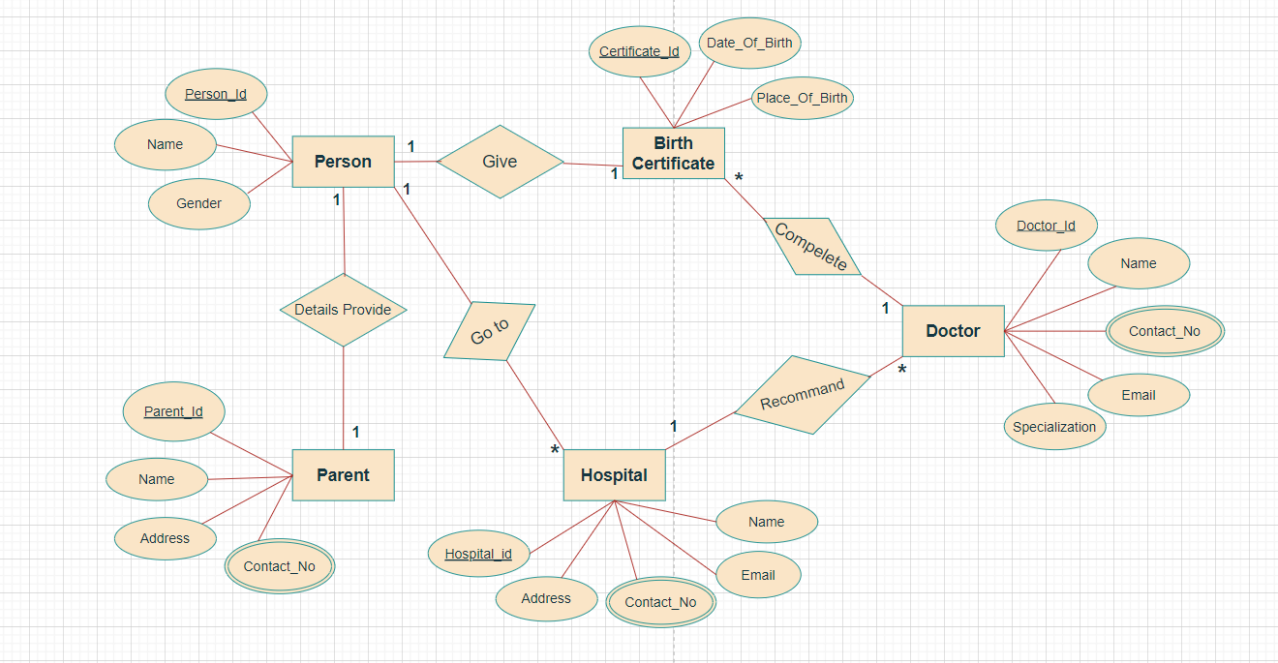


Figure 4: ER Diagram

Normalization:

Details_Provide:

UNF

Details_Provide(Person_Id, Name, Gender, Parent_Id, Name, Address, Contact_No)

1NF

Contact No is a multi valued attribute.

1. Person_Id, Name, Gender, Parent_Id, Name, Address, Contact_No

2NF

1. Person_Id, Name, Gender

2. Parent_Id, Name, Address, Contact_No

3NF

There is no transitive dependency. Relation Already in 3NF.

1. Person_Id, Name, Gender

2. Parent_Id, Name, Address, Contact_No

Table Creation

1. Person_Id, Name, Gender, Parent_Id

2. Parent_Id, Name, Address, Contact_No

Go_To:

UNF

Go_To(Person_Id, Name, Gender, Hospital_Id, Name, Email, Address, Contact_No)

1NF

Contact No is a multi valued attribute.

1. Person_Id, Name, Gender, Hospital_Id, Name, Email, Address, Contact_No

2NF

1. Person_Id, Name, Gender

2. Hospital_Id, Name, Email, Address, Contact_No

3NF

There is no transitive dependency. Relation Already in 3NF.

1. Person_Id, Name, Gender

2. Hospital_Id, Name, Email, Address, Contact_No

Table Creation

1. Person_Id, Name, Gender, Hospital_Id

2. Hospital_Id, Name, Email, Address, Contact_No

Recommand:

UNF

Recommand(Specialization, Doctor_Id, Name, Email, Contact_No, Hospital_Id, Name, Email, Address, Contact_No)

1NF

Contact No is a multi valued attribute.

1. Specialization, Doctor_Id, Name, Email, Contact_No, Hospital_Id, Name, Email, Address, Contact_No

2NF

1. Specialization, Doctor_Id, Name, Email, Contact_No

2. Hospital_Id, Name, Email, Address, Contact_No

3NF

There is no transitive dependency. Relation Already in 3NF.

1. Specialization, Doctor_Id, Name, Email, Contact_No

2. Hospital_Id, Name, Email, Address, Contact_No

Table Creation

1. Specialization, Doctor_Id, Name, Email, Contact_No

2. Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id

Complete:

UNF

Complete(Specialization, Doctor_Id, Name, Email, Contact_No, Certificate_Id, Date_Of_Birth, Place_Of_Birth)

1NF

Contact No is a multi valued attribute.

1. Specialization, Doctor_Id, Name, Email, Contact_No, Certificate_Id, Date_Of_Birth, Place_Of_Birth

2NF

1. Specialization, Doctor_Id, Name, Email, Contact_No

2. Certificate_Id, Date_Of_Birth, Place_Of_Birth

3NF

There is no transitive dependency. Relation Already in 3NF.

1. Specialization, Doctor_Id, Name, Email, Contact_No

2. Certificate_Id, Date_Of_Birth, Place_Of_Birth

Table Creation

1. Specialization, Doctor_Id, Name, Email, Contact_No, Certificate_Id

2. Certificate_Id, Date_Of_Birth, Place_Of_Birth

Give:

UNF

Give(Person_Id, Name, Gender, Certificate_Id, Date_Of_Birth, Place_Of_Birth)

1NF

There is no multi valued attribute. Relation Already in 1NF.

1. Person_Id, Name, Gender, Certificate_Id, Date_Of_Birth, Place_Of_Birth

2NF

1. Person_Id, Name, Gender

2. Certificate_Id, Date_Of_Birth, Place_Of_Birth

3NF

There is no transitive dependency. Relation Already in 3NF.

1. Person_Id, Name, Gender

2. Certificate_Id, Date_Of_Birth, Place_Of_Birth

Table Creation

1. Person_Id, Name, Gender

2. Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id

Temporary Tables:

1. Person_Id, Name, Gender, Parent_Id

2. Parent_Id, Name, Address, Contact_No

3. Person_Id, Name, Gender, Hospital_Id

4. Hospital_Id, Name, Email, Address, Contact_No

5. Specialization, Doctor_Id, Name, Email, Contact_No

6. Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id

7. Specialization, Doctor_Id, Name, Email, Contact_No, Certificate_Id

8. Certificate_Id, Date_Of_Birth, Place_Of_Birth

9. Person_Id, Name, Gender

10. Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id

Final Tables:

1. Person_Id, Name, Gender, Parent_Id, Hospital_Id

2. Parent_Id, Name, Address, Contact_No

3. Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id

4. Specialization, Doctor_Id, Name, Email, Contact_No, Certificate_Id

5. Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id

Schema Diagram:

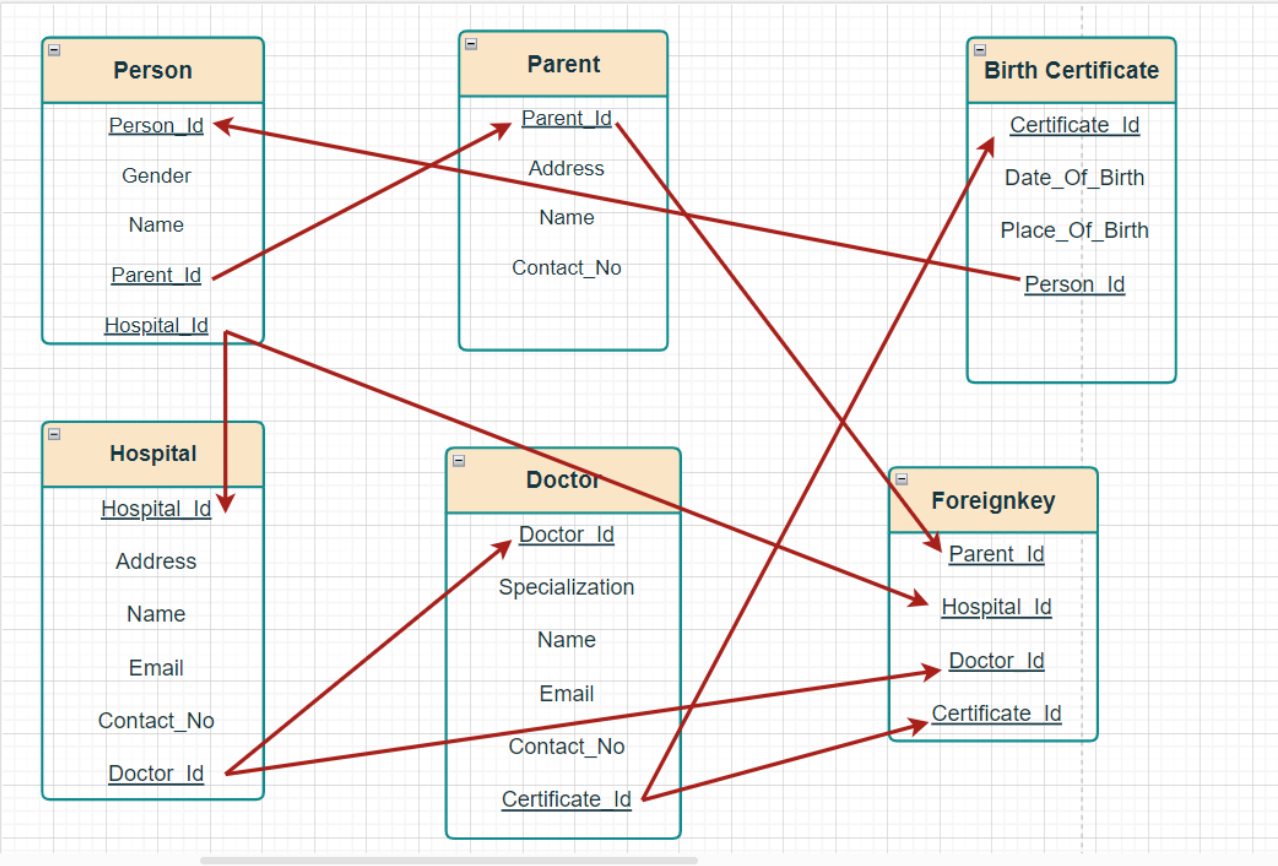


Figure 5: Schema Diagram

Table Creation:

Create User

Create user BirthCertificateSystem identified by system ;
Grant connect, create view, resource , unlimited tablespace to BirthCertificateSystem;
ALTER USER BirthCertificateSystem DEFAULT TABLESPACE USERS;
ALTER USER BirthCertificateSystem TEMPORARY TABLESPACE TEMP;

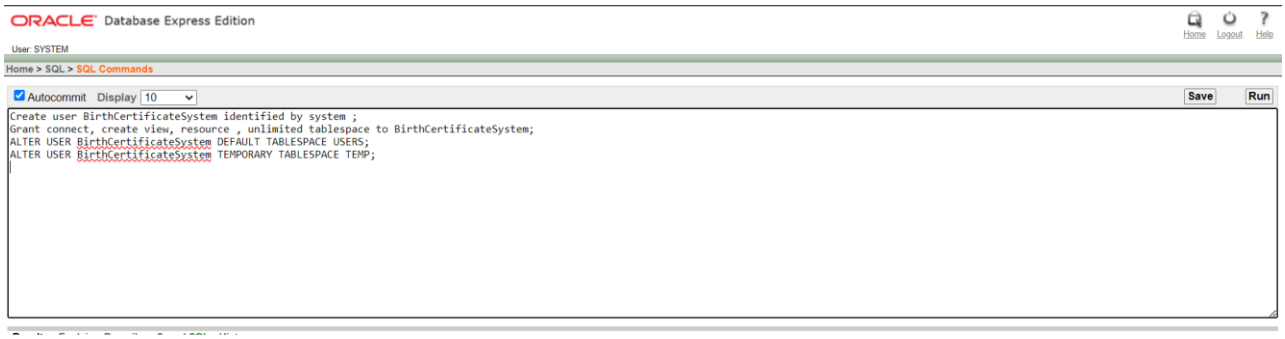
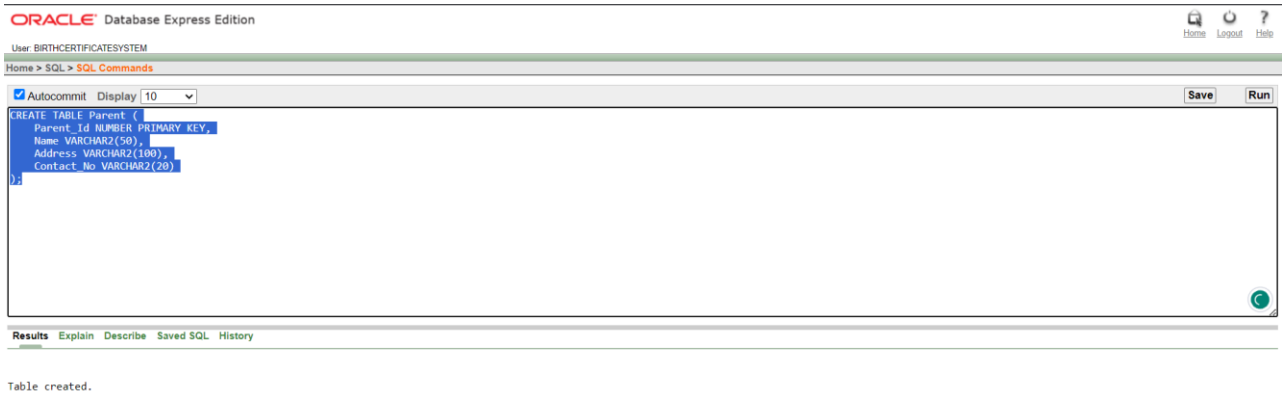


