THE STATE UNIVERSITY OF ZANZIBAR
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

NAME	**REG NO**
ASHURA SALUM KHAMIS	BCS/18/23/011/TZ

TITLE OF PROJECT:HOSPITAL OR CLINICAL MANAGEMENT SYSTEM

1. System Description

This Hospital/Clinic Management System is designed to streamline the core operations of a medical facility, focusing on patient appointments and billing. The primary goal is to efficiently manage patient information, doctor schedules, appointment bookings, treatment records, and invoicing. This system aims to improve administrative efficiency and patient care coordination by enabling clinic staff to easily schedule appointments, track treatments, generate bills, and retrieve various reports.

The system's design prioritizes data integrity and ease of use, ensuring that critical medical and financial information is accurately recorded and readily accessible. It facilitates clear oversight of patient interactions from initial appointment scheduling through to treatment and final billing

2. Schema diagram

The database schema diagram is designed with five normalized tables: patients, doctors, appointments, treatments, and invoices.Each table serves a distinct purpose, and they are linked via primary and foreign keys to maintain data integrity and relationships.

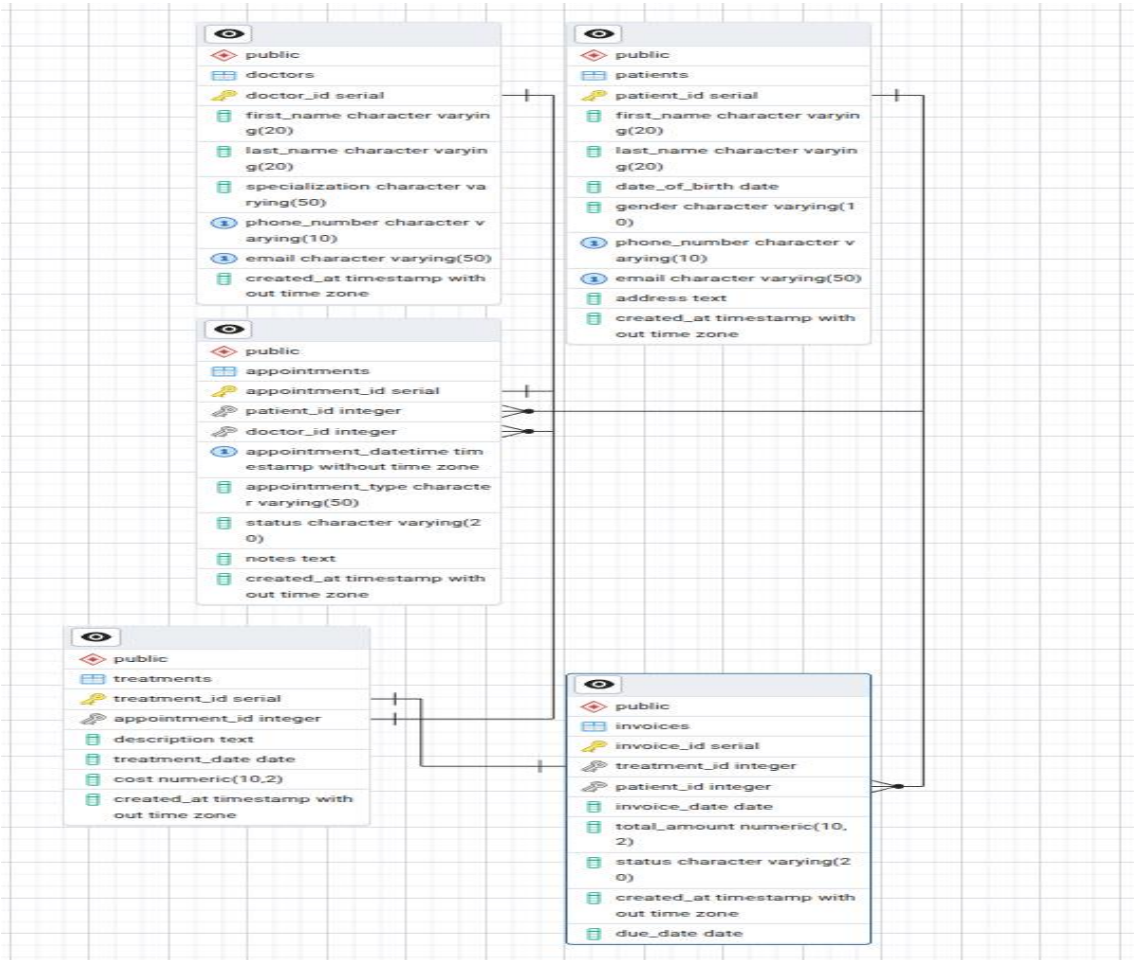

Figure 1 Schema Diagram for five table patients,doctors,appointments,treatments and invoices table

3. Procedure

schedule_appointment Procedure

This procedure is designed to register a new appointment in the system. It implements critical business logic, including transaction control (BEGIN, COMMIT, ROLLBACK) and conditional logic (IF...THEN...ELSE) to ensure data consistency. The procedure checks for the existence of the patient and doctor, and most importantly, it prevents double-booking a doctor at the same time slot. If an appointment slot is already taken or if the patient/doctor does not exist, the transaction is rolled back, and an informative error message is provided.

SQL script

Query | Query History

```
CREATE OR REPLACE PROCEDURE schedule_appointment(
    p_patient_id INT,
    p_doctor_id INT,
    p_appointment_datetime TIMESTAMP,
    p_appointment_type VARCHAR(50)
