HOSPITAL MANAGEMENT SYSTEM

DBMS PROJECT

BE Second Year
Group No: 2CO4
SUBMITTED TO :
Dr. Anil Vashisht

SUBMITTED BY:

Lavisha Lakhmani (102103109)
Shaina (102103118)
Varshitha Pentu (102103107)
Jaskiran Walia (102283023)

AIM:-

Our project is based on Hospital Management System. The primary aim of a hospital management system is to provide a digital platform to efficiently manage and streamline the operations and services of a hospital. It includes managing patient information, appointments, electronic health records (EHRs), billing and payment, inventory management, and resource utilization.

The system is designed to improve the quality of healthcare services by enabling healthcare providers to make informed decisions based on real-time data and analytics. It also helps to minimize errors and reduce the workload of hospital staff, allowing them to focus on delivering high-quality patient care. Additionally, a hospital management system aims to enhance patient satisfaction by providing a seamless experience from admission to discharge. It ensures that patients receive timely and accurate information about their health status and treatment plan, and helps to minimize waiting times and delays.

Overall, the goal of a hospital management system is to improve the efficiency, effectiveness, and quality of healthcare services while reducing costs and enhancing patient outcomes.

DESCRIPTION:

This project involves the development of a database management system for a healthcare organization. The database consists of five tables: Employee, Doctor, Nurse, Patient, and Department. The tables are interrelated and allow the healthcare organization to efficiently manage its operations.

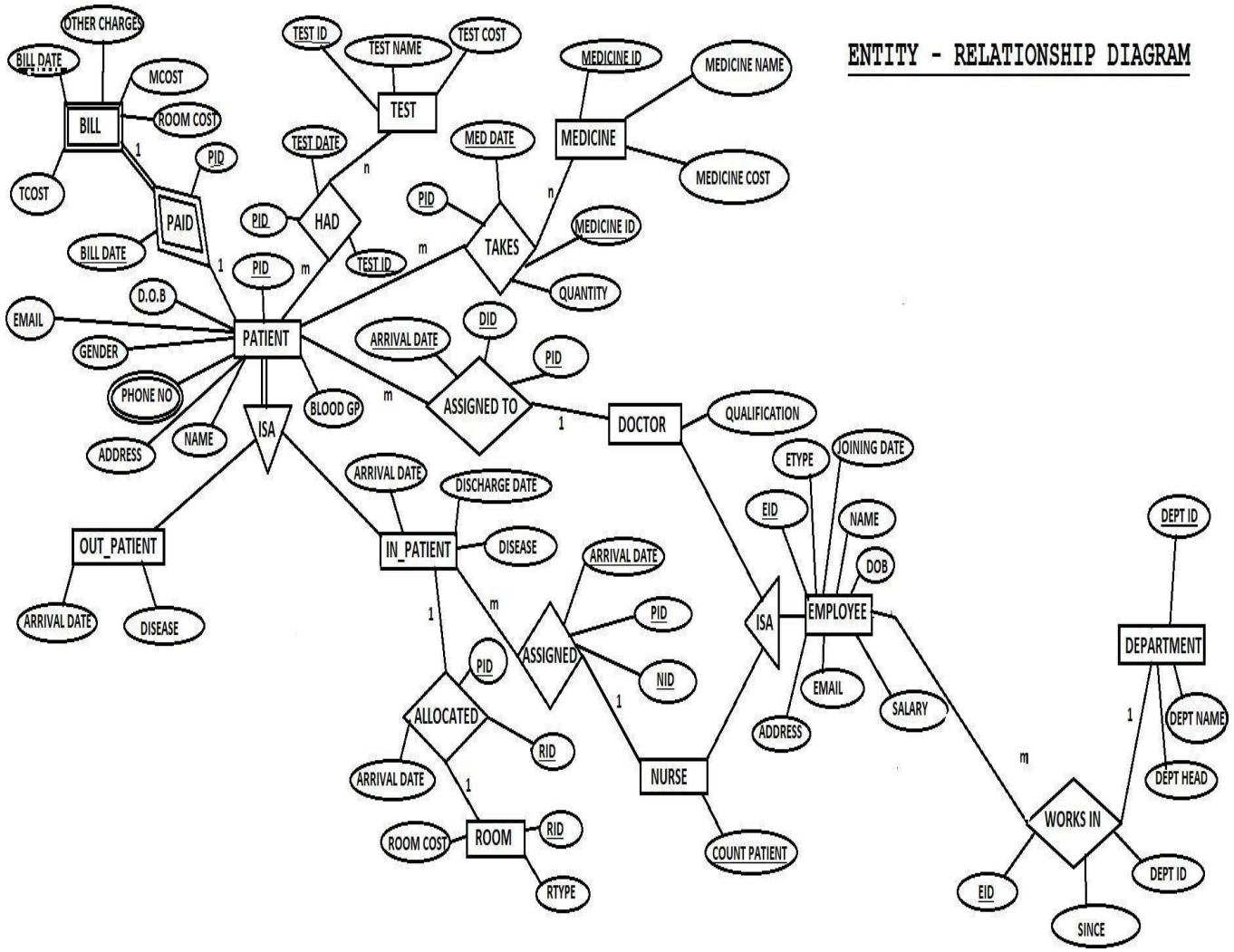
The Employee table contains information about all employees of the organization, including their ID, name, designation, and contact information. The Doctor and Nurse tables are a subset of the Employee table and contain information about the healthcare professionals working in the organization. The Doctor table includes information about the doctors' specialization and their area of expertise, while the Nurse table contains information about the nurses' shift schedules and areas of responsibility.