Table Create

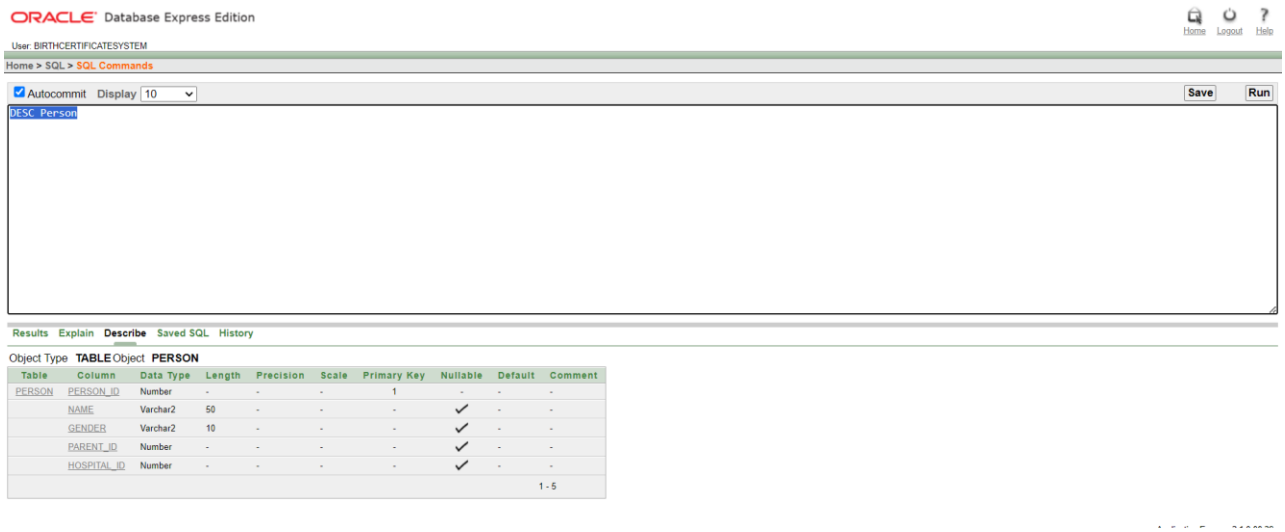
1. CREATE TABLE Parent (
   Parent_Id NUMBER PRIMARY KEY,
   Name VARCHAR2(50),
   Address VARCHAR2(100),
   Contact_No VARCHAR2(20)
);



2. CREATE TABLE Person (
    Person_Id NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Gender VARCHAR2(10),
    Parent_Id NUMBER,
    CONSTRAINT fk_Parent FOREIGN KEY (Parent_Id) REFERENCES Parent(Parent_Id)
);

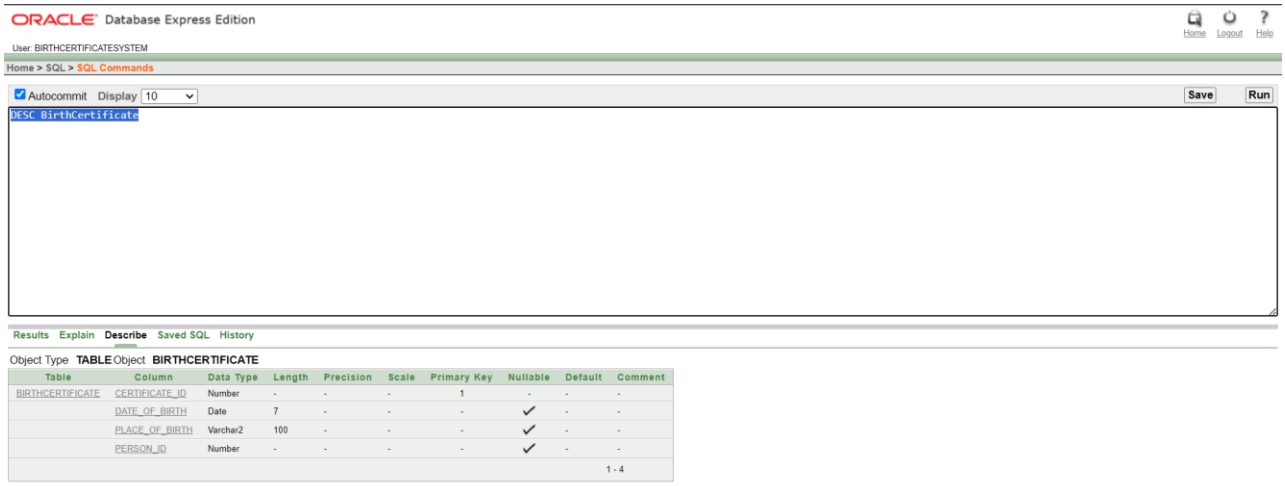
ALTER TABLE Person
ADD Hospital_Id NUMBER;

ALTER TABLE Person
ADD CONSTRAINT fk_Hospital FOREIGN KEY (Hospital_Id) REFERENCES
Hospital(Hospital_Id);

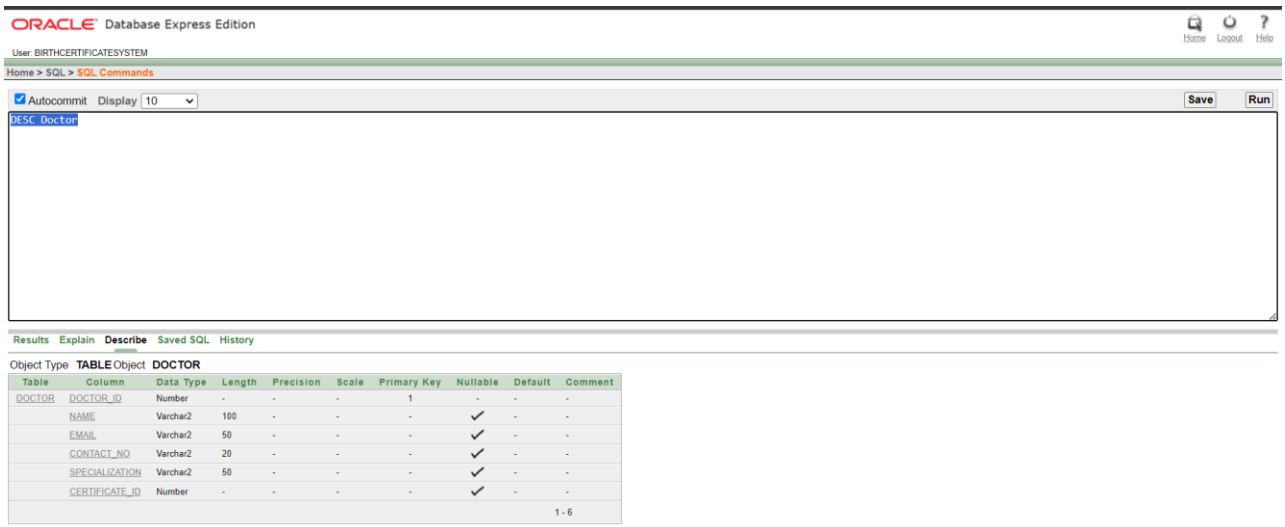


3. CREATE TABLE BirthCertificate (
    Certificate_Id NUMBER PRIMARY KEY,
    Date_Of_Birth DATE,
    Place_Of_Birth VARCHAR2(100),

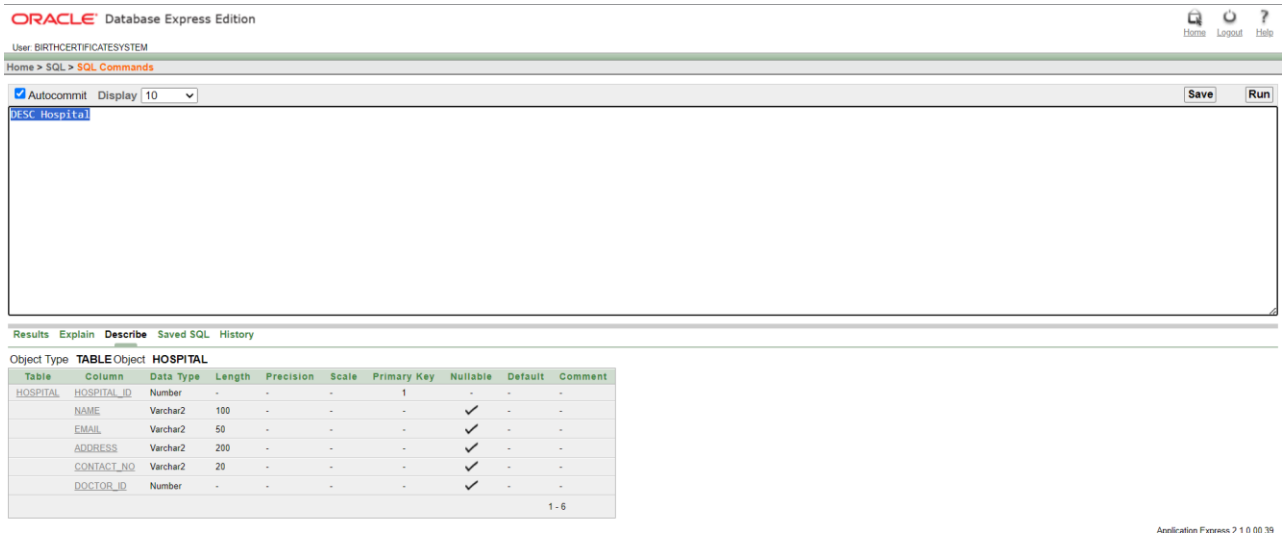
Person_Id NUMBER,
CONSTRAINT fk_Person FOREIGN KEY (Person_Id) REFERENCES Person(Person_Id)
);



4. CREATE TABLE Doctor (
Doctor_Id NUMBER PRIMARY KEY,
Name VARCHAR2(100),
Email VARCHAR2(50),
Contact_No VARCHAR2(20),
Specialization VARCHAR2(50),
Certificate_Id NUMBER,
CONSTRAINT fk_Certificate FOREIGN KEY (Certificate_Id) REFERENCES BirthCertificate (Certificate_Id)
);



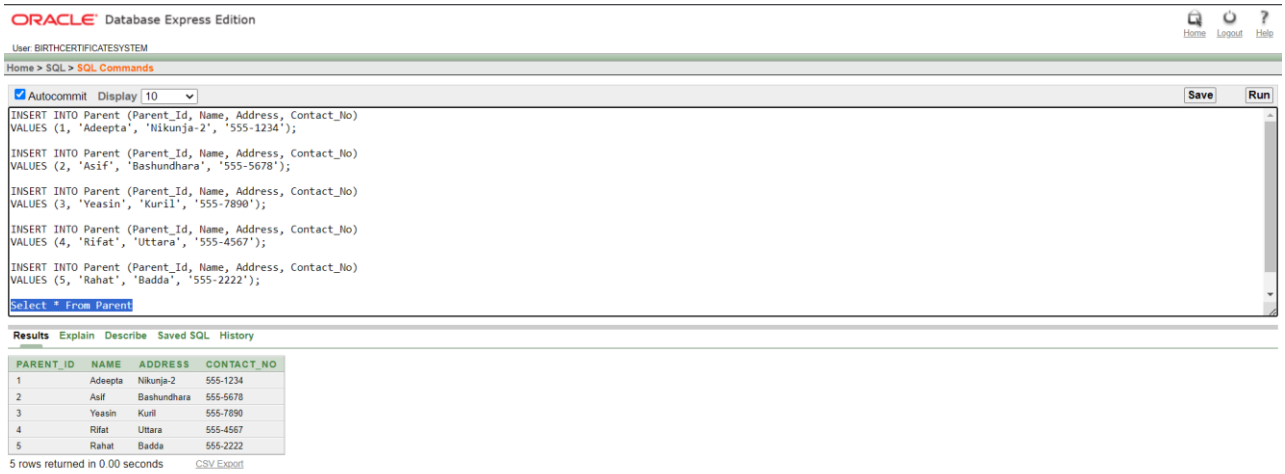
5. CREATE TABLE Hospital (
    Hospital_Id NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    Email VARCHAR2(50),
    Address VARCHAR2(200),
    Contact_No VARCHAR2(20),
    Doctor_Id NUMBER,
    CONSTRAINT fk_Doctor FOREIGN KEY (Doctor_Id) REFERENCES Doctor(Doctor_Id)
);



Data Insertion

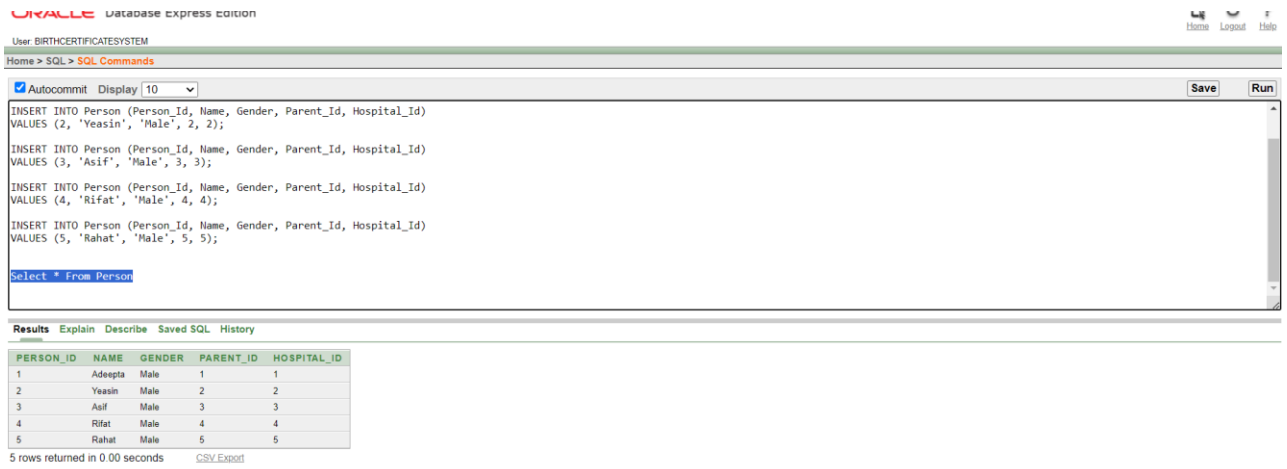
For Table 1:

```
INSERT INTO Parent (Parent_Id, Name, Address, Contact_No)
VALUES (1, 'Adeepta', 'Nikunja-2', '555-1234');

INSERT INTO Parent (Parent_Id, Name, Address, Contact_No)
VALUES (2, 'Asif', 'Bashundhara', '555-5678');

INSERT INTO Parent (Parent_Id, Name, Address, Contact_No)
VALUES (3, 'Yeasin', 'Kuril', '555-7890');

INSERT INTO Parent (Parent_Id, Name, Address, Contact_No)
VALUES (4, 'Rifat', 'Uttara', '555-4567');

INSERT INTO Parent (Parent_Id, Name, Address, Contact_No)
VALUES (5, 'Rahat', 'Badda', '555-2222');
```



For Table 2:

```
INSERT INTO Person (Person_Id, Name, Gender, Parent_Id, Hospital_Id)
VALUES (1, 'Adeepta', 'Male', 1, 1);

INSERT INTO Person (Person_Id, Name, Gender, Parent_Id, Hospital_Id)
VALUES (2, 'Yeasin', 'Male', 2, 2);

INSERT INTO Person (Person_Id, Name, Gender, Parent_Id, Hospital_Id)
VALUES (3, 'Asif', 'Male', 3, 3);

INSERT INTO Person (Person_Id, Name, Gender, Parent_Id, Hospital_Id)
VALUES (4, 'Rifat', 'Male', 4, 4);

INSERT INTO Person (Person_Id, Name, Gender, Parent_Id, Hospital_Id)
VALUES (5, 'Rahat', 'Male', 5, 5);
```



For Table 3:

INSERT INTO BirthCertificate (Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id)
VALUES (1, TO_DATE('2000-01-15', 'YYYY-MM-DD'), 'City A', 1);