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_doctor_name VARCHAR(20);
    v_patient_name VARCHAR(20);
    v_appointment_exists BOOLEAN;
BEGIN

    SELECT first_name || ' ' || last_name INTO v_doctor_name FROM doctors WHERE doctor_id = p_doctor_id;
    SELECT first_name || ' ' || last_name INTO v_patient_name FROM patients WHERE patient_id = p_patient_id;

    IF v_doctor_name IS NULL THEN
        RAISE EXCEPTION 'Doctor with ID % does not exist.', p_doctor_id;
    END IF;

    IF v_patient_name IS NULL THEN
        RAISE EXCEPTION 'Patient with ID % does not exist.', p_patient_id;
    END IF;

    SELECT EXISTS (
        SELECT 1
        FROM appointments
        WHERE doctor_id = p_doctor_id
          AND appointment_datetime = p_appointment_datetime
    ) INTO v_appointment_exists;

    IF v_appointment_exists THEN
        RAISE EXCEPTION 'Appointment slot for Doctor % at % is already booked.', v_doctor_name, p_appointment_datetime;
    ELSE

        INSERT INTO appointments (patient_id, doctor_id, appointment_datetime, appointment_type, status)
        VALUES (p_patient_id, p_doctor_id, p_appointment_datetime, p_appointment_type, 'Scheduled');

        RAISE NOTICE 'Appointment scheduled successfully for Patient % with Doctor % on % for %.',
                     v_patient_name, v_doctor_name, p_appointment_datetime, p_appointment_type;
    END IF;

    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'An error occurred: %', SQLERRM;
        ROLLBACK;
        RAISE;
END;
$$;
```

Figure 2 SQL script for schedule_appointment Procedure

Output

```
141  CALL schedule_appointment(1, 1, '2025-06-07 10:00:00', 'New Consultation');
142
```

Data Output | Messages | Notifications

```
NOTICE:  Appointment scheduled successfully for Patient Ashura Khamis with Doctor Dr. Omar Mohd on 2025-06-07 10:00:00 for New Consultation.
NOTICE:  An error occurred: cannot commit while a subtransaction is active

ERROR:  cannot commit while a subtransaction is active
CONTEXT:  PL/pgSQL function schedule_appointment(integer,integer,timestamp without time zone,character varying) line 42 at COMMIT

SQL state: 2D000
```

Figure 3 Successful appointment booking

```
144
145  CALL schedule_appointment(2, 1, '2025-06-07 10:00:00', 'Follow-up');
146
147
```