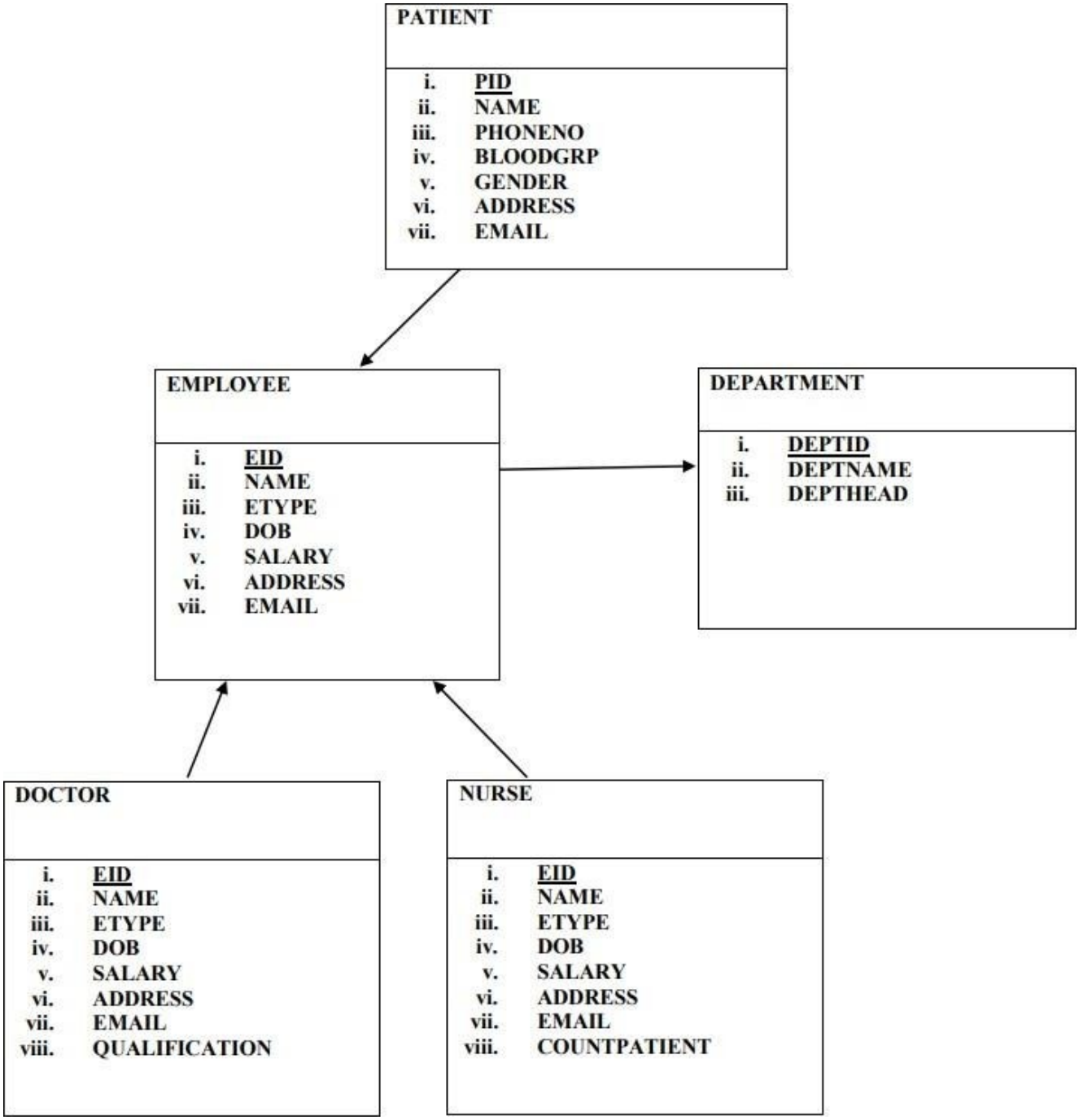
The Patient table contains information about all the patients who visit the organization, including their ID, name, age, and contact information. This table is used to maintain the medical history of the patients, including their diagnosis, treatment, and follow-up care.