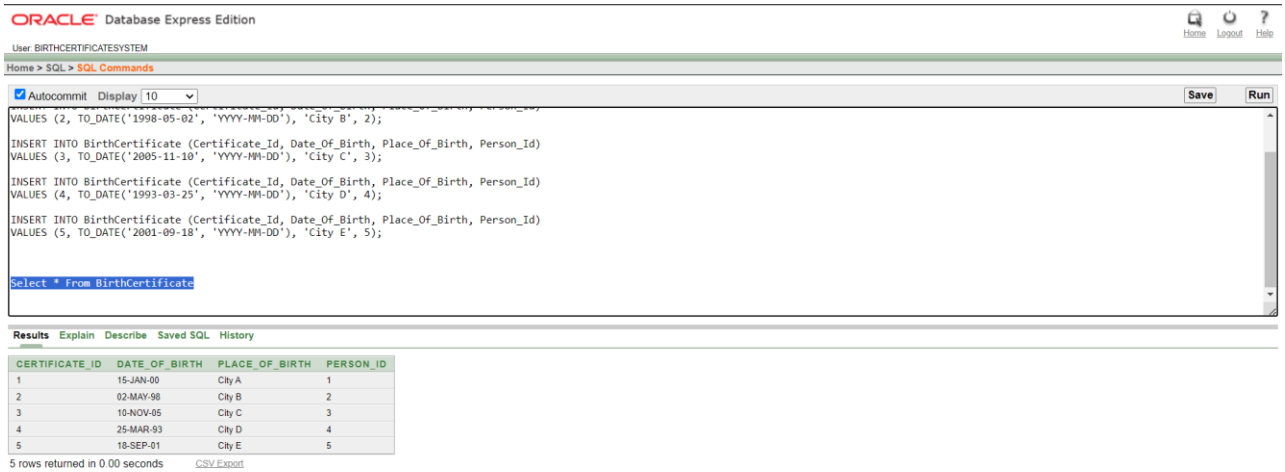
INSERT INTO BirthCertificate (Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id)
VALUES (2, TO_DATE('1998-05-02', 'YYYY-MM-DD'), 'City B', 2);

INSERT INTO BirthCertificate (Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id)
VALUES (3, TO_DATE('2005-11-10', 'YYYY-MM-DD'), 'City C', 3);

INSERT INTO BirthCertificate (Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id)

VALUES (4, TO_DATE('1993-03-25', 'YYYY-MM-DD'), 'City D', 4);

INSERT INTO BirthCertificate (Certificate_Id, Date_Of_Birth, Place_Of_Birth, Person_Id)
VALUES (5, TO_DATE('2001-09-18', 'YYYY-MM-DD'), 'City E', 5);



For Table 4:

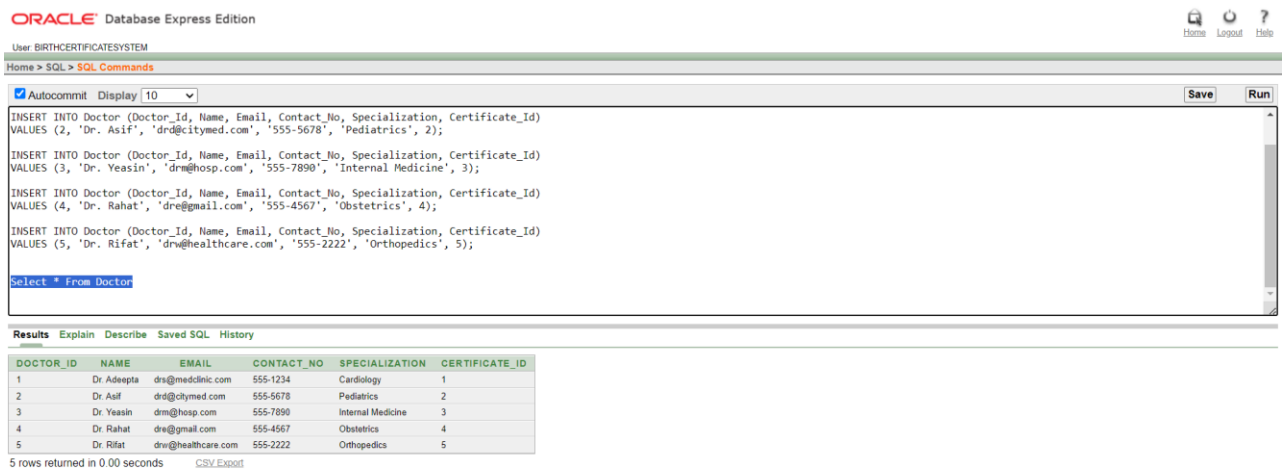
INSERT INTO Doctor (Doctor_Id, Name, Email, Contact_No, Specialization, Certificate_Id)
VALUES (1, 'Dr. Adeepta', 'drs@medclinic.com', '555-1234', 'Cardiology', 1);

INSERT INTO Doctor (Doctor_Id, Name, Email, Contact_No, Specialization, Certificate_Id)
VALUES (2, 'Dr. Asif', 'drd@citymed.com', '555-5678', 'Pediatrics', 2);

INSERT INTO Doctor (Doctor_Id, Name, Email, Contact_No, Specialization, Certificate_Id)
VALUES (3, 'Dr. Yeasin', 'drm@hosp.com', '555-7890', 'Internal Medicine', 3);

INSERT INTO Doctor (Doctor_Id, Name, Email, Contact_No, Specialization, Certificate_Id)
VALUES (4, 'Dr. Rahat', 'dre@gmail.com', '555-4567', 'Obstetrics', 4);

INSERT INTO Doctor (Doctor_Id, Name, Email, Contact_No, Specialization, Certificate_Id)
VALUES (5, 'Dr. Rifat', 'drw@healthcare.com', '555-2222', 'Orthopedics', 5);



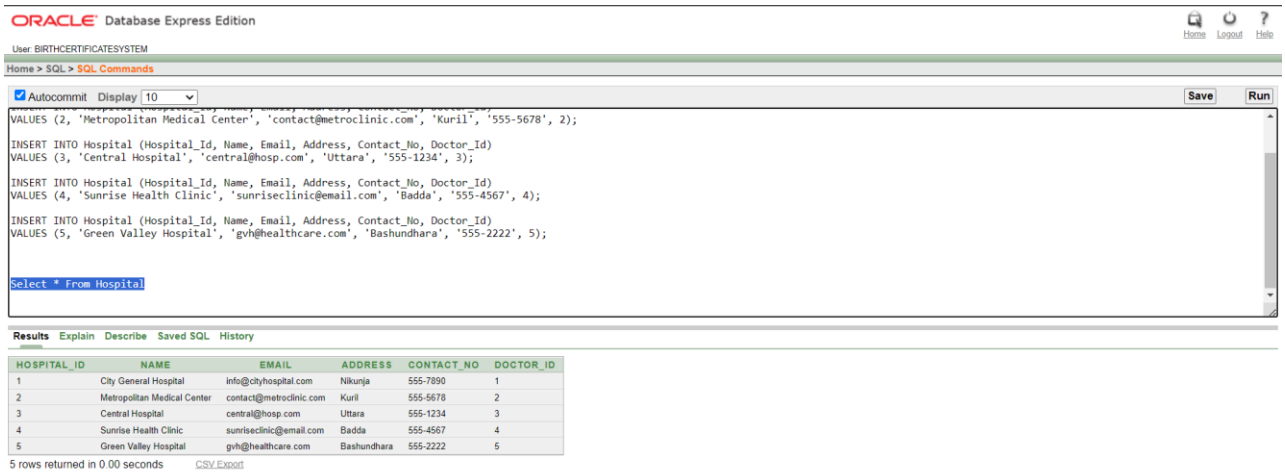
For Table 5:

INSERT INTO Hospital (Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id)
VALUES (1, 'City General Hospital', 'info@cityhospital.com', 'Nikunja', '555-7890', 1);

INSERT INTO Hospital (Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id)
VALUES (2, 'Metropolitan Medical Center', 'contact@metroclinic.com', 'Kuril', '555-5678', 2);

INSERT INTO Hospital (Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id)
VALUES (3, 'Central Hospital', 'central@hosp.com', 'Uttara', '555-1234', 3);

```
INSERT INTO Hospital (Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id)
VALUES (4, 'Sunrise Health Clinic', 'sunriseclinic@email.com', 'Badda', '555-4567', 4);

INSERT INTO Hospital (Hospital_Id, Name, Email, Address, Contact_No, Doctor_Id)
VALUES (5, 'Green Valley Hospital', 'gvh@healthcare.com', 'Bashundhara', '555-2222', 5);
```



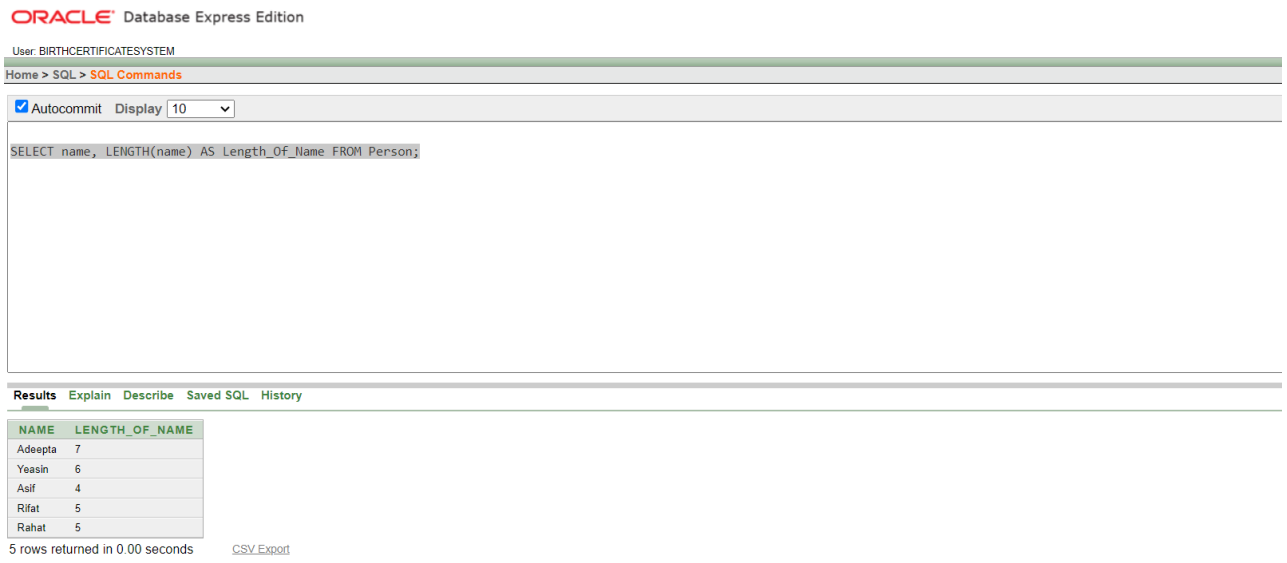
Query Writing:

SQL:

Single-row function:

1.Retrieve the length of each customer's name.

SELECT name, LENGTH(name) AS Length_Of_Name FROM Person;

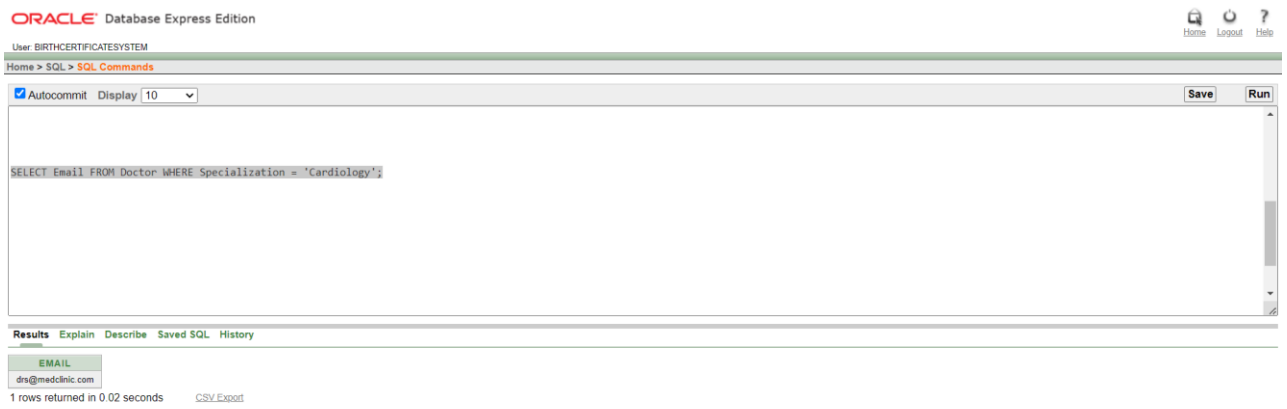


2. Display the customer names in uppercase.

SELECT name, UPPER(name) AS Name_In_Uppercase FROM Person;

3. Retrieve the email addresses of doctors who specialize in "Cardiology."

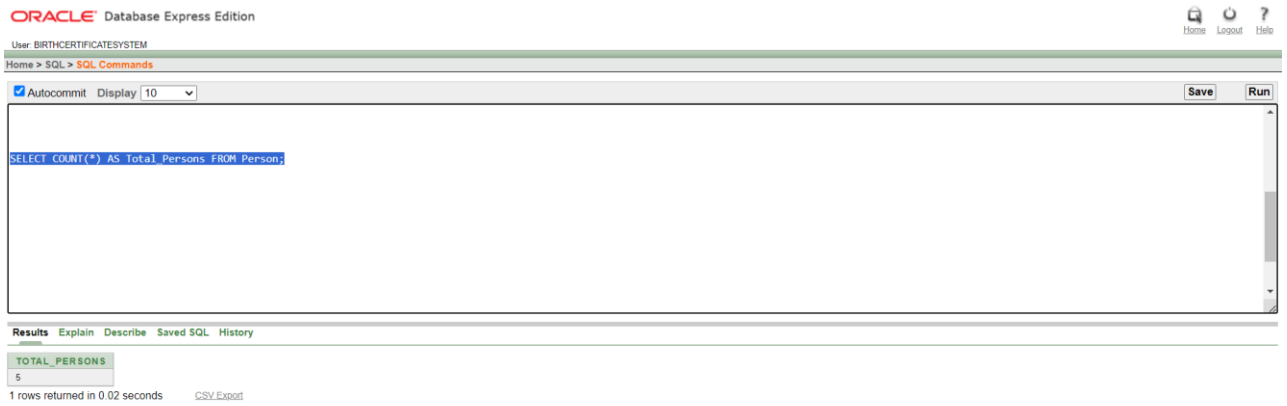
SELECT Email FROM Doctor WHERE Specialization = 'Cardiology';



Group Function:

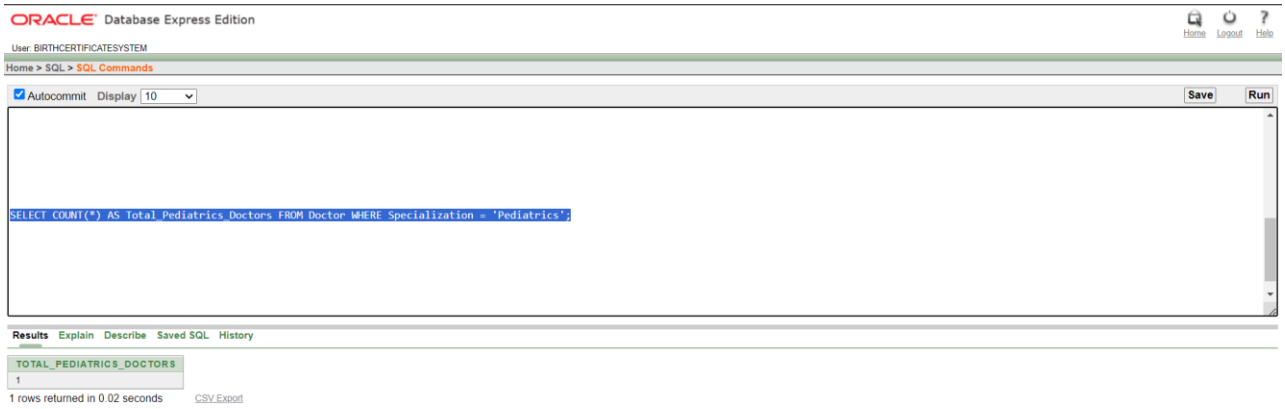
1. Calculate the total number of persons in the "Person" table.