Data Output | Messages | Notifications

```
NOTICE:  Appointment scheduled successfully for Patient Khadija Khamis with Doctor Dr. Omar Mohd on 2025-06-07 10:00:00 for Follow-up.
NOTICE:  An error occurred: cannot commit while a subtransaction is active

ERROR:  cannot commit while a subtransaction is active
CONTEXT:  PL/pgSQL function schedule_appointment(integer,integer,timestamp without time zone,character varying) line 42 at COMMIT

SQL state: 2D000
```

Figure 4 Attempt to book an already booked slot

```
149
150
151  CALL schedule_appointment(999, 1, '2025-06-09 10:00:00', 'New Consultation');
152
153
```

Data Output | Messages | Notifications

```
NOTICE:  An error occurred: Patient with ID 999 does not exist.

ERROR:  Patient with ID 999 does not exist.
CONTEXT:  PL/pgSQL function schedule_appointment(integer,integer,timestamp without time zone,character varying) line 19 at RAISE

SQL state: P0001
```

Figure 5 Attempt to book for a non-existent patient

```
157
158  CALL schedule_appointment(1, 999, '2025-06-09 10:00:00', 'New Consultation');
159
```

Data Output | Messages | Notifications

```
NOTICE:  An error occurred: Doctor with ID 999 does not exist.

ERROR:  Doctor with ID 999 does not exist.
CONTEXT:  PL/pgSQL function schedule_appointment(integer,integer,timestamp without time zone,character varying) line 15 at RAISE

SQL state: P0001
```

Figure 6 Attempt to book for a non-existent doctor

4. Functions

calculate_total_bill_per_patient Function

This function returns a single value, representing the total amount billed to a specific patient. It aggregates the total_amount from all invoices associated with the given patient.

SQL script

```
Query  Query History
168
169
170 CREATE OR REPLACE FUNCTION calculate_total_bill_per_patient(p_patient_id INT)
171 RETURNS NUMERIC(10, 2)
172 LANGUAGE plpgsql
173 AS $$
174 DECLARE
175     v_total_bill NUMERIC(10, 2) := 0;
176 BEGIN
177     SELECT SUM(i.total_amount)
178     INTO v_total_bill
179     FROM invoices i
180     WHERE i.patient_id = p_patient_id;
181     IF v_total_bill IS NULL THEN
182         RETURN 0;
183     END IF;
184
185     RETURN v_total_bill;
186 END;
187 $$;
188
```

Figure 7 Function of calculate_total_bill_per_patient Function

Output

```
190
191
192 SELECT calculate_total_bill_per_patient(1);
193
```

Data Output | Messages | Notifications

SQL

	calculate_total_bill_per_patient numeric
1	150000.00

Figure 8 function bill for patient id 1 calculate_total_bill_per_patient(1)

```
201
202 SELECT calculate_total_bill_per_patient(3);
203
204
```

Data Output | Messages | Notifications

SQL

	calculate_total_bill_per_patient numeric
1	0

Figure 9 function for patient_id 3 calculate_total_bill_per_patient(3)

get_patients_treated_by_doctor Function

This function returns a table of patient information, appointment details, and treatment costs for all patients seen by a specific doctor.

SQL script

```
CREATE OR REPLACE FUNCTION get_patients_treated_by_doctor(p_doctor_id INT)
RETURNS TABLE (
    patient_id INT,
    patient_first_name VARCHAR,
    patient_last_name VARCHAR,
    appointment_datetime TIMESTAMP,
    treatment_description TEXT,
    treatment_cost NUMERIC(10, 2)
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        p.patient_id,
        p.first_name,
        p.last_name,
        a.appointment_datetime,
        t.description,
        t.cost
    FROM patients p
    JOIN appointments a ON p.patient_id = a.patient_id
    JOIN treatments t ON a.appointment_id = t.appointment_id
    WHERE a.doctor_id = p_doctor_id
    ORDER BY a.appointment_datetime;
END;
$$;
```

