DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES2201800157 SAKSHI VATTIKUTI

CRIME RECORD MANAGEMENT SYSTEM

A Criminal Record Management system is used to maintain and keep track of criminals and the crimes committed by them .

This database model is created with an assumption that one FIR can have only one victim and accused , however one victim or accused can have multiple FIRs filed.

An FIR filed can have only one petitioner , but a single petitioner can file any number of FIRs.

Transactions for inserting and updating are created , to allow a user to file an FIR and to be able to update victim , accused and crime details.

Once an FIR is filed , it is automatically assigned an officer , who assigns a case to it .

An officer can handle many cases .

A trigger is fired if the user enters the same FIR details , or accused or victim details.

In case the victim details are not known and wish to be updated later on , if the newly inserted victim or accused detail repeats , the new row is deleted and the fir is updated with the existing details.

Retrieval queries for retrieving accused , victim and crime details are written to filter out the cases on user's choice.

INTRODUCTION

MINIWORLD ENTITIES

A. **FIR:** First Information Report is a document filed at the police station , with minimum crime details.The FIR has an ID , incident place , incident time ,incident date, lodge time ,lodge place,lodge date and the crime category.
B. **Petitioner** : A person who files an FIR is called the petitioner . He/she will have to provide details such as name , address ,contact , age and gender while filing an FIR.
One petitioner can file multiple FIRs.
C. **Accused** : The person who is accused of a crime is called accused. An accused has an ID , name , aadhar number , contact , address , alias , gender , age and identification.
The status of an accused can be ARRESTED ,DEAD , BAILED and so on .
All the accused may or may not be known at the time of lodging a complaint , so the petitioner is required to provide only the accused gender while filing an FIR.
Later when the accused details are known , they can be updated ,provided they know the FIR id.
D. **Victim** : A person who has been attacked in the incident is a victim.The victim has an ID, name , contact , address , gender and age.Victim details may or may not be known at the time of lodging a complaint , so the petitioner is asked only for the victim gender while filing an FIR. Later when the victim details are known , they can be updated , provided they know the FIR id.
E. **Officer** : An officer is the person who looks into the FIR and assigns a case to it .An officer can assign multiple cases.Each officer has a name , id and a rank.
F. **Case**: Case is the description of the crime that has occurred.Each case has a case status as OPEN or CLOSED and complete details about the crime .