SELECT COUNT(*) AS Total_Persons FROM Person;



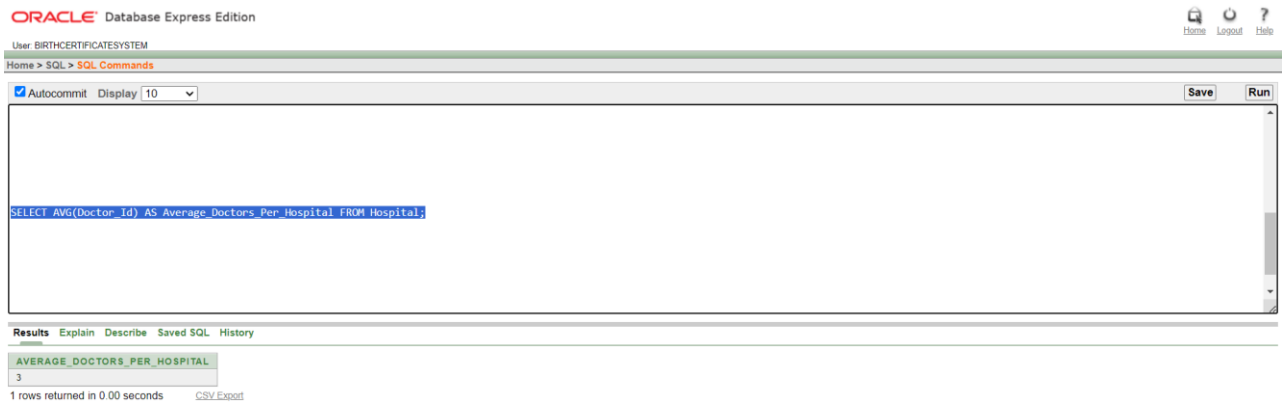
2. Calculate the total number of doctors who have a specialization in "Pediatrics" in the "Doctor" table.

SELECT COUNT(*) AS Total_Pediatrics_Doctors FROM Doctor WHERE Specialization = 'Pediatrics';



3. Find the average number of doctors per hospital in the "Hospital" table.

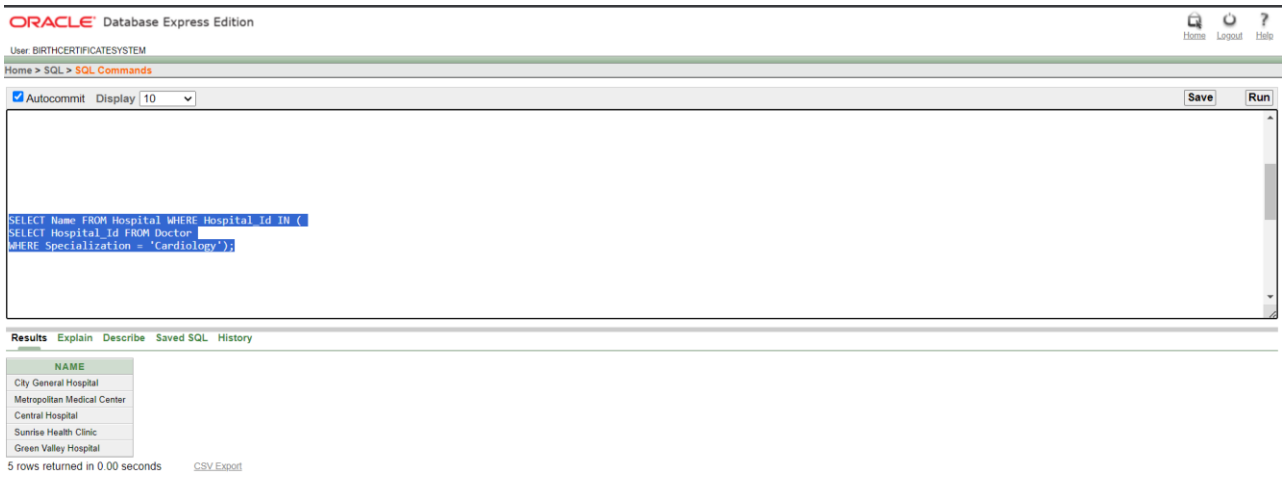
SELECT AVG(Doctor_Id) AS Average_Doctors_Per_Hospital FROM Hospital;



Sub Query:

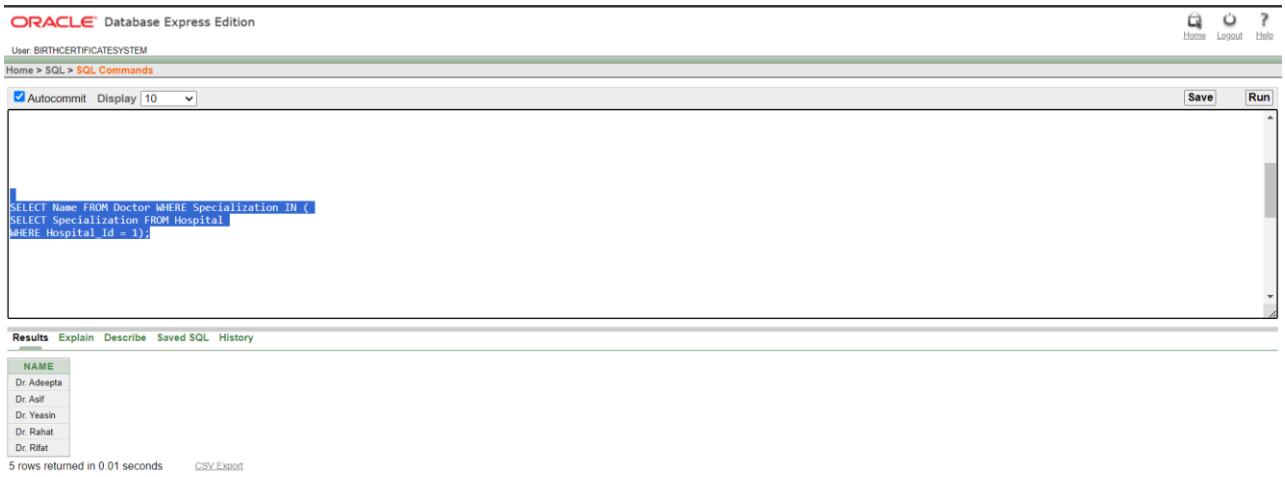
1.Find the names of hospitals where at least one doctor specializes in "Cardiology."

```
SELECT Name FROM Hospital WHERE Hospital_Id IN (
SELECT Hospital_Id FROM Doctor
WHERE Specialization = 'Cardiology');
```



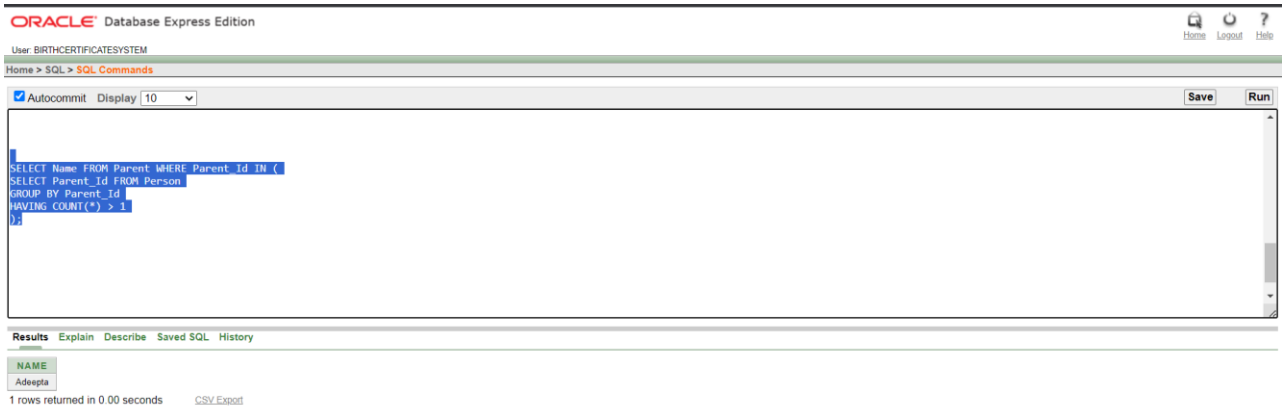
2. Find the names of doctors who have a specialization matching any of the doctors in Hospital with ID 1.

SELECT Name FROM Doctor WHERE Specialization IN (
SELECT Specialization FROM Hospital
WHERE Hospital_Id = 1);



3. Find the names of parents who have more than one child

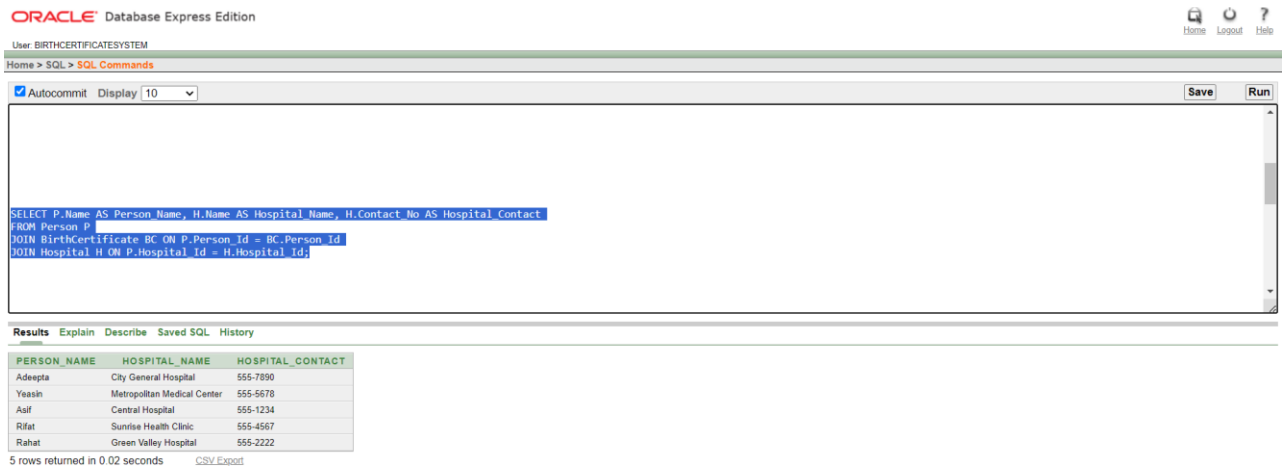
```
SELECT Name FROM Parent WHERE Parent_Id IN (
SELECT Parent_Id FROM Person
GROUP BY Parent_Id
HAVING COUNT(*) > 1
);
```



Joinning

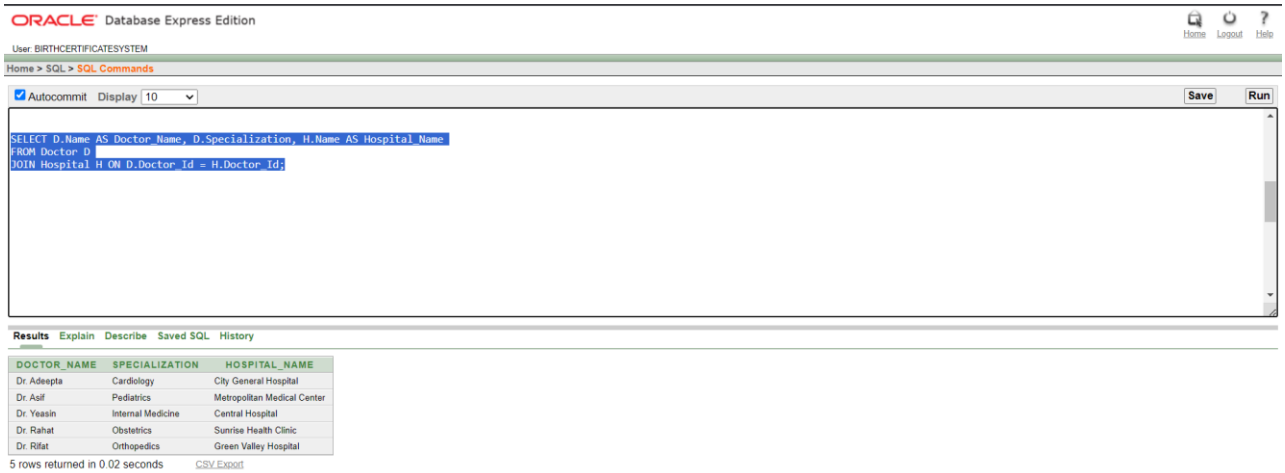
1. Retrieve the names of persons along with the hospital names where they were born, and the contact numbers of those hospitals.

```
SELECT P.Name AS Person_Name, H.Name AS Hospital_Name, H.Contact_No AS Hospital_Contact
FROM Person P
JOIN BirthCertificate BC ON P.Person_Id = BC.Person_Id
JOIN Hospital H ON P.Hospital_Id = H.Hospital_Id;
```



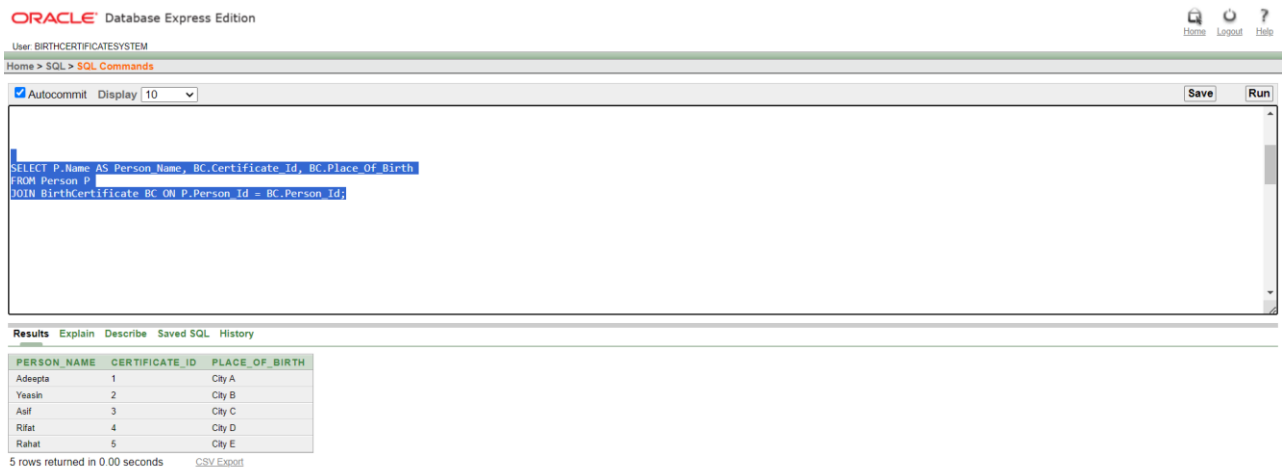
2. Retrieve the names of doctors along with their specialization and the hospital names where they work.