Figure 10 SQL script get_patients_treated_by_doctor

Output

```
SELECT * FROM get_patients_treated_by_doctor(1);
```

Data Output | Messages | Notifications

	patient_id integer	patient_first_name character varying	patient_last_name character varying	appointment_datetime timestamp without time zone	treatment_description text	treatment_cost numeric
1	1	Ashura	Khamis	2025-06-05 10:00:00	General check-up and blood pressure measureme...	150000.00

Figure 11 Show the patient who treated by doctor who have doctor_id =1

```
SELECT * FROM get_patients_treated_by_doctor(999);
```

Data Output | Messages | Notifications

patient_id integer	patient_first_name character varying	patient_last_name character varying	appointment_datetime timestamp without time zone	treatment_description text	treatment_cost numeric

Figure 12 Show there is no existance of doctor_id=999

6. Cursors

process_overdue_invoices_and_notify Procedure

This procedure demonstrates the effective use of a cursor to iterate over a specific subset of data: currently pending and overdue invoices. For each such invoice, it generates a notification message. This logic could easily be extended to send emails, update a status field, or perform other batch operations.

SQL script

Query Query History

```
242
243 CREATE OR REPLACE PROCEDURE process_overdue_invoices_and_notify()
244 LANGUAGE plpgsql
245 AS $$
246 DECLARE
247     invoice_rec RECORD;
248     overdue_count INT := 0;
249     invoice_cursor CURSOR FOR
250         SELECT
251             i.invoice_id,
252             p.first_name || ' ' || p.last_name AS patient_name,
253             i.total_amount,
254             i.invoice_date,
255             i.due_date,
256             i.status
257         FROM invoices i
258         JOIN patients p ON i.patient_id = p.patient_id
259         WHERE i.status = 'Pending' AND i.due_date < CURRENT_DATE;
260 BEGIN
261     RAISE NOTICE '--- Starting Overdue Invoice Processing ---';
262
263     OPEN invoice_cursor;
264
265     LOOP
266         FETCH invoice_cursor INTO invoice_rec;
267         EXIT WHEN NOT FOUND;
268
269         overdue_count := overdue_count + 1;
270
271         RAISE NOTICE 'Invoice #% for Patient "%" (Amount: %.2f, Due: %) is OVERDUE and still %.',
272                      invoice_rec.invoice_id,
273                      invoice_rec.patient_name,
274                      invoice_rec.total_amount,
275                      invoice_rec.due_date,
276                      invoice_rec.status;
277
278     CLOSE invoice_cursor;
279
280     IF overdue_count = 0 THEN
281         RAISE NOTICE 'No overdue invoices found today.';
282     ELSE
283         RAISE NOTICE 'Processed % overdue invoices.', overdue_count;
284     END IF;
285
286     RAISE NOTICE '--- Overdue Invoice Processing Finished ---';
287
288 END;
289 $$;
```

Figure 13 cursor for process_overdue_invoices_and_notify

Output

```
293
294  UPDATE invoices SET due_date = '2025-06-01' WHERE invoice_id = 2;
295  CALL process_overdue_invoices_and_notify();
296
297
```

Data Output | Messages | Notifications

```
NOTICE:  --- Starting Overdue Invoice Processing ---
NOTICE:  Invoice #2 for Patient "Khadija Khamis" (Amount: 200000.00.2f, Due: 2025-06-01) is OVERDUE and still Pendi
NOTICE:  Processed 1 overdue invoices.
NOTICE:  --- Overdue Invoice Processing Finished ---
CALL

Query returned successfully in 161 msec.
```

Figure 14 Make one invoice overdue for testing

```
301  SELECT invoice_id, patient_id, total_amount, due_date, status FROM invoices WHERE invoice_id = 2;
```

Data Output | Messages | Notifications

	invoice_id [PK] integer	patient_id integer	total_amount numeric (10,2)	due_date date	status character varying (20)
1	2	2	200000.00	2025-06-01	Pending

Figure 15 Check the table directly if is update the status