RELATIONS

A. Petitioner:FIR → 1:N

B. FIR:Case → 1:1
C. Officer:Case → 1:N
D. Victim:FIR → 1:N
E. Accused:FIR → 1:N

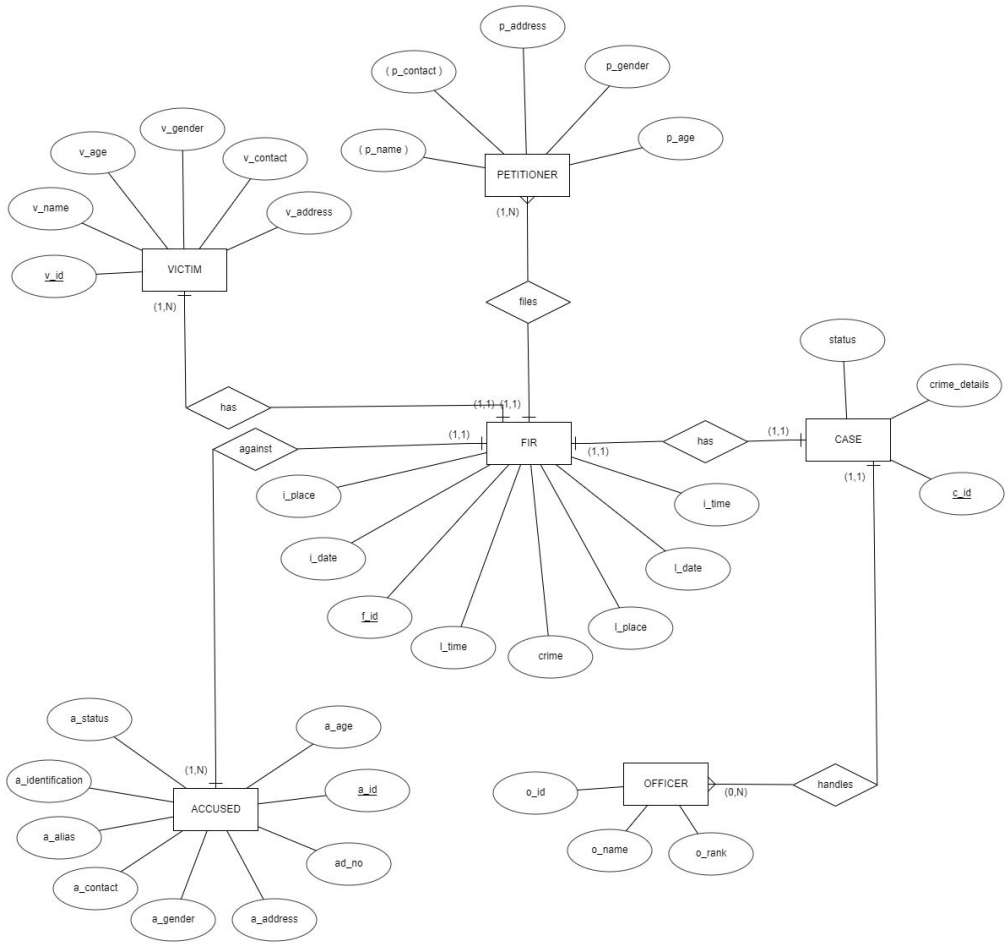
TRANSACTIONS

A. The user/petitioner is asked to enter details required to file an FIR. The petitioner details are entered into the petitioner table .
B. A new row is created for the accused , where only the gender column is filled .
C. A new row is created for the victim , where only the gender is filled.
D. A new row is inserted into the FIR table with FIR details and the accused and victim ID.
E. A new row is inserted into the CASE table , for the new FIR filed.
F. When the user chooses to update details , the fields are updated with new values only if they don't already exist in the table.
   If they already exist , then the existing ID is retrieved and updated as the new ID in FIR table.
   The new row which had been created while filing the FIR is deleted.

DATA TYPES

1. INTEGER : It is used to store whole numbers, that is, numbers without fractional components, of various ranges. Attempts to store values outside of the allowed range will result in an error.
2. VARCHAR : It is used to store alphanumeric characters.It can have a varying length of upto 8000 characters.
3. DATE : The DATE data type accepts date values. When storing a date value, PostgreSQL uses the yyyy-mm-dd format. A particular date value , like DAY , MONTH , YEAR can also be extracted from this.
4. TIME : A time value may have a precision up to 6 digits. The precision specifies the number of fractional digits placed in the second field.The TIME data type requires 8 bytes and its allowed range is from 00:00:00 to 24:00:00.Current time can be retrieved with current_time function.
5. SERIAL : It is a pseudo-type used to define auto-increment columns in tables. It creates sequences to be followed while inserting rows.One does not have to insert the column while inserting rows into the data.
6. NUMERIC : This type can store numbers with many digits. Typically,we use the NUMERIC type for the monetary or other amounts which precision is required.

ER DIAGRAM



FUNCTIONAL DEPENDENCY AND NORMALIZATION

FUNCTIONAL DEPENDENCY

A functional dependency is given by X -> Y , where X and Y are attributes of a table.

A. case.c_id → {o_id,c_status,c_description}
B. fir.f_id → { l_date ,l_time ,l_place,i_date,i_time,i_place}
C. accused.a_id → {a_name,ad_no,a_gender,a_age,a_contact,a_ddress,a_status,a_alias , a_identification}
D. accused.ad_no → {a_name,a_id,a_gender,a_age,a_contact,a_ddress,a_status,a_alias ,a_identification}

E. petitioner{Name , Contact} -> {Address , Age,Gender }
F. officer.o_id → {o_name , o_rank}
G. victim.v_id → {v_name , v_age , v_gender ,v_contact , v_address}

KEYS BASED ON FD

1. case.c_id : Primary Key
2. fir.f_id : Primary Key
3. accused.a_id : Primary Key
4. accused.ad_no : Candidate Key , as it is unique for each accused.
5. petitioner.(Name , Contact) : Primary Key
   Many petitioners can have the same name . Thus it can't be a primary key alone.
6. officer.o_id : Primary Key
7. victim.v_id : Primary Key

NORMALIZATION

1. 1st Normal Form :

A relation is in first normal form if every attribute in that relation is a single valued attribute.
→ All the tables are in 1NF.

2. 2nd Normal Form :

Table is in 1NF (First normal form)
No non-prime attribute is dependent on the proper subset of any candidate key of table. (No partial functional Dependency)
→ All the tables are in 2NF.

3. 3rd Normal Form :

Table must be in 2NF
There must be no Transitive dependencies .
In the **case** table , f_id is unique but c_id is the primary key . There is a transitive dependency.

c_id	**f_id**	c_status	c_description

→ Case table is not in 3NF because : c_id → f_id
f_id → status , c_description
c_id → status , c_description
Case table is decomposed into case_fir and case_details to resolve this.
case_fir : {c_id , f_id} and case_details : {c_id , c_status , c_description}

Testing for lossless join → When a natural join is performed on both the tables , it yields the original table without any loss of data .It has a lossless join .