SELECT D.Name AS Doctor_Name, D.Specialization, H.Name AS Hospital_Name
FROM Doctor D
JOIN Hospital H ON D.Doctor_Id = H.Doctor_Id;



3. Retrieve the names of persons along with the certificate IDs and their place of birth.

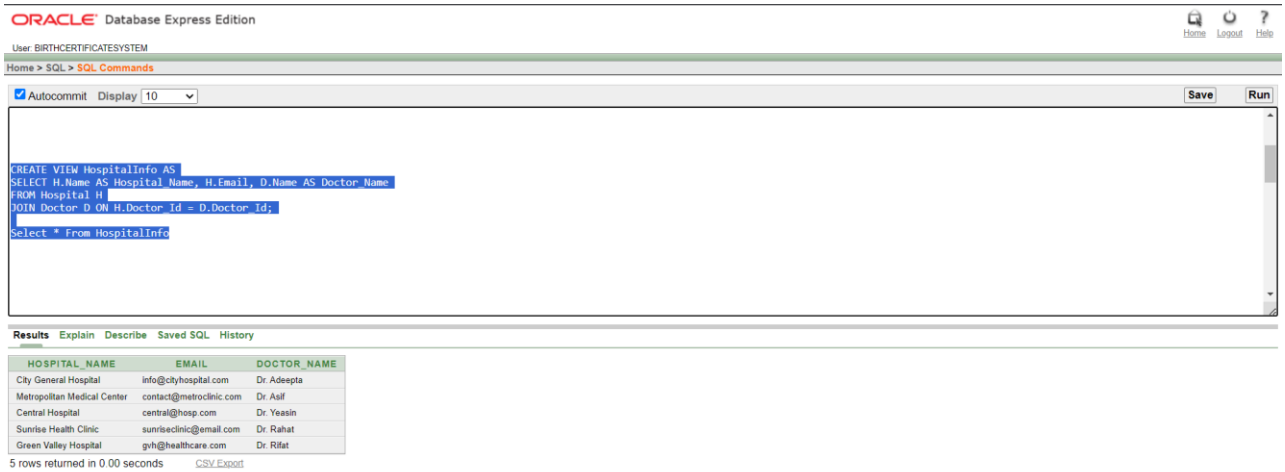
SELECT P.Name AS Person_Name, BC.Certificate_Id, BC.Place_Of_Birth
FROM Person P
JOIN BirthCertificate BC ON P.Person_Id = BC.Person_Id;



View

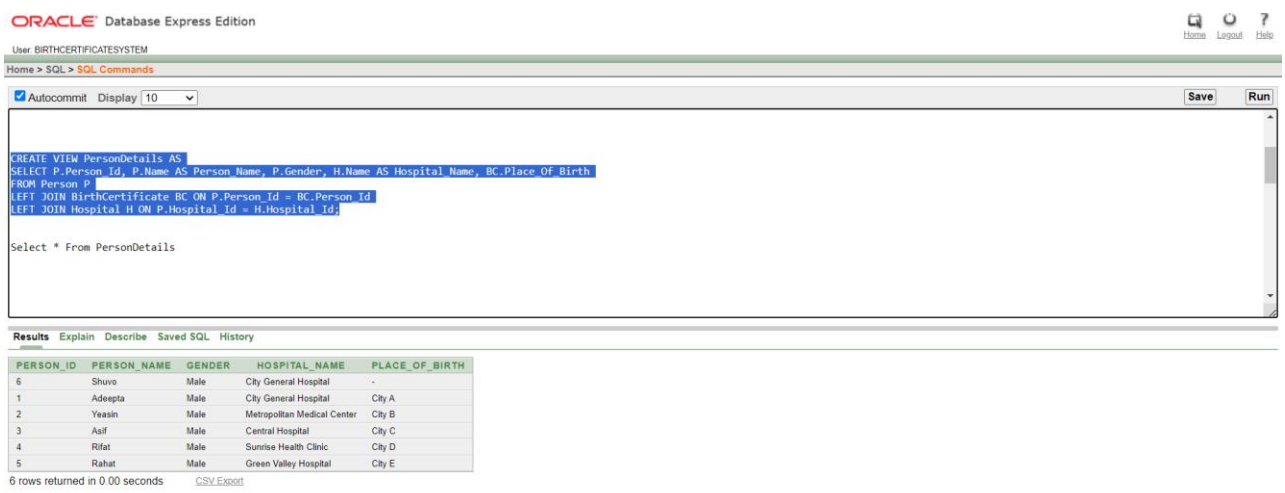
1. Create a view that displays the names of Hospitals along with their Email and Doctor names.

```
CREATE VIEW HospitalInfo AS
SELECT H.Name AS Hospital_Name, H.Email, D.Name AS Doctor_Name
FROM Hospital H
JOIN Doctor D ON H.Doctor_Id = D.Doctor_Id;

Select * From HospitalInfo
```



2. Create a view that provides information about persons including their gender, hospital names, and place of birth.

```
CREATE VIEW PersonDetails AS
SELECT P.Person_Id, P.Name AS Person_Name, P.Gender, H.Name AS Hospital_Name,
BC.Place_Of_Birth
FROM Person P
LEFT JOIN BirthCertificate BC ON P.Person_Id = BC.Person_Id
LEFT JOIN Hospital H ON P.Hospital_Id = H.Hospital_Id;

Select * From PersonDetails
```

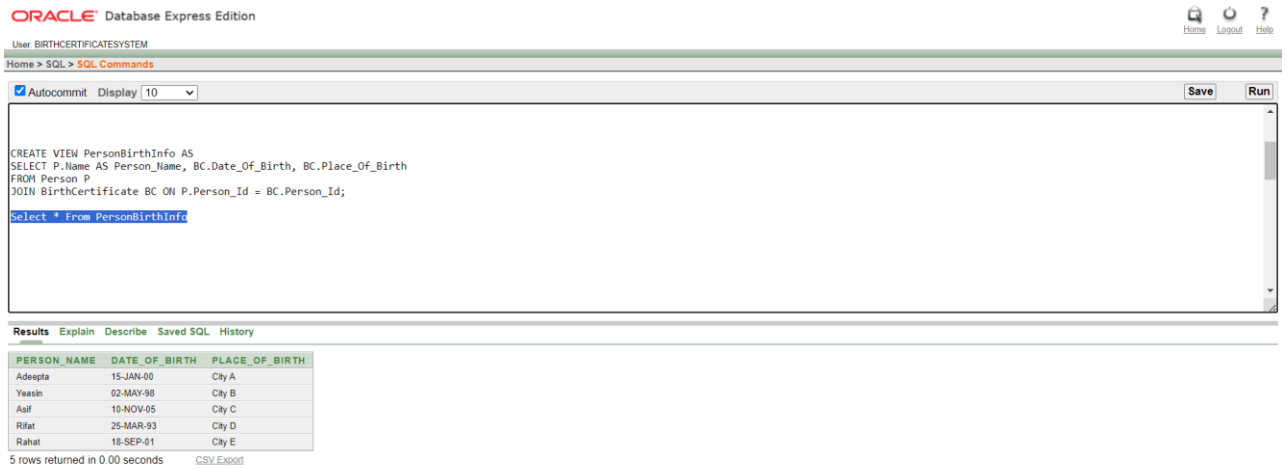


3. Create a view that combines information from the BirthCertificate table and the Person table to show names of persons along with their date of birth and place of birth.

```
CREATE VIEW PersonBirthInfo AS
SELECT P.Name AS Person_Name, BC.Date_Of_Birth, BC.Place_Of_Birth
FROM Person P
```

JOIN BirthCertificate BC ON P.Person_Id = BC.Person_Id;

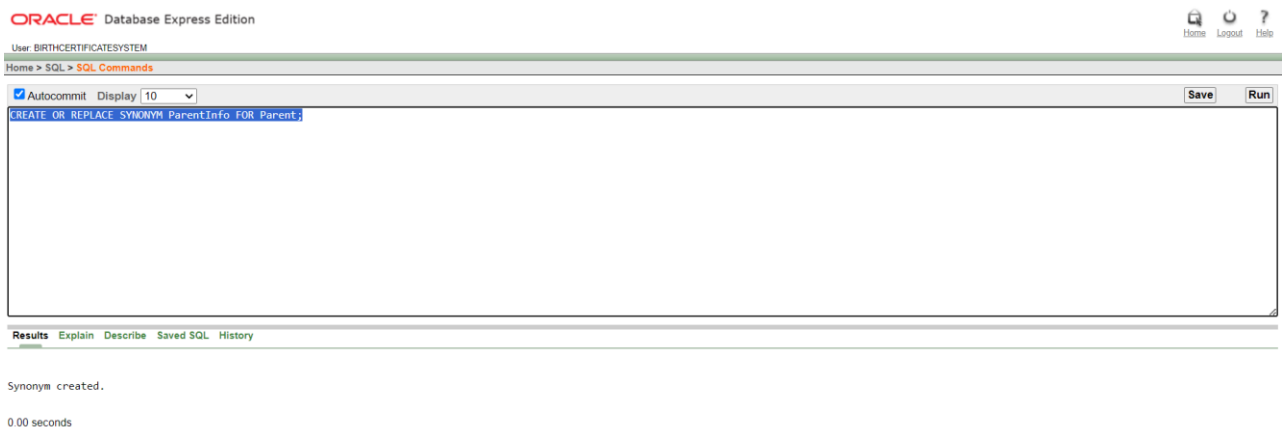
Select * From PersonBirthInfo



Synonym:

1. Create a synonym named "ParentInfo" that refers to the "Parent" table.

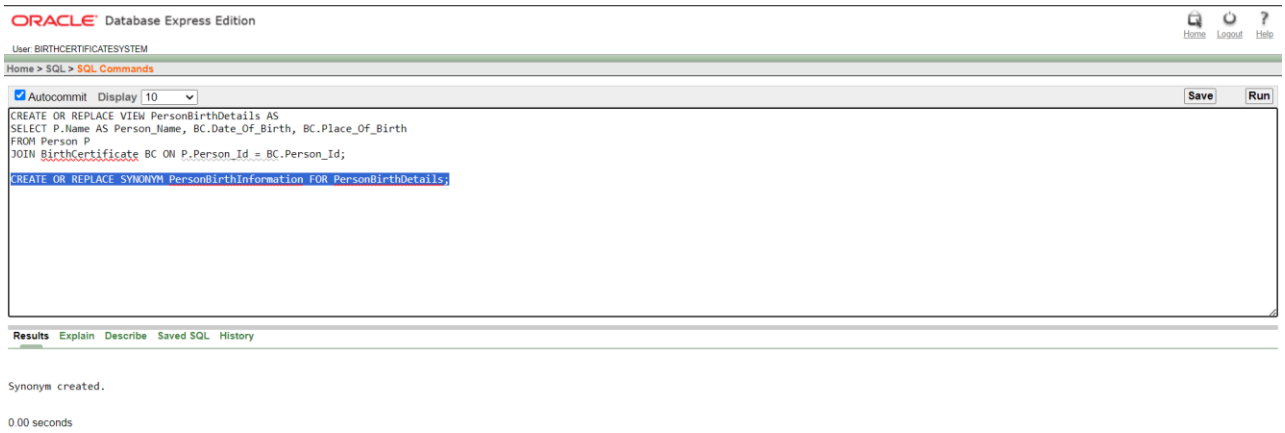
CREATE OR REPLACE SYNONYM ParentInfo FOR Parent;



2. Create a synonym named "PersonBirthInformation" that refers to a query combining information from the "Person" and "BirthCertificate" tables to show names of persons along with their date of birth and place of

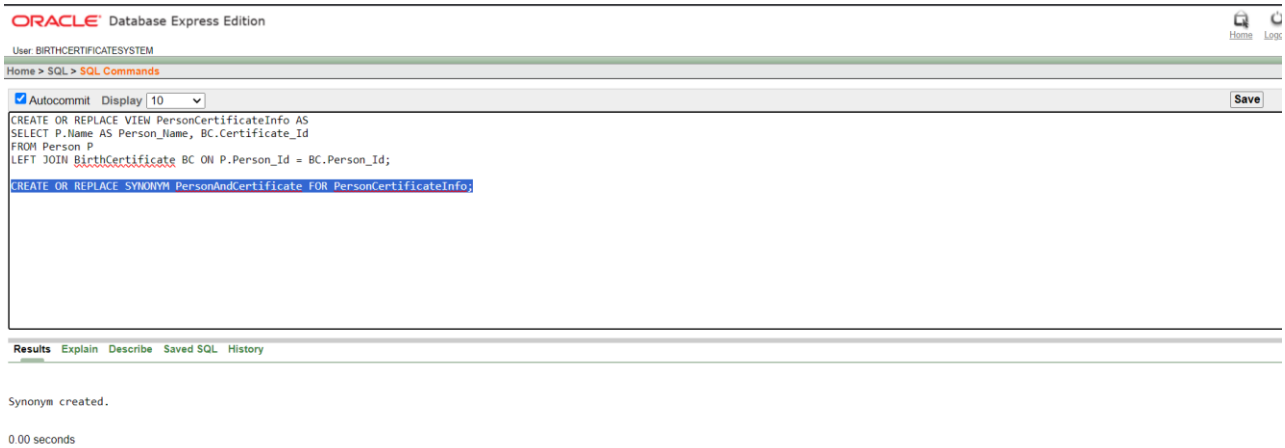
birth.

```
CREATE OR REPLACE VIEW PersonBirthDetails AS
SELECT P.Name AS Person_Name, BC.Date_Of_Birth, BC.Place_Of_Birth
FROM Person P
JOIN BirthCertificate BC ON P.Person_Id = BC.Person_Id;

CREATE OR REPLACE SYNONYM PersonBirthInformation FOR PersonBirthDetails;
```



3. Create a synonym named "PersonAndCertificate" that refers to a query combining information from the "Person" and "BirthCertificate" tables to show names of persons along with their certificate IDs, if available.

```
CREATE OR REPLACE VIEW PersonCertificateInfo AS
SELECT P.Name AS Person_Name, BC.Certificate_Id
FROM Person P
LEFT JOIN BirthCertificate BC ON P.Person_Id = BC.Person_Id;

CREATE OR REPLACE SYNONYM PersonAndCertificate FOR PersonCertificateInfo;
```

