Database Design and Development Database Project

By

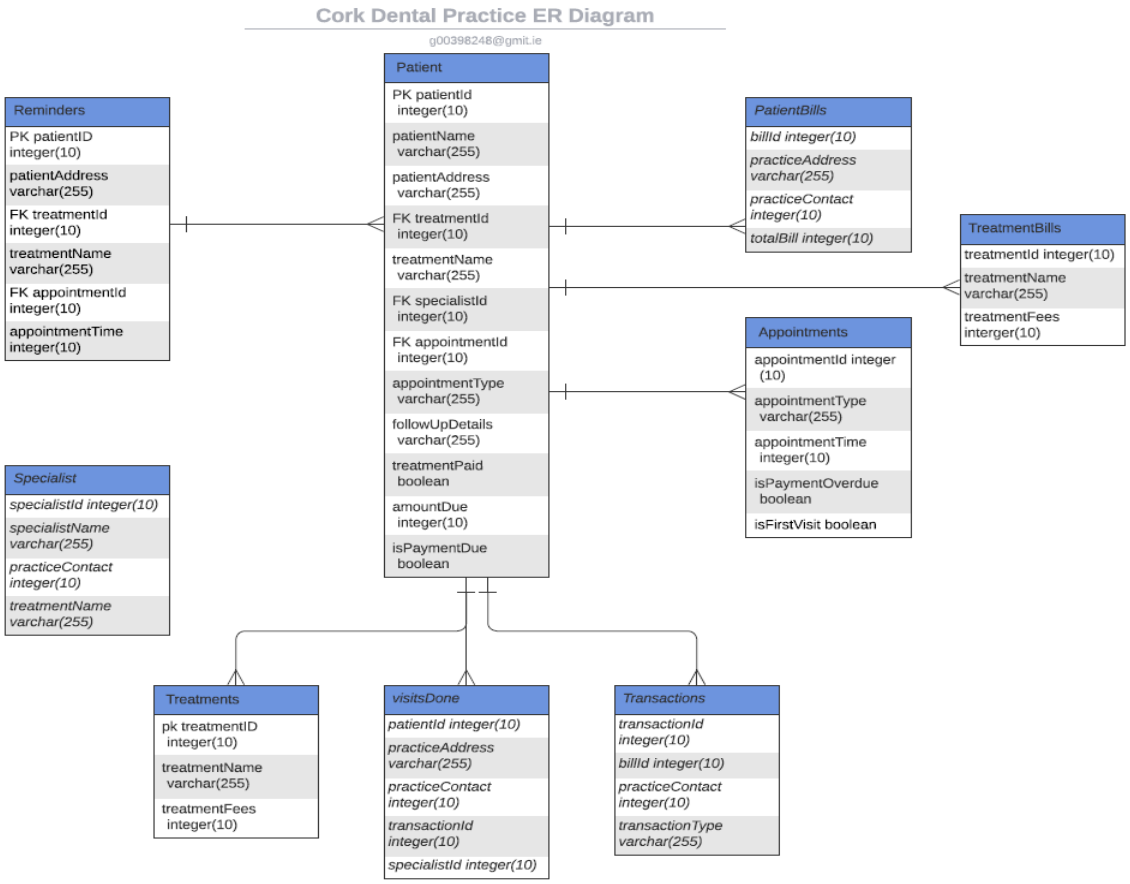
G00398248
Database Design and Development
Eoin Foley

INTRODUCTION

This project report summarizes the design and construction of a database intended for use within a commercially operated dental practice. The project focuses on the implementation of three specific areas.

1. Creation of an Entity Relationship Diagram (ERD)
2. Design of a relational database and queries
3. Codd rules development and implementation via the database schema.
4. Queries together with a broad narrative to describe the daily operations of the dental practice

ER DIAGRAM & RELATIONAL SCHEMA



In this ER Diagram Patient is the main entity as it has connections to all the foreign key elements. The relationship is one to many as each patient can avail of many treatments. There will be only one appointment record, reminder record and bill record for one patient, zero or one specialist record.

CODD RULES

Rule 1: The Information Rule: Data can be retrieved by matching identifiers

Eg: a simple SELECT statement explaining how a unique value was returned using a primary key

- SELECT patientID FROM `patient`; -- patientID is the Primary Key, hence it retrieves all unique values

Rule 2: **The Guaranteed Access Rule**: Data from a table can be accessed to field level

Eg: a simple SELECT statement demonstrating that unique values can be accessed in a table via a combination of table name, primary key and column name.