DDL

1. OFFICER

   CREATE SEQUENCE public.officer_id INCREMENT 1 START 100 MINVALUE 100 MAXVALUE 300 CACHE 1;
   CREATE TABLE public.officer
   (o_id integer NOT NULL DEFAULT nextval('officer_id'::regclass),o_name character varying(50) , o_rank character varying(10) , CONSTRAINT officer_pkey PRIMARY KEY (o_id))

2. ACCUSED

   CREATE SEQUENCE public.accused_id INCREMENT 1 START 10000 MINVALUE 10000 MAXVALUE 30000 CACHE 1;
   CREATE TABLE public.accused
   (a_id integer NOT NULL DEFAULT nextval('accused_id'::regclass),ad_no character varying(10) ,a_name character varying(20) ,a_gender character(1) ,a_status varchar(20),a_alias character varying(10),a_identification character varying(100) ,a_age numeric(2,0),a_contact numeric(10,0),a_address character varying(100) ,CONSTRAINT accused_pkey PRIMARY KEY (a_id),CONSTRAINT accused_ad_no_check CHECK (ad_no::text ~ '^[0-9]*$'::text),CONSTRAINT accused_a_gender_check CHECK (a_gender = ANY (ARRAY['M'::bpchar, 'F'::bpchar, 'm'::bpchar, 'f'::bpchar])),CONSTRAINT accused_a_status_check CHECK (a_status::text = ANY (ARRAY['Bailed'::character varying, 'Released'::character varying, 'Arrested'::character varying, 'Escaped'::character varying]::text[])))

3. VICTIM

   CREATE SEQUENCE public.victim_idINCREMENT 1 START 1000MINVALUE 1000 MAXVALUE 3000CACHE 1;
   CREATE TABLE public.victim
   (v_id integer NOT NULL DEFAULT nextval('victim_id'::regclass), v_name character varying(40),v_age numeric(2,0), v_contact numeric(10,0),v_gender character varying(1) ,v_address character varying(100) ,CONSTRAINT victim_pkey PRIMARY KEY (v_id),CONSTRAINT victim_v_gender_check CHECK (v_gender::text = ANY (ARRAY['M'::character varying, 'F'::character varying, 'm'::character varying, 'f'::character varying]::text[])))

4. PETITIONER

   CREATE TABLE public.petitioner(p_name character varying(50) ,p_contact numeric(10,0),p_address character varying(100),p_gender character(1) ,p_age integer,CONSTRAINT petitioner_pkey PRIMARY KEY (p_name ,p_contact),CONSTRAINT petitioner_p_gender_check CHECK (p_gender = ANY (ARRAY['M'::bpchar, 'F'::bpchar, 'm'::bpchar, 'f'::bpchar])))

5. FIR

```
CREATE TABLE fir
(f_id SERIAL PRIMARY KEY,v_id int , l_place varchar(100),l_date date DEFAULT current_date, l_time time default current_time , i_place varchar(100),i_date date, i_time time,p_name varchar(50),p_contact numeric(10),a_id int , crime varchar(1000),constraint fir_fk1 FOREIGN KEY(a_id) references accused(a_id),constraint fir_fk2 FOREIGN KEY(p_name,p_contact) references petitioner(p_name,p_contact),constraint fir_fk3 FOREIGN KEY(v_id) references victim(v_id));
```

6. CASE_FIR

```
CREATE SEQUENCE public.case_idINCREMENT 1START 10000MINVALUE 10000 MAXVALUE 30000 CACHE 1;
CREATE TABLE case_fir(c_id integer DEFAULT nextval('case_id'::regclass) PRIMARY KEY , f_id integer, constraint case_fir_fk FOREIGN KEY(f_id) references fir(f_id));
```

7. CASE_DETAILS

```
create table case_details(c_id int PRIMARY KEY,o_id int ,c_status varchar(7) CHECK(c_status in ('OPEN','CLOSED')),c_description varchar(1000), CONSTRAINT cd_fk1 FOREIGN KEY(c_id) references case_fir(c_id),constraint cd_fk2 FOREIGN KEY(o_id) references officer(o_id));
```

TRIGGERS

PostgreSQL Triggers are database callback functions, which are automatically performed/invoked when a specified database event occurs.