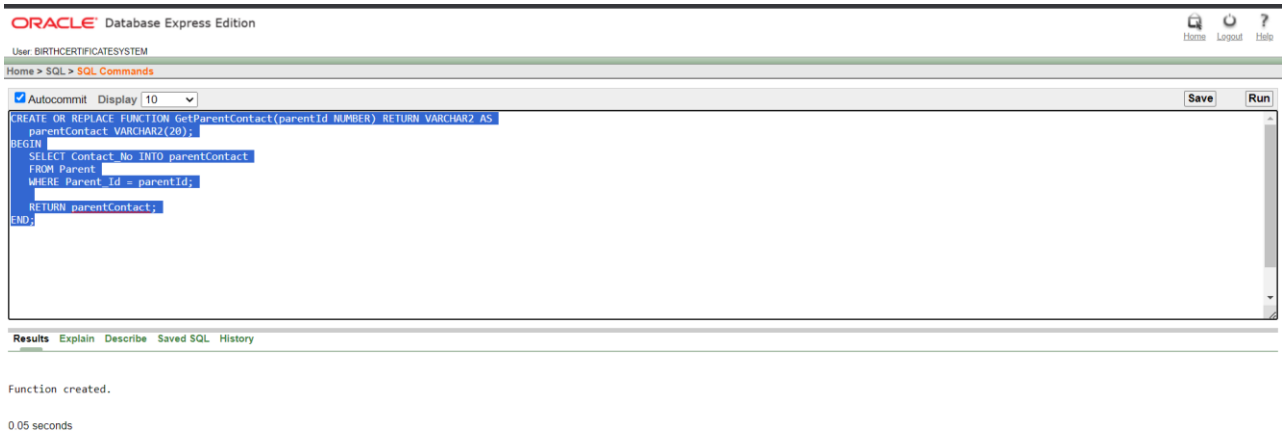


PL/SQL:

Function:

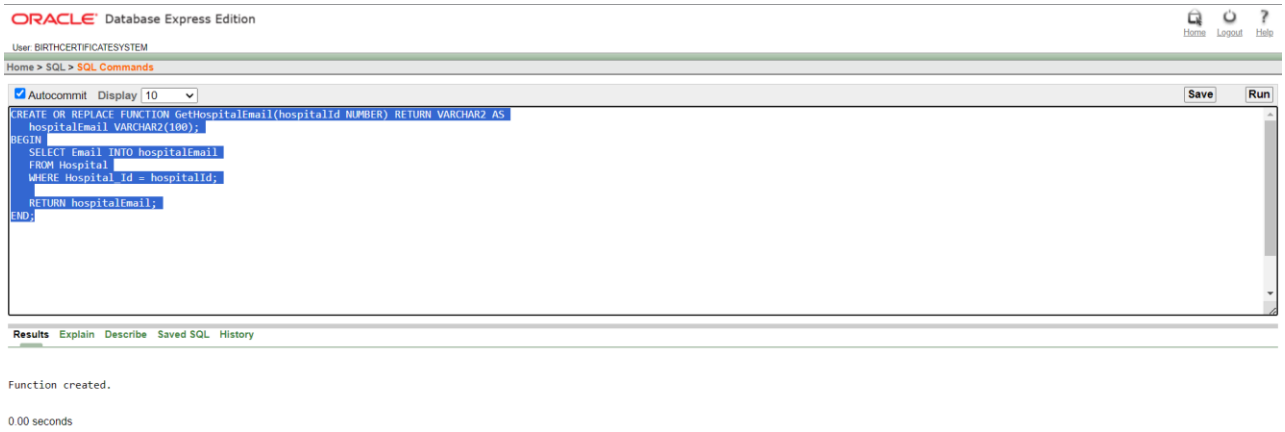
1. Create a PL/SQL function named "GetParentContact" that takes a parent ID as input and returns the contact number of the parent.

```
CREATE OR REPLACE FUNCTION GetParentContact(parentId NUMBER) RETURN VARCHAR2 AS
  parentContact VARCHAR2(20);
BEGIN
  SELECT Contact_No INTO parentContact
  FROM Parent
  WHERE Parent_Id = parentId;

  RETURN parentContact;
END;
```



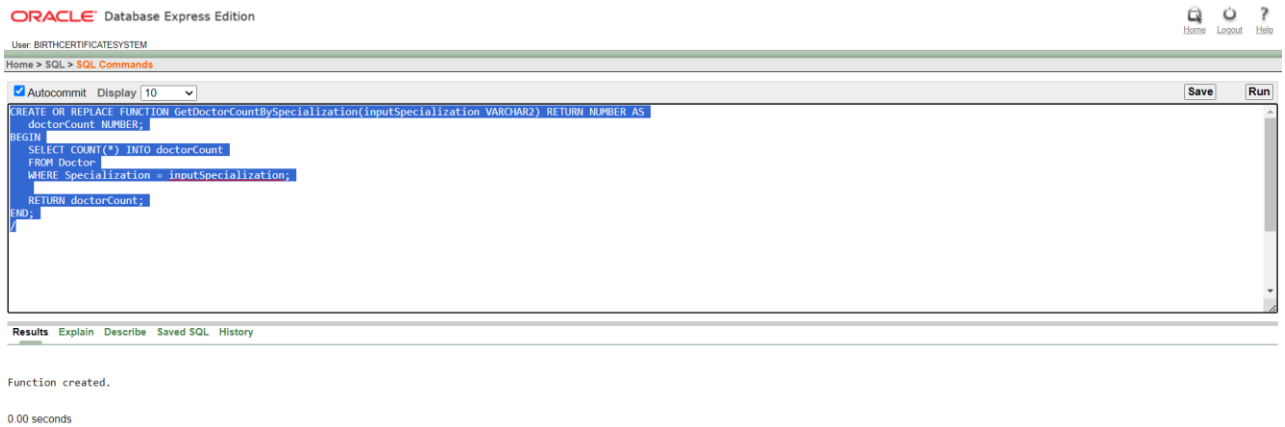
2. Create a PL/SQL function named "GetHospitalEmail" that takes a hospital ID as input and returns the email address of the hospital.

```
CREATE OR REPLACE FUNCTION GetHospitalEmail(hospitalId NUMBER) RETURN VARCHAR2
AS
  hospitalEmail VARCHAR2(100);
BEGIN
  SELECT Email INTO hospitalEmail
  FROM Hospital
  WHERE Hospital_Id = hospitalId;

  RETURN hospitalEmail;
END;
```



3. Create a PL/SQL function named "GetDoctorCountBySpecialization" that takes a specialization as input and returns the count of doctors with that specialization.

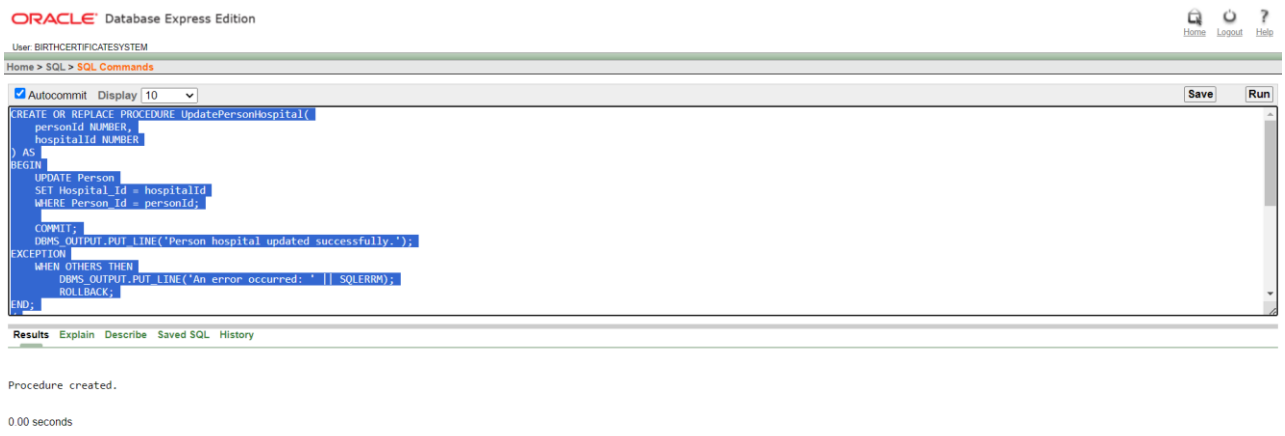
```
CREATE OR REPLACE FUNCTION GetDoctorCountBySpecialization(inputSpecialization VARCHAR2)
RETURN NUMBER AS
  doctorCount NUMBER;
BEGIN
  SELECT COUNT(*) INTO doctorCount
  FROM Doctor
  WHERE Specialization = inputSpecialization;

  RETURN doctorCount;
END;
/
```



Procedure:

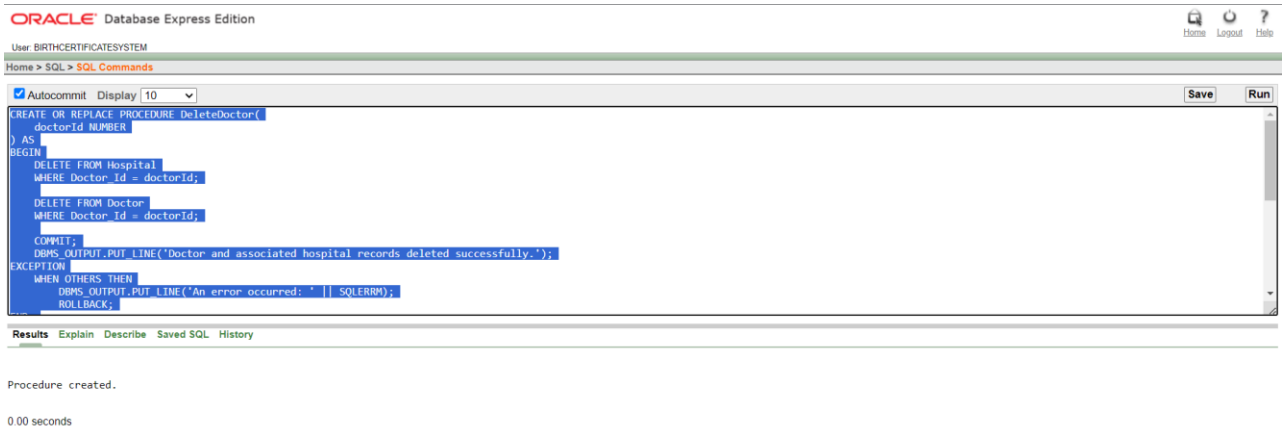
1. Create a PL/SQL procedure named "UpdatePersonHospital" that takes a person's ID and a hospital's ID as input, and updates the person's hospital association.

```
CREATE OR REPLACE PROCEDURE UpdatePersonHospital(
    personId NUMBER,
    hospitalId NUMBER
) AS
BEGIN
    UPDATE Person
    SET Hospital_Id = hospitalId
    WHERE Person_Id = personId;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Person hospital updated successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
        ROLLBACK;
END;
/
```



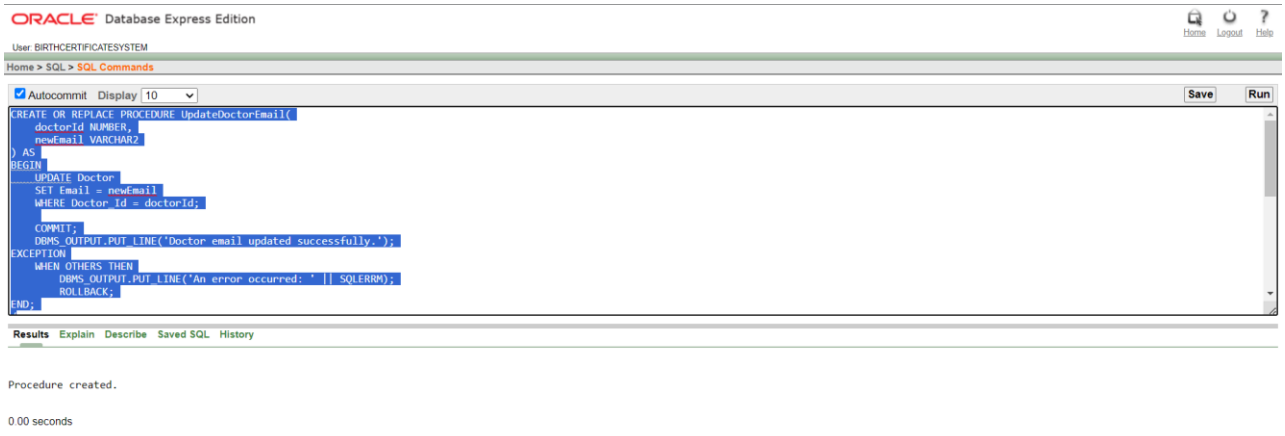
2. Create a PL/SQL procedure named "DeleteDoctor" that takes a doctor's ID as input, and deletes the doctor's record along with associated hospital records.

```
CREATE OR REPLACE PROCEDURE DeleteDoctor(
    doctorId NUMBER
) AS
BEGIN
    DELETE FROM Hospital
    WHERE Doctor_Id = doctorId;

    DELETE FROM Doctor
    WHERE Doctor_Id = doctorId;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Doctor and associated hospital records deleted successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
        ROLLBACK;
END;
/
```



3. Create a PL/SQL procedure named "UpdateDoctorEmail" that takes a doctor's ID and a new email address as input, and updates the doctor's email.

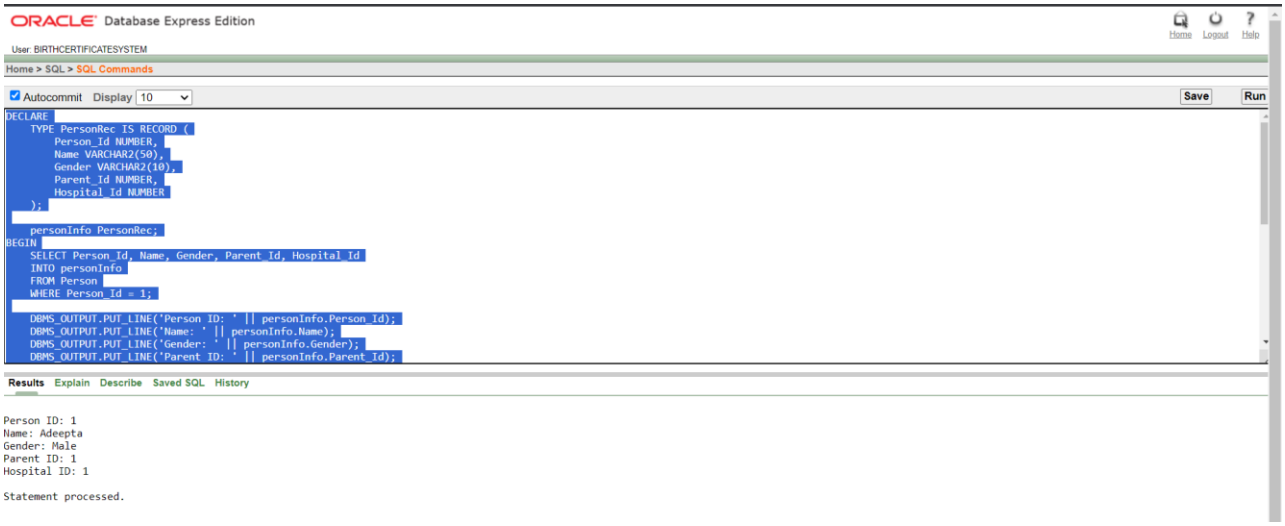
```
CREATE OR REPLACE PROCEDURE UpdateDoctorEmail(
    doctorId NUMBER,
    newEmail VARCHAR2
) AS
BEGIN
    UPDATE Doctor
    SET Email = newEmail
    WHERE Doctor_Id = doctorId;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Doctor email updated successfully.');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
        ROLLBACK;
END;
/
```