- SELECT PatientID, PatientName, PatientAddress FROM `patient`; -- patientID is the Primary Key, hence it retrieves all unique values and corresponding information on the other selected columns

Rule 3: **Systematic Treatment of Null Values**

Eg: A example of table that has a NULL value associated with a number of columns

- SELECT * FROM `patient` WHERE SpecialistID is NULL; -- Check how created schema treats NULLs

In this table, if there's no specialist associated with a patient then the followup notes are blank, hence both columns have NULLs

Rule 4: **Dynamic Online Catalog based on the relational model**: The meta data (data about data)

Eg: the tables you create are stored in special tables called the INFORMATION_SCHEMA an example of the entries for your tables will suffice

- SELECT * FROM information_schema.tables WHERE TABLE_SCHEMA = 'dentalpractice'; -- Search for table names in the information_schema master table where the schema name matches our created schema (dentalpractice)

Rule 5: **The Comprehensive Data Sub Language Rule**:

Eg: Add DDL and DML statements

DDL (Data Definition Language) --> CREATE, ALTER, DROP

- CREATE:

```
CREATE TABLE `appointments` (
  `AppointmentID` INT(10) NOT NULL,
  `AppointmentType` VARCHAR(25) NOT NULL,
  `AppointmentTime` INT(10) NOT NULL,
  `isPaymentOverdue` tinyint(1) NOT NULL,
  `isFirstVisit` tinyint(1) NOT NULL
)
```

- ALTER:

```
ALTER TABLE `practices`
  ADD PRIMARY KEY (`PraticeName`);
ALTER TABLE `appointments` CHANGE `AppointmentTime`
`AppointmentDate` DATE NOT NULL;
```

- DROP:

```
DROP table appointments;
```

DML (Data Manipulation Language) --> INSERT, UPDATE, DELETE

- INSERT:

```
INSERT INTO `appointments`(`AppointmentID`, `AppointmentType`,
`AppointmentTime`, `isPaymentOverdue`, `isFirstVisit`) VALUES
('6745','phone','2022-03-08','0','1')
```

- UPDATE:

```
UPDATE `appointments` SET `AppointmentDate` = '2022-04-04' WHERE
`appointments`.`AppointmentID` = 6745
```

- DELETE:

```
DELETE FROM appointments WHERE `appointments`.`AppointmentID` =
6745
```

Rule 6: The View Updating Rule:

- CREATE VIEW specialistView as (

```
SELECT * FROM dentalpractice.patient WHERE SpecialistID is not NULL
```

) -- This view contains all the entries which has a specialist appointment associated

- UPDATE `specialistview` SET `AmountDue`='1' -- updating view to set AmountDue as 1 as these patients are referred to a specialist for a followup and their payment is due
- SELECT * FROM dentalpractice.patient -- this is to demonstrate that updating the view actually updates all the values in the base table

Rule 7: **High Level Insert Update and Delete Rule:**

UPDATE patientbills

SET BillExceeded =

CASE

WHEN AmountDue > 200 THEN 1

ELSE 0

END

- BillExceeded becomes a boolean '1' iff amountDue goes above 200.
- This UPDATE query will update all the eligible rows asper the condition

Rule 8: **Physical Data Independence**: SQL Not Required

Rule 9: **Logical Data Independence:**

Rule 10: Integrity Independence: This is basically the need for the relationship between Primary and Foreign Keys. A query that demonstrates this working in your database.

- ALTER TABLE `patient` ADD FOREIGN KEY (`BillID`) REFERENCES `patientbills`(`BillID`) ON DELETE RESTRICT ON UPDATE RESTRICT;
- **SQL**:

UPDATE patient a

JOIN patientbills b ON a.BillID = b.BillID

SET a.AmountDue = b.AmountDue

) -- This view contains all the entries which has a specialist appointment associated

- UPDATE `specialistview` SET `AmountDue`='1' -- updating view to set AmountDue as 1 as these patients are referred to a specialist for a followup and their payment is due

SELECT * FROM dentalpractice.patient -- this is to demonstrate that updating the view actually updates all the values in the base table

NARRATIVE TO ACCOMPANY SQL QUERIES

1. Dr Mary Mulcahy runs a dental practice in a small town in Cork. She can treat most cases, although a few specialist cases are referred to larger practices in Cork city where the required expertise and equipment are available.

- CREATE Statements

2. She arranges a suitable appointment by referring to the appointments diary unless they owe over a certain amount, or for too long, as seen from the patient's chart.

- Boolean Update
    - BillExceeded? (if > 200)
    - DaysAmountDue? (if > 30)
    - If both are 0, then give appointment else don't
- **SQL**:

```
UPDATE appointments a
JOIN patientbills b ON a.PatientID = b.PatientID
SET a.isPaymentOverdue =
CASE
  WHEN b.BillExceeded = 1 OR b.DaysAmountDue > 30 THEN 1
  ELSE 0
END
```

3. She writes the new appointment into the diary and, if it is the patient's first visit, she makes a new chart for the patient and puts it into the charts filing cabinet.