The Department table contains information about the various departments in the organization, such as the emergency department, surgery department, and pharmacy department. This table is used to manage the resources and staff of each department.

The project uses technologies such as SQL and PL/SQL to perform various operations on the database. SQL is used to create, retrieve, update, and delete data from the database. PL/SQL is used to create stored procedures and functions that can be executed on the database. The database management system developed in this project allows the healthcare organization to efficiently manage its operations, including patient care, resource management, and employee management. The system provides a centralized platform for storing and retrieving patient data, enabling the healthcare professionals to make informed decisions about patient care. Furthermore, it helps the organization to manage its resources effectively, ensuring that the right staff are assigned to the right department and that the right resources are available when needed.

ER- DIAGRAM


ENTITY - RELATIONSHIP DIAGRAM
OTHER CHARGES
BILL DATE
MCOST
BILL
ROOM COST
TCOST
PID
PAID
BILL DATE
TEST ID
TEST NAME
TEST COST
TEST
TEST DATE
HAD
PID
TEST ID
MED DATE
TAKES
MEDICINE ID
MEDICINE NAME
MEDICINE
MEDICINE COST
QUANTITY
D.O.B
EMAIL
GENDER
PHONE NO
ADDRESS
NAME
PATIENT
BLOOD GP
ISA
ARRIVAL DATE
DID
PID
ASSIGNED TO
DOCTOR
QUALIFICATION
OUT_PATIENT
ARRIVAL DATE
DISEASE
IN_PATIENT
DISCHARGE DATE
DISEASE
ARRIVAL DATE
ASSIGNED
PID
NID
ALLOCATED
PID
RID
ARRIVAL DATE
ROOM
RID
ROOM COST
RTYPE
NURSE
COUNT PATIENT
ISA
ETYPE
JOINING DATE
EID
NAME
DOB
EMPLOYEE
EMAIL
SALARY
ADDRESS
DEPT ID
DEPARTMENT
DEPT NAME
DEPT HEAD
WORKS IN
DEPT ID
SINCE
EID


PATIENT
i. PID
ii. NAME
iii. PHONENO
iv. BLOODGRP
v. GENDER
vi. ADDRESS
vii. EMAIL
EMPLOYEE
i. EID
ii. NAME
iii. ETYPE
iv. DOB
v. SALARY
vi. ADDRESS
vii. EMAIL
DEPARTMENT
i. DEPTID
ii. DEPTNAME
iii. DEPTHEAD
DOCTOR
i. EID
ii. NAME
iii. ETYPE
iv. DOB
v. SALARY
vi. ADDRESS
vii. EMAIL
viii. QUALIFICATION
NURSE
i. EID
ii. NAME
iii. ETYPE
iv. DOB
v. SALARY
vi. ADDRESS
vii. EMAIL
viii. COUNTPATIENT

Normalization Process:

1NF- First Normal Form

If a relation contains a composite or multi-valued attribute, it violates the first normal form, or the relationship is in the first normal form if it does not contain any composite or multi-valued attribute. A relation is in its first normal form if every attribute in that relation is singled valued attribute. A table is in 1 NF iff:

1. There are only Single Valued Attributes.
2. Attribute Domain does not change.
3. There is a unique name for every Attribute/Column.
4. The order in which data is stored does not matter.