Record:

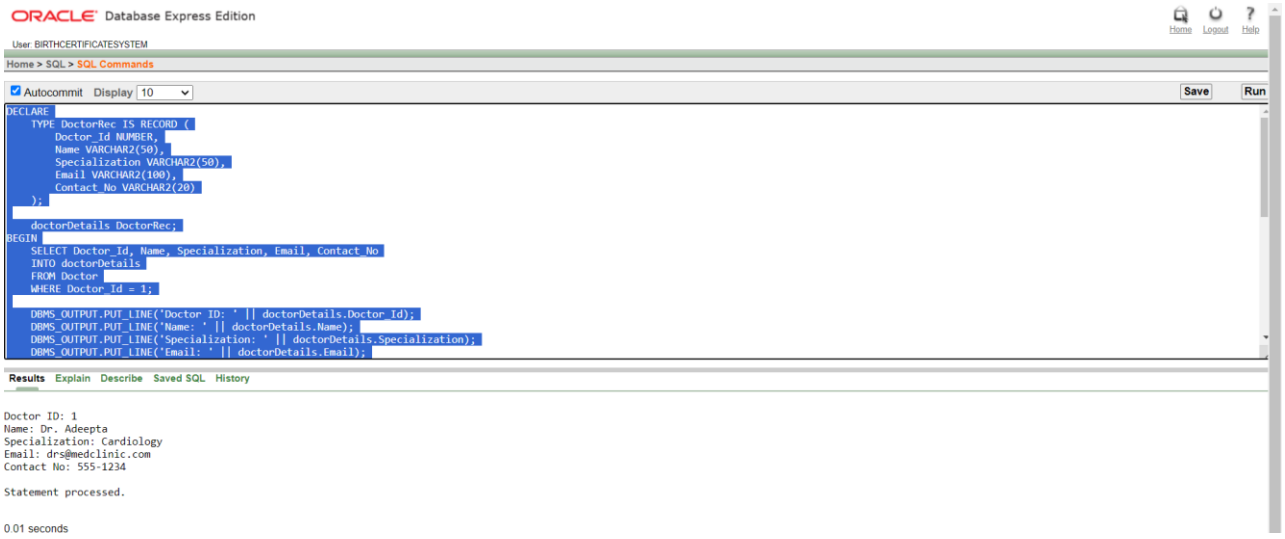
1.Create a PL/SQL record named "PersonInfo" that can hold the attributes of a person (Person_Id, Name, Gender, Parent_Id, Hospital_Id). Use this record to fetch and display information about a specific person.

```
DECLARE
   TYPE PersonRec IS RECORD (
      Person_Id NUMBER,
      Name VARCHAR2(50),
      Gender VARCHAR2(10),
      Parent_Id NUMBER,
      Hospital_Id NUMBER
   );

   personInfo PersonRec;
BEGIN
   SELECT Person_Id, Name, Gender, Parent_Id, Hospital_Id
   INTO personInfo
   FROM Person
   WHERE Person_Id = 1;

   DBMS_OUTPUT.PUT_LINE('Person ID: ' || personInfo.Person_Id);
   DBMS_OUTPUT.PUT_LINE('Name: ' || personInfo.Name);
   DBMS_OUTPUT.PUT_LINE('Gender: ' || personInfo.Gender);
   DBMS_OUTPUT.PUT_LINE('Parent ID: ' || personInfo.Parent_Id);
   DBMS_OUTPUT.PUT_LINE('Hospital ID: ' || personInfo.Hospital_Id);
END;
/
```



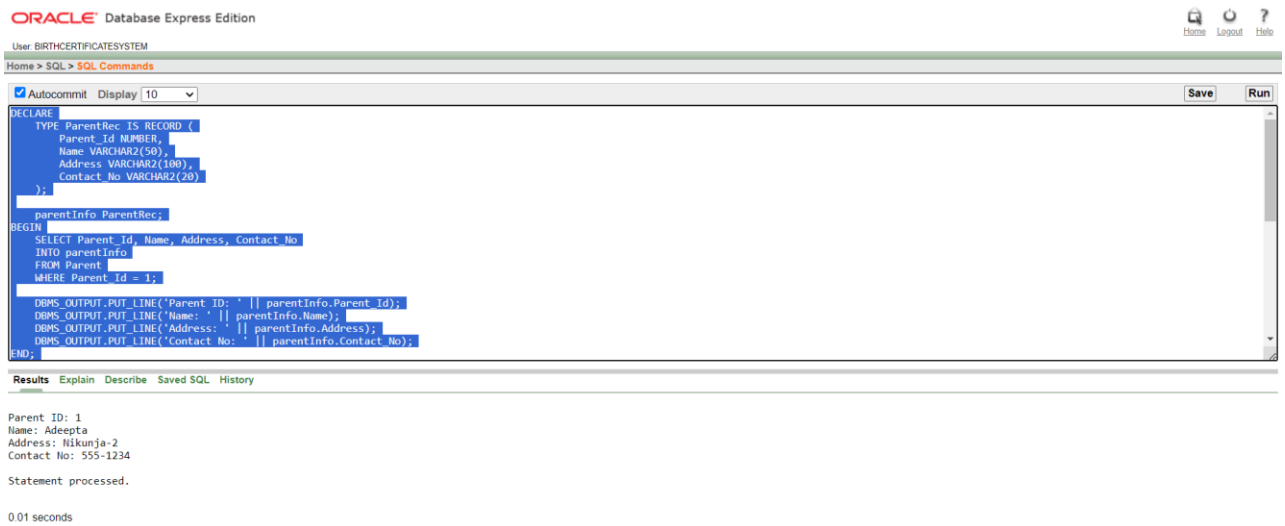
2. Create a PL/SQL record named "DoctorDetails" that can hold the attributes of a doctor (Doctor_Id, Name, Specialization, Email, Contact_No). Use this record to fetch and display information about a specific doctor.

```
DECLARE
   TYPE DoctorRec IS RECORD (
      Doctor_Id NUMBER,
      Name VARCHAR2(50),
      Specialization VARCHAR2(50),
      Email VARCHAR2(100),
      Contact_No VARCHAR2(20)
   );

   doctorDetails DoctorRec;
BEGIN
   SELECT Doctor_Id, Name, Specialization, Email, Contact_No
   INTO doctorDetails
   FROM Doctor
   WHERE Doctor_Id = 1;

   DBMS_OUTPUT.PUT_LINE('Doctor ID: ' || doctorDetails.Doctor_Id);
   DBMS_OUTPUT.PUT_LINE('Name: ' || doctorDetails.Name);
   DBMS_OUTPUT.PUT_LINE('Specialization: ' || doctorDetails.Specialization);
   DBMS_OUTPUT.PUT_LINE('Email: ' || doctorDetails.Email);
   DBMS_OUTPUT.PUT_LINE('Contact No: ' || doctorDetails.Contact_No);
END;
/
```



3. Create a PL/SQL record named "ParentInfo" that can hold the attributes of a parent (Parent_Id, Name, Address, Contact_No). Use this record to fetch and display information about a specific parent.

```
DECLARE
   TYPE ParentRec IS RECORD (
      Parent_Id NUMBER,
      Name VARCHAR2(50),
      Address VARCHAR2(100),
      Contact_No VARCHAR2(20)
   );

   parentInfo ParentRec;
BEGIN
   SELECT Parent_Id, Name, Address, Contact_No
   INTO parentInfo
   FROM Parent
   WHERE Parent_Id = 1;

   DBMS_OUTPUT.PUT_LINE('Parent ID: ' || parentInfo.Parent_Id);
   DBMS_OUTPUT.PUT_LINE('Name: ' || parentInfo.Name);
   DBMS_OUTPUT.PUT_LINE('Address: ' || parentInfo.Address);
   DBMS_OUTPUT.PUT_LINE('Contact No: ' || parentInfo.Contact_No);
END;
/
```



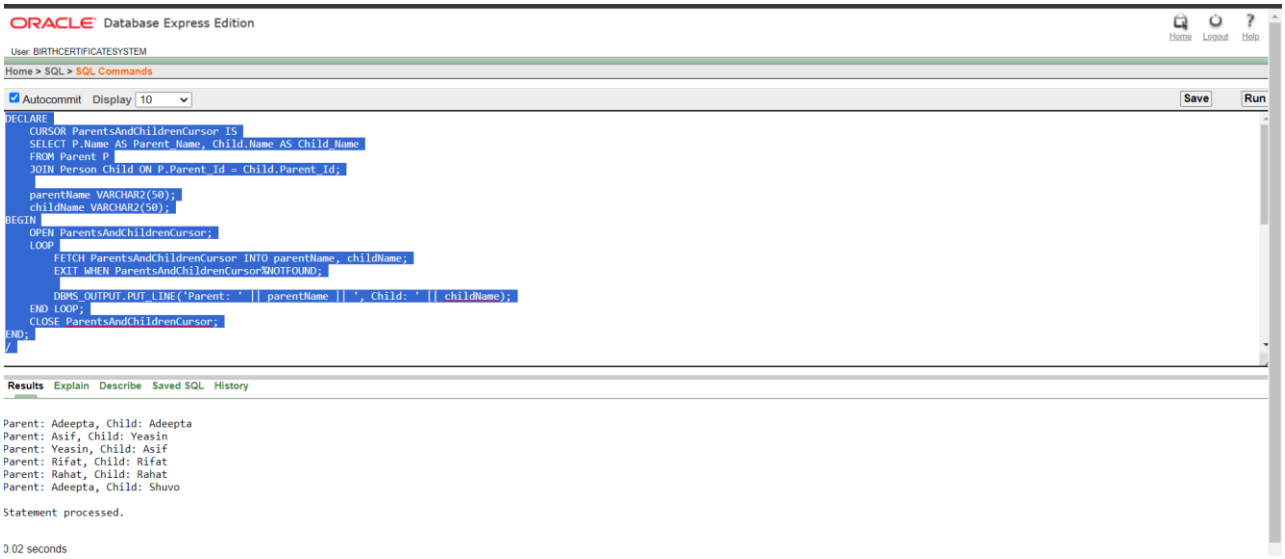
Cursor:

1.Create a PL/SQL cursor named "MalePatientsCursor" that retrieves the names of male patients from the Person table.

```
DECLARE
   CURSOR MalePatientsCursor IS
   SELECT Name
   FROM Person
   WHERE Gender = 'Male';

   patientName VARCHAR2(50);
BEGIN
   OPEN MalePatientsCursor;
   LOOP
      FETCH MalePatientsCursor INTO patientName;
      EXIT WHEN MalePatientsCursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Male patient name: ' || patientName);
   END LOOP;
   CLOSE MalePatientsCursor;
END;
/
```

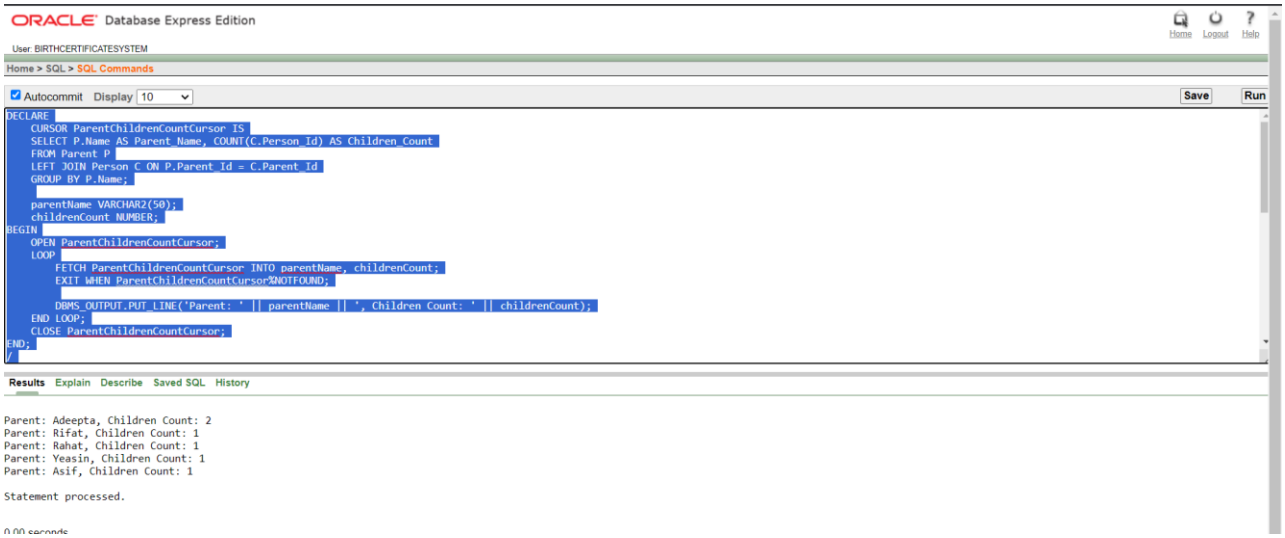
2. Create a PL/SQL cursor named "ParentsAndChildrenCursor" that retrieves the names of parents and their corresponding children's names from the Parent and Person tables.

```
DECLARE
   CURSOR ParentsAndChildrenCursor IS
   SELECT P.Name AS Parent_Name, Child.Name AS Child_Name
   FROM Parent P
   JOIN Person Child ON P.Parent_Id = Child.Parent_Id;

   parentName VARCHAR2(50);
   childName VARCHAR2(50);
BEGIN
   OPEN ParentsAndChildrenCursor;
   LOOP
      FETCH ParentsAndChildrenCursor INTO parentName, childName;
      EXIT WHEN ParentsAndChildrenCursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Parent: ' || parentName || ', Child: ' || childName);
   END LOOP;
   CLOSE ParentsAndChildrenCursor;
END;
/
```



3. Create a PL/SQL cursor named "ParentChildrenCountCursor" that retrieves the count of children for each parent from the Parent and Person tables. Display the parent's name along with the count of their children.