- SQL to check if firstVisit --> create a new chart (insert in the patients) else update existing chart (update existing patient in the patients)
- Join foreign key appointsID patients to Primary key appID in appointments table

- **SQL**:

SELECT * FROM `appointments` WHERE isFirstVisit = 1 -- first visit

SELECT * FROM `appointments` WHERE isFirstVisit = 0 -- not a first visit

UPDATE table patients where isFirstVisit = 0

UPDATE `patient` SET `TreatmentID`='[value-4]',`TreatmentName`='[value-5]',`SpecialistID`='[value-6]',`BillID`='[value-7]',`AppointmentID`='[value-8]',`AppointmentType`='[value-9]',`FollowUpDetails`='[value-10]',`TreatmentPaid`='[value-11]',`AmountDue`='[value-12]',`isPaymentOverdue`='[value-13]' WHERE 'patientID' = '8287'

and INSERT where isFirstVisit = 1

INSERT INTO `patient`(`PatientID`, `PatientName`, `PatientAddress`, `TreatmentID`, `TreatmentName`, `SpecialistID`, `BillID`, `AppointmentID`, `AppointmentType`, `FollowUpDetails`, `TreatmentPaid`, `AmountDue`, `isPaymentOverdue`) VALUES ('[value-1]','[value-2]','[value-3]','[value-4]','[value-5]','[value-6]','[value-7]','[value-8]','[value-9]','[value-10]','[value-11]','[value-12]','[value-13]')

4. Sometimes patients contact Helen to rearrange or even cancel appointments. (Checked - RULE 5)

- Rearrange: Update statement in the appointments table

Cancellations are done by simply tippexing out the appointment in the diary. The details in the patient's chart are also updated with rearrangements and cancellations (late cancellations are charged a €10 late cancellation fee).

- Cancel: Delete statement in the appointments table and update AmountDue in patients table as amount + 10

- **SQL**:

DELETE FROM appointments WHERE AppointmentID = 6748;

UPDATE patient SET AmountDue=AmountDue + 10 WHERE AppointmentID = 6748;

5. Every Tuesday morning, Helen checks the appointment diary and makes a list of all next week's appointments RULE 7

- Pull out next week appointments from the appointments table and add it(INSERT) to the reminders table
- **SQL**:

Helen gets ID's who have appointments in the next week - Dates between this Saturday and next Saturday

```
SELECT * FROM `appointments` WHERE AppointmentDate BETWEEN
STR_TO_DATE(CONCAT(YEAR(NOW()),WEEK(NOW(),7), ' Saturday'),
'%X%V %W') AND
STR_TO_DATE(CONCAT(YEAR(NOW()),WEEK(NOW(),7), ' Saturday'),
'%X%V %W') + 8
```

Helens notes down the IDs from the above query and puts them in below query.

```
UPDATE reminders a
JOIN appointments b ON a.AppointmentID = b.AppointmentID
SET a.AppointmentNextWeek =
CASE
  WHEN a.AppointmentID IN ('6746', '6747') THEN 1
  ELSE 0
END
```

She puts a reminder to those where AppointmentNextWeek = 1

```
SELECT * FROM reminders where AppointmentNextWeek  = 1
```

6. Next, at about 2.30 p.m. she prepares bills by searching the patient charts to find details of any unpaid treatments. Then she looks up the Treatment Fees guidelines book, which Dr Mulcahy updates from time to time

- Search from treatments table, fetch the fees and add it to the Amount Due

```
UPDATE `patient` a

JOIN treatments b ON a.TreatmentID = b.TreatmentID

SET AmountDue= a.AmountDue + b.TreatmentFees
```

7. Patients pay by cheque, credit card or cash, either by post or by dropping in. The bill or the bill number is enclosed with the payment. Treatments which have been paid for are marked as such in the patient's file so that they will not be billed again

- SELECT * FROM patient WHERE AmountDue > 0 -- These patients have their bills paid

8. Helen takes the card and arranges appointments with the patient for any required follow-up treatments written on the card, and enters them into the appointments diary.

- Update patient table with follow-up details
- **SQL**:

```
UPDATE `patient` SET `FollowUpDetails`='Consult a Specialist for …. '
WHERE PatientID = '8286'
```

9. If the patient needs specialist treatment which Dr Mulcahy cannot provide, she writes the name of an appropriate specialist on the filled visit card and the secretary sends a patient referral to the specialist.

- Update patient table with specialist details and join the specialist table

```
UPDATE `patient` SET `SpecialistID`='6294' WHERE PatientID = '8286'
```