PATIENT TABLE:

pid – pid column satisfies all the above conditions.
Patient name –Name column satisfies all the above conditions.
blood grp– blood grp column satisfies all the above conditions.

Phone No – Here phone number is a multivalued column. To get our table in a 1NF form we need to make it a single-valued column.

Gender-gender column satisfies all the above conditions.

Address-address column satisfies all the above conditions.

Email-email column satisfies all the above condition

PATIENT

PID	NAME	BLOODGRP	PHONENO	GENDER	ADDRESS	EMAIL

PID	NAME	BLOODGRP	PHONENO1	PHONENO2	GENDER	ADDRESS	EMAIL

EMPLOYEE TABLE

Eid – eid column satisfies all the above conditions.

etype – etype column satisfies all the above conditions.

Name – name column satisfies all the above conditions. Dob – dob column satisfies all the above conditions.

salary-salary column satisfies all the above conditions.

Address-address column satisfies all the above conditions. Email-email column satisfies all the above conditions.

All the attributes satisfy the above 4 conditions. Our EMPLOYEE table is alreadyin First Normal Form

EMPLOYEE

EID	NAME	ETYPE	DOB	GENDER	SALARY	EMAIL

DOCTOR TABLE

All the attributes satisfy the above 4 conditions. Our Doctor table is already in First Normal Form.

DOCTOR

EID	NAME	ETYPE	DOB	GENDER	SALARY	QUALIFICATION	EMAIL

NURSE Table

All the attributes satisfy the above 4 conditions. Our Nurse table is already in First Normal Form.

NURSE

EID	NAME	ETYPE	DOB	GENDER	SALARY	COUNTPATIENT	EMAIL

DEPARTMENT

All the attributes satisfy the above 4 conditions. Our Department table is already in First Normal Form.

DEPTID	DEPTNAME	DEPTHEAD

Now we have our database schema normalized to First Normal Form

2NF- Second Normal Form

To be in the second normal form, a relation must be in the first normal form and the relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes that are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Patient Table

PATIENT

PID	NAME

PID	PHONE_NO_1	PHONE_NO_2

PID	ADDRESS	GENDER	EMAIL

EmployeeTable

Employee

EID	NAME	ETYPE	DOB	GENDER	SALARY	EMAIL

DOCTOR Table

DOCTOR

EID	NAME	ETYPE	DOB	GENDER	SALARY	QUALIFICATION	EMAIL

NURSE Table

NURSE

EID	NAME	ETYPE	DOB	GENDER	SALARY	COUNTPATIENT	EMAIL

DEPARTMENT Table

DEPARTMENT

DEPTID	DEPTNAME	DEPTHEAD

3NF- Third Normal Form

A relation that is in First and Second Normal Form and in which no non-primary key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF). If A->B and B->C are two FDs then A->C is called transitive dependency.