1. Trigger to avoid duplicate Accused details

```
CREATE FUNCTION accused_func()
RETURNS TRIGGER
AS $$
DECLARE cnt integer;
BEGIN
IF EXISTS (select table_name from information_schema.tables where table_name='acc_table')
THEN DROP TABLE acc_table;END IF;
CREATE TEMP TABLE acc_table AS SELECT a_id,ad_no from  accused WHERE NEW.ad_no = accused.ad_no;SELECT count(*) INTO cnt from acc_table;
if cnt>=1
THEN RAISE EXCEPTION 'Accused details already exists';RETURN NULL;
END IF;
IF cnt=0 THEN RETURN NEW;END IF; RETURN NULL;
END;$$ LANGUAGE plpgsql;
CREATE TRIGGER accused_trigger BEFORE INSERT OR UPDATE
ON accused FOR EACH ROW EXECUTE PROCEDURE accused_func();
```

2. Trigger to avoid duplicate rows in Victim table

```
CREATE FUNCTION victim_func()
RETURNS TRIGGER
AS $$
DECLARE cnt_vic integer;
BEGIN IF EXISTS (select table_name from information_schema.tables where table_name='vic_table')
THEN DROP TABLE vic_table; END IF;CREATE TEMP TABLE vic_table AS SELECT v_id from  victim
WHERE (NEW.v_name = victim.v_name AND NEW.v_contact=victim.v_contact AND
NEW.v_age=victim.v_age AND NEW.v_address=victim.v_address) ; SELECT count(*) INTO cnt_vic
from vic_table;if cnt_vic>=1 THEN RAISE EXCEPTION 'Victim details already exists';RETURN NULL;
END IF;IF cnt_vic=0 THEN RETURN NEW; END IF;RETURN NULL;
END;$$ LANGUAGE plpgsql;
CREATE TRIGGER victim_trigger BEFORE INSERT OR UPDATE
ON victim FOR EACH ROW EXECUTE PROCEDURE victim_func();
```

3. Trigger to avoid duplicate FIRs

```
CREATE FUNCTION FIR_func()
RETURNS TRIGGER
AS $$
DECLARE cnt_FIR integer;
BEGIN IF EXISTS (select table_name from information_schema.tables where table_name='fir_table')
THEN DROP TABLE fir_table;END IF;CREATE TEMP TABLE FIR_table AS SELECT v_id from  fir
WHERE (NEW.crime = fir.crime AND NEW.i_place=fir.i_place AND NEW.i_date=fir.i_date AND
NEW.i_time=fir.i_time) ; SELECT count(*) INTO cnt_FIR from FIR_table; if cnt_FIR>=1 THEN  RAISE
EXCEPTION 'FIR already filed on this crime'; RETURN NULL;END IF; IF cnt_FIR=0 THEN RETURN
NEW; END IF; RETURN NULL;
END; $$ LANGUAGE plpgsql;
CREATE TRIGGER FIR_trigger BEFORE INSERT
ON fir FOR EACH ROW EXECUTE PROCEDURE FIR_func();
```

SQL QUERIES

1. Display the names of those victims , the status of the case , and the officer who filed the case , where the incident took place in Mawat

```
SELECT v_name,o_name,c_status from
((((fir join victim on fir.v_id = victim.v_id)  JOIN case_fir on fir.f_id = case_fir.f_id
)JOIN case_details on case_fir.c_id = case_details.c_id) JOIN officer on officer.o_id =
case_details.o_id)where i_place = 'Mawat'
```

2. Display the victim names,address and the petitioner name of the crimes which took place in May of any year where the petitioner is a Female.

```
SELECT v_name, p_name From (SELECT * FROM victim NATURAL JOIN fir WHERE
EXTRACT(MONTH FROM i_date)=5) AS FOO NATURAL JOIN petitioner Where
p_gender='F';
```

3. Displaying the minimum age of victims based on their genders who were involved in a crime before the 15 th of any month.

```
SELECT v_gender, Min(ALL v_age) FROM(SELECT v_name, c_description, a_name,
v_age, v_gender FROM (SELECT * FROM case_details NATURAL JOIN FIR NATURAL
JOIN case_fir WHERE EXTRACT(DAY from i_date)<=15)
AS func NATURAL JOIN accused NATURAL JOIN victim) AS foo
GROUP BY(v_gender);
```

4. Display the criminal details and associated crimes , with the victim name for a criminal named Roosevelt Morgan.

```
select a_name,a_contact,ad_no,c_description,v_name,crime from
((((fir JOIN accused AC on AC.a_id =fir.a_id )JOIN case_fir CF on CF.f_id = fir.f_id)JOIN
case_details CD on CD.c_id = CF.c_id  )JOIN victim on fir.v_id = victim.v_id) where
AC.a_name = 'Roosevelt Morgan'
```

5. Display the count of status of criminals where the case is open.

```
SELECT a_status, Count(a_status)FROM(SELECT DISTINCT(a_id), a_status, a_name FROM
(SELECT * FROM (case_details NATURAL JOIN case_fir NATURAL JOIN fir) WHERE
c_status='OPEN') AS func NATURAL JOIN accused) AS foo Group BY(a_status);
```

CONCLUSION

This management system is capable of managing records of crimes.
It allows users to file an fir without actually having any hassle of human interference.
This database also provides a cumulative record of knowing the criminals and their statuses understanding the crimes in their place .