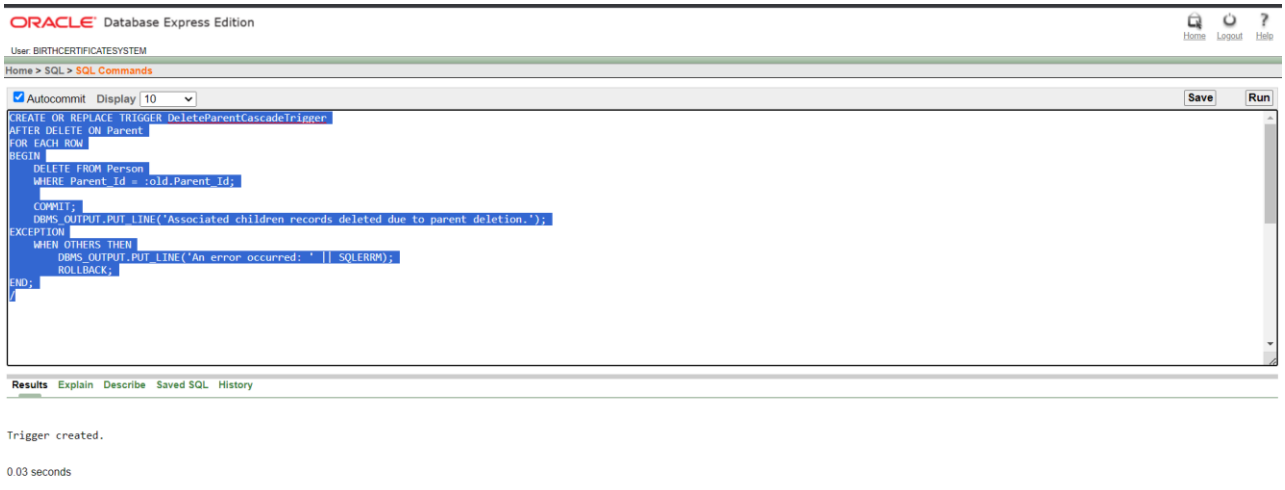
```
DECLARE
   CURSOR ParentChildrenCountCursor IS
   SELECT P.Name AS Parent_Name, COUNT(C.Person_Id) AS Children_Count
   FROM Parent P
   LEFT JOIN Person C ON P.Parent_Id = C.Parent_Id
   GROUP BY P.Name;

   parentName VARCHAR2(50);
   childrenCount NUMBER;
BEGIN
   OPEN ParentChildrenCountCursor;
   LOOP
      FETCH ParentChildrenCountCursor INTO parentName, childrenCount;
      EXIT WHEN ParentChildrenCountCursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Parent: ' || parentName || ', Children Count: ' || childrenCount);
   END LOOP;
   CLOSE ParentChildrenCountCursor;
END;
/
```



Trigger:

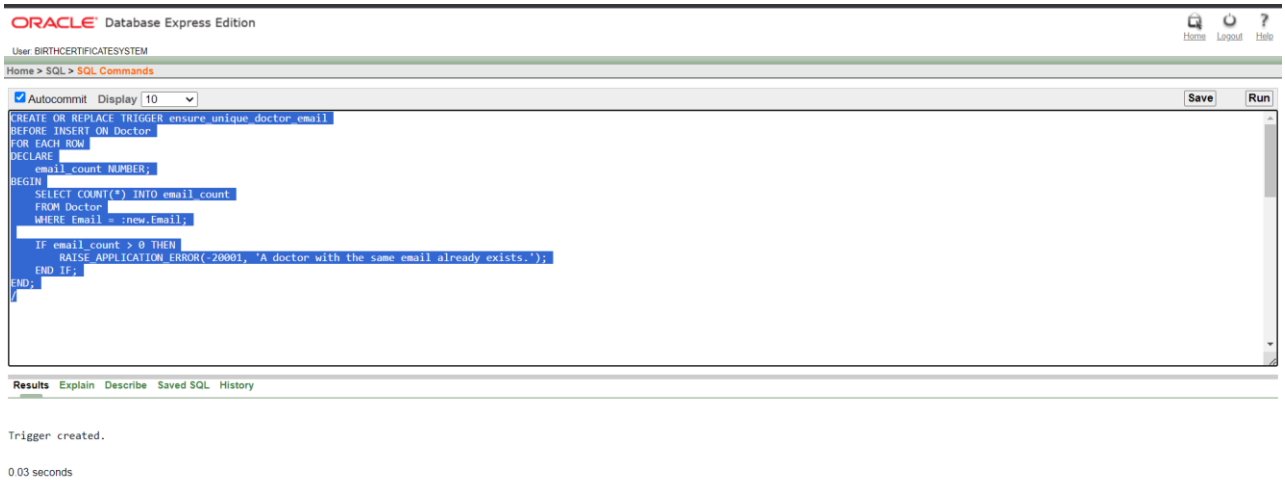
1. Create a PL/SQL trigger named "DeleteParentCascadeTrigger" that automatically deletes the associated children records in the Person table when a parent record is deleted from the Parent table.

```
CREATE OR REPLACE TRIGGER DeleteParentCascadeTrigger
AFTER DELETE ON Parent
FOR EACH ROW
BEGIN
   DELETE FROM Person
   WHERE Parent_Id = :old.Parent_Id;

   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Associated children records deleted due to parent deletion.');
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
      ROLLBACK;
END;
/
```



2. Create a trigger named ensure_unique_doctor_email that prevents the insertion of a new doctor into the Doctor table if their email already exists in the table.

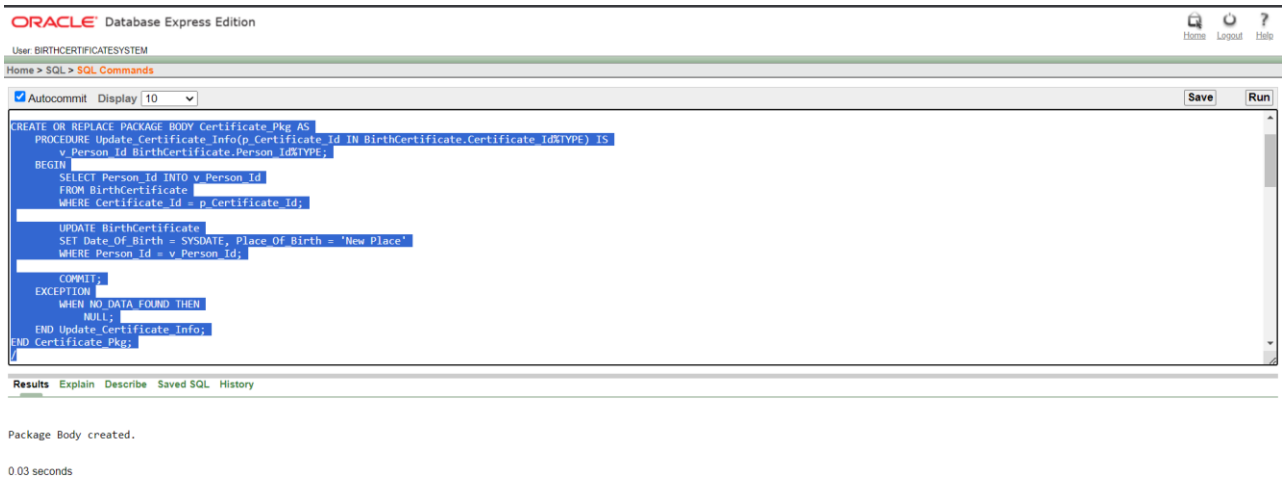
```
CREATE OR REPLACE TRIGGER ensure_unique_doctor_email
BEFORE INSERT ON Doctor
FOR EACH ROW
DECLARE
   email_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO email_count
   FROM Doctor
   WHERE Email = :new.Email;

   IF email_count > 0 THEN
      RAISE_APPLICATION_ERROR(-20001, 'A doctor with the same email already exists.');
   END IF;
END;
/
```



Package:

1.Create a PL/SQL package named Certificate_Pkg with a procedure named Update_Certificate_Info that takes a Certificate_Id as input and updates the place of birth and date of birth in the BirthCertificate table for the corresponding person.

```
CREATE OR REPLACE PACKAGE Certificate_Pkg AS
   PROCEDURE Update_Certificate_Info(p_Certificate_Id IN BirthCertificate.Certificate_Id%TYPE);
END Certificate_Pkg;
/

CREATE OR REPLACE PACKAGE BODY Certificate_Pkg AS
   PROCEDURE Update_Certificate_Info(p_Certificate_Id IN BirthCertificate.Certificate_Id%TYPE) IS
      v_Person_Id BirthCertificate.Person_Id%TYPE;
   BEGIN
      SELECT Person_Id INTO v_Person_Id
      FROM BirthCertificate
      WHERE Certificate_Id = p_Certificate_Id;

      UPDATE BirthCertificate
      SET Date_Of_Birth = SYSDATE, Place_Of_Birth = 'New Place'
      WHERE Person_Id = v_Person_Id;

      COMMIT;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         NULL;
   END Update_Certificate_Info;
END Certificate_Pkg;
/
```



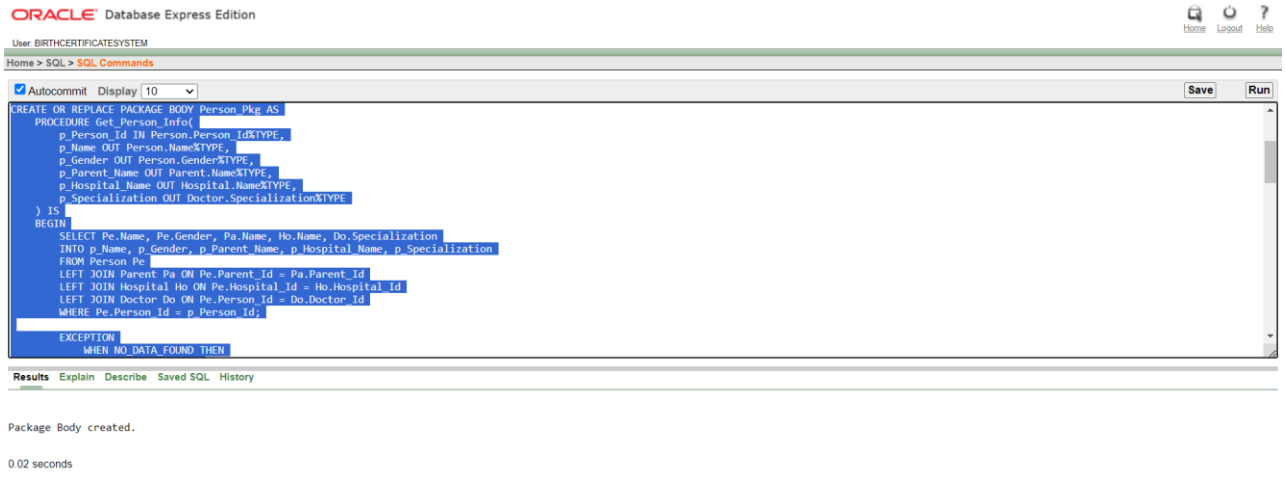
2. Create a PL/SQL package named Hospital_Pkg with a function named Get_Doctor_Count that takes a Hospital_Id as input and returns the number of doctors working at that hospital.

```
CREATE OR REPLACE PACKAGE Hospital_Pkg AS
   FUNCTION Get_Doctor_Count(p_Hospital_Id IN Hospital.Hospital_Id%TYPE) RETURN NUMBER;
END Hospital_Pkg;
/

CREATE OR REPLACE PACKAGE BODY Hospital_Pkg AS
   FUNCTION Get_Doctor_Count(p_Hospital_Id IN Hospital.Hospital_Id%TYPE) RETURN NUMBER IS
      v_Doctor_Count NUMBER;
   BEGIN
      SELECT COUNT(*) INTO v_Doctor_Count
      FROM Doctor
      WHERE Doctor_Id = p_Hospital_Id; -- Assuming Doctor_Id is the foreign key in the Hospital table

      RETURN v_Doctor_Count;
   END Get_Doctor_Count;
END Hospital_Pkg;
/
```

3. Create a PL/SQL package named Person_Pkg with a procedure named Get_Person_Info that takes a Person_Id as input and returns the name, gender, parent name, and hospital name of the person. If the person is a doctor, also return their specialization.

```
CREATE OR REPLACE PACKAGE Person_Pkg AS
   PROCEDURE Get_Person_Info(
      p_Person_Id IN Person.Person_Id%TYPE,
      p_Name OUT Person.Name%TYPE,
      p_Gender OUT Person.Gender%TYPE,
      p_Parent_Name OUT Parent.Name%TYPE,
      p_Hospital_Name OUT Hospital.Name%TYPE,
      p_Specialization OUT Doctor.Specialization%TYPE
   );
END Person_Pkg;
/

CREATE OR REPLACE PACKAGE BODY Person_Pkg AS
```

```
    PROCEDURE Get_Person_Info(
        p_Person_Id IN Person.Person_Id%TYPE,
        p_Name OUT Person.Name%TYPE,
        p_Gender OUT Person.Gender%TYPE,
        p_Parent_Name OUT Parent.Name%TYPE,
        p_Hospital_Name OUT Hospital.Name%TYPE,
        p_Specialization OUT Doctor.Specialization%TYPE
    ) IS
    BEGIN
        SELECT Pe.Name, Pe.Gender, Pa.Name, Ho.Name, Do.Specialization
        INTO p_Name, p_Gender, p_Parent_Name, p_Hospital_Name, p_Specialization
        FROM Person Pe
        LEFT JOIN Parent Pa ON Pe.Parent_Id = Pa.Parent_Id
        LEFT JOIN Hospital Ho ON Pe.Hospital_Id = Ho.Hospital_Id
        LEFT JOIN Doctor Do ON Pe.Person_Id = Do.Doctor_Id
        WHERE Pe.Person_Id = p_Person_Id;

        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                p_Name := NULL;
                p_Gender := NULL;
                p_Parent_Name := NULL;
                p_Hospital_Name := NULL;
                p_Specialization := NULL;
    END Get_Person_Info;
END Person_Pkg;
/
```



Relation Algebra:

1. Find out his Person_Id whose User_name is Adeepta
   $\prod$*person_id*(σ*User_name*= “Adeepta” (*Person*))

2. Find out the Person whose Person_address is Badda
   $\prod$*Person_name*(σ*Person_address*= “Mirpur” (*Person*))

3. Find out the Person address whose Contact_number is “458856”.
   $\prod$*Person_address*(σ*Contact_number*= “458856” (*Person*))

4. Find out the Person name whose Person_address is Mirpur.
   $\prod$*Person_name*(σ*Person_address*= “Mirpur” (*Person*))

5. Find out the Birth Certificate which name is Adeepta.
   $\prod$*Certificate*(σ*Person_name*= “Adeepta” (*Birth Certificate*)).

Conclusion:

Findings:

a. The "Person" table keeps track of details about specific people, such as their name, date of birth, and gender.
b. The "Parent" table contains information on parents, such as their names, addresses, and phone numbers.
c. Information regarding birth certificates, including their unique identity, associated person ID, date and place of birth, and parents' names, is kept in the "Birth Certificate" table.

d. The "Doctor" table contains data on the doctors or healthcare providers who assisted in the delivery of the child, including their name, specialization, and contact details.
e. "Hospital" table keeps details on the hospital or healthcare facility where the birth occurred, such as its name, address, and phone number.

Potential Future Work:

1. Improve Data Integrity: Add more constraints, including NOT NULL constraints or unique constraints, to protect the tables' data integrity.

2. Strengthen Relationship Management: Examine and improve table relationships while making sure that appropriate primary key and foreign key constraints are in place to preserve data consistency.

3. Increase Functionality: Take into account adding extra tables or features to offer broader functionality, such as keeping track of appointments, logging medical history, or handling financial data.

4. Apply data validation rules and tests to ensure that the provided data complies with stated requirements, such as accurate dates or acceptable email formats.

5. Improve User Interface: Make the user interface easier to use and intuitive so that data entering, searching, and reporting are all made possible.

6. Put Security Measures in Place: Boost Data Security with Appropriate Access Controls, Encryption, and Data Backup Techniques.

7. Performance Optimization: By making the proper indexes and optimizing SQL statements, analyses and improve query performance.

These suggested changes are meant to improve the project's usability, data integrity, functionality, and security. Consider variables including feasibility, time limits, and the particular requirements of the project when deciding which topics to priorities for the Final Term.