Patient

PID	NAME

PID	PHONE_NO_1	PHONE_NO_2

PID	ADDRESS	GENDER	EMAIL

EmployeeTable

Employee

EID	NAME	ETYPE	DOB	GENDER	SALARY	EMAIL

DOCTOR Table

DOCTOR

EID	NAME	ETYPE	DOB	GENDER	SALARY	QUALIFICATION	EMAIL

NURSE Table

NURSE

EID	NAME	ETYPE	DOB	GENDER	SALARY	COUNTPATIENT	EMAIL

DEPARTMENT Table

DEPARTMENT

DEPTID	DEPTNAME	DEPTHEAD

PLSQL COMMANDS TO CREATE TABLE

```
CREATE TABLE  patient
(  pid int,
fname varchar(20) not null,  lname varchar(20),
gender varchar(6) not null,
dob date not null,  blood_group varchar(3),  doc_id int,
HNo varchar(10),
street varchar(20),  city varchar(16),  state varchar(20),  email varchar(30),  Primary Key(pid)
);


CREATE TABLE  Employee
(
    empid int,
   fname varchar(20) not null,  mname varchar(20),
   lname varchar(20),
   gender varchar(6) not null,
   emptype varchar(20) not null,  Hno varchar(10),
   street varchar(20),  city varchar(20), state  varchar(20),
   date_of_joining date,
   email varchar(30),  deptid int,
   since date,  date_of_birth date,
      PRIMARY key(empid)
      );

CREATE TABLE department
(  deptid int,
dname varchar(20) not null,  dept_headid int(10),
PRIMARY key(deptid)
);

CREATE TABLE nurse_assigned
(
nid int,
countpatient int,
PRIMARY KEY(nid),
FOREIGN KEY(nid) REFERENCES employee(empid)
);

CREATE TABLE doctor
(
doc_id int,
qualification varchar(20),
PRIMARY KEY(doc_id),
FOREIGN KEY(doc_id) REFERENCES employee(empid) ON DELETE CASCADE
);
```

PLSQL COMMANDS TO CREATE TRIGGERS

Trigger for the patient table to ensure that the blood group entered is valid:

```
CREATE TRIGGER tr_patient_blood_group
BEFORE INSERT OR UPDATE ON patient
FOR EACH ROW
BEGIN
   IF NEW.blood_group NOT IN ('A+', 'A-', 'B+', 'B-', 'O+', 'O-', 'AB+', 'AB-') THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Invalid blood group';
   END IF;
END;
```

Trigger for the department table to ensure that the department head ID exists in the Employee table:

```
CREATE TRIGGER tr_department_headid
BEFORE INSERT OR UPDATE ON department
FOR EACH ROW
BEGIN
   IF NEW.dept_headid NOT IN (SELECT empid FROM Employee) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Department head ID does not exist in the Employee table';
   END IF;
END;
```

Trigger for the Employee table to ensure that the employee's date of joining is not in the future:

```
CREATE TRIGGER tr_employee_date_of_joining
BEFORE INSERT OR UPDATE ON Employee
FOR EACH ROW
BEGIN
   IF NEW.date_of_joining > NOW() THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Date of joining cannot be in the future';
   END IF;
END;
```

Trigger for the nurse_assigned table to ensure that the count of patients assigned is not negative:

```
CREATE TRIGGER tr_nurse_assigned_countpatient
BEFORE INSERT OR UPDATE ON nurse_assigned
FOR EACH ROW
BEGIN
   IF NEW.countpatient < 0 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Count of patients assigned cannot be negative';
   END IF;
END;
```

Trigger for the doctor table to ensure that the qualification entered is valid:

```
CREATE TRIGGER tr_doctor_qualification
BEFORE INSERT OR UPDATE ON doctor
FOR EACH ROW
BEGIN
   IF NEW.qualification NOT IN ('MD', 'MBBS', 'BAMS', 'BHMS') THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Invalid qualification';
   END IF;
END;
```

PLSQL COMMANDS FOR PROCEDURES, FUNCTIONS AND PACKAGES

```
CREATE OR REPLACE PACKAGE hospital_pkg AS

  -- Procedure to insert a new patient record
  PROCEDURE add_patient(
    fname IN patient.fname%TYPE,
    lname IN patient.lname%TYPE,
    gender IN patient.gender%TYPE,
    dob IN patient.dob%TYPE,
    blood_group IN patient.blood_group%TYPE,
    doc_id IN patient.doc_id%TYPE,
    HNo IN patient.HNo%TYPE,
    street IN patient.street%TYPE,
    city IN patient.city%TYPE,
    state IN patient.state%TYPE,
    email IN patient.email%TYPE
  );

  -- Procedure to update an existing patient record
  PROCEDURE update_patient(
    pid IN patient.pid%TYPE,
    fname IN patient.fname%TYPE,
    lname IN patient.lname%TYPE,
    gender IN patient.gender%TYPE,
    dob IN patient.dob%TYPE,
    blood_group IN patient.blood_group%TYPE,
    doc_id IN patient.doc_id%TYPE,
    HNo IN patient.HNo%TYPE,
    street IN patient.street%TYPE,
    city IN patient.city%TYPE,
    state IN patient.state%TYPE,
    email IN patient.email%TYPE
  );

  -- Procedure to delete a patient record
  PROCEDURE delete_patient(
    pid IN patient.pid%TYPE
  );

  -- Function to get a list of all patients
  FUNCTION get_all_patients RETURN SYS_REFCURSOR;

  -- Function to get a patient record by ID
  FUNCTION get_patient_by_id(
    pid IN patient.pid%TYPE
  ) RETURN SYS_REFCURSOR;
```

```
-- Procedure to insert a new employee record
  PROCEDURE add_employee(
     fname IN employee.fname%TYPE,
     mname IN employee.mname%TYPE,
     lname IN employee.lname%TYPE,
     gender IN employee.gender%TYPE,
     emptype IN employee.emptype%TYPE,
     Hno IN employee.Hno%TYPE,
     street IN employee.street%TYPE,
     city IN employee.city%TYPE,
     state IN employee.state%TYPE,
     date_of_joining IN employee.date_of_joining%TYPE,
     email IN employee.email%TYPE,
     deptid IN employee.deptid%TYPE,
     since IN employee.since%TYPE,
     date_of_birth IN employee.date_of_birth%TYPE
  );

  -- Procedure to update an existing employee record
  PROCEDURE update_employee(
     empid IN employee.empid%TYPE,
     fname IN employee.fname%TYPE,
     mname IN employee.mname%TYPE,
     lname IN employee.lname%TYPE,
     gender IN employee.gender%TYPE,
     emptype IN employee.emptype%TYPE,
     Hno IN employee.Hno%TYPE,
     street IN employee.street%TYPE,
     city IN employee.city%TYPE,
     state IN employee.state%TYPE,
     date_of_joining IN employee.date_of_joining%TYPE,
     email IN employee.email%TYPE,
     deptid IN employee.deptid%TYPE,
     since IN employee.since%TYPE,
     date_of_birth IN employee.date_of_birth%TYPE
  );

  -- Procedure to delete an employee record
  PROCEDURE delete_employee(
     empid IN employee.empid%TYPE
  );

  -- Function to get a list of all employees
  FUNCTION get_all_employees RETURN SYS_REFCURSOR;

  -- Function to get an employee record by ID
  FUNCTION get_employee_by_id(
     empid IN employee.empid%TYPE
  ) RETURN SYS_REFCURSOR;
```

```
-- Procedure to insert a new department record
  PROCEDURE insert_department(
  p_deptid IN department.deptid%TYPE,
  p_dname IN department.dname%TYPE,
  p_dept_headid IN department.dept_headid%TYPE
)
IS
BEGIN
  INSERT INTO department(deptid, dname, dept_headid)
  VALUES(p_deptid, p_dname, p_dept_headid);
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('New department record inserted successfully.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);
    ROLLBACK;
END;
```

PLSQL COMMANDS FOR VIEWS, INDEXES AND SEQUENCES

View for getting the name of the doctor for each patient:

```
CREATE VIEW patient_doctor AS
SELECT p.pid, CONCAT(e.fname, ' ', e.lname) AS doctor_name
FROM patient p
INNER JOIN doctor d ON p.doc_id = d.doc_id
INNER JOIN employee e ON d.doc_id = e.empid;
```

Index for the employee table to improve the performance of queries that filter by the date of joining:

```
CREATE INDEX idx_employee_date_of_joining ON employee (date_of_joining);
```

Sequence for generating unique department IDs:

```
CREATE SEQUENCE seq_deptid START WITH 1 INCREMENT BY 1;
```

View for getting the number of patients assigned to each nurse:

```
CREATE VIEW nurse_assigned_count AS
SELECT n.nid, n.countpatient, CONCAT(e.fname, ' ', e.lname) AS nurse_name
FROM nurse_assigned n
INNER JOIN employee e ON n.nid = e.empid;
```

Index for the patient table to improve the performance of queries that filter by the blood group:

```
CREATE INDEX idx_patient_blood_group ON patient (blood_group);
```