Codd's Rules

Brian Cheasty

A00267976

Rule 1:

All information in a relational database is represented explicitly at the logical level in exactly one way – by values in a table.

The below images are taken from table to demonstrate that we have met this criterion. All tables are in 3NF.

Patient Table:

+ Options

	Patient_ID	Patient_Name	Patient_Address	Tel_No	Date_of_Birth	Patient_Set_Up_Date	Gender	Marketing_Preference
Edit Copy Delete	1	Con Kirwan	5 Waterford	0851234568	1979-07-05	1980-01-01	Male	No
Edit Copy Delete	2	Brian Cheasty	4 Waterford	0851234567	1980-05-28	1990-01-01	Male	Yes
Edit Copy Delete	3	Sean Nolan	6 Waterford	0851234569	1982-12-12	2000-01-01	Male	Yes
Edit Copy Delete	4	Sean Matthews	7 Waterford	0851234570	1985-07-14	2005-01-01	Male	No
Edit Copy Delete	5	Rhona Caffrey	8 Waterford	0851234571	1985-07-15	2005-01-02	Female	Yes
Edit Copy Delete	6	Kevin Cheasty	9 Waterford	0851234572	1985-07-15	2005-01-02	Male	Yes

Appointments Table:

+ Options

	Appointment_ID	Patient_ID	Appointment_Status	Schedule_ID
Edit Copy Delete	1001	1	Booking	504
Edit Copy Delete	1002	2	NULL	500
Edit Copy Delete	1004	3	Booking	518
Edit Copy Delete	1005	4	NULL	500
Edit Copy Delete	1006	5	NULL	500
Edit Copy Delete	1007	6	NULL	500

Treatments Table:

+ Options

	Treatment_ID	Treatment_Name	Treatment_Description	Price
Edit Copy Delete	100	Clean	Polish of teeth	70.00
Edit Copy Delete	101	Bridges and Impants	Bridges and implants are two ways to replace a mis...	500.00
Edit Copy Delete	102	Extraction	A severely damaged tooth may need to be extracted	201.00
Edit Copy Delete	103	Bonding	Bonding is a treatment that can be used to repair ...	100.00
Edit Copy Delete	104	Braces	Not Available	NULL
Edit Copy Delete	105	Crowns and Caps	A crown or cap is a cover that fits over a tooth t...	300.00

Treatments Booked Table:

+ Options

Treatment_ID	Appointment_ID
108	NULL
112	1002
107	1004
112	1005
112	1006
112	1007
106	1009
100	1011
102	1013
112	1014
107	1016
100	1018

Treatments Provided Table:

+ Options

Appointment_ID	Treatment_ID	Treatment_Status	Treatment_Notes	Treatment_Date	Treatment_Price
1001	NULL	NULL	NULL	NULL	NULL
1002	NULL	NULL	NULL	NULL	NULL
1004	109	Complete	No Issue	2019-04-24	600.00
1005	NULL	NULL	NULL	NULL	NULL
1006	NULL	NULL	NULL	NULL	NULL
1007	NULL	NULL	NULL	NULL	NULL
1009	102	Complete	No Issue	2019-04-24	201.00
1011	NULL	NULL	NULL	NULL	NULL

Bill Table:

+ Options

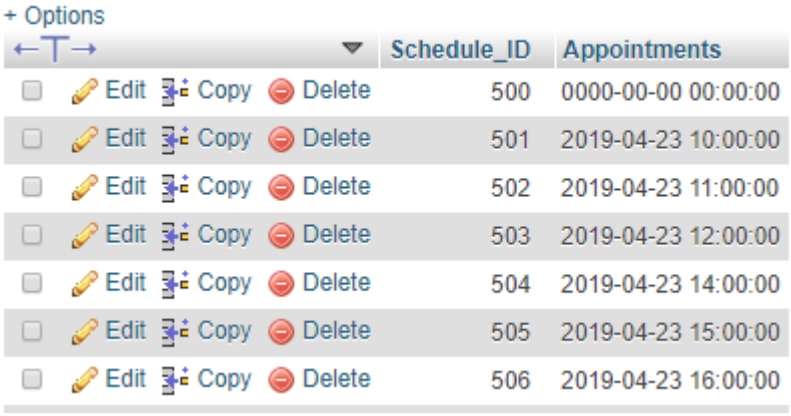
←T→	Bill_ID	Treatment_Cost	Bill_Date	Appointment_ID
Edit Copy Delete	1501	NULL	NULL	1001
Edit Copy Delete	1502	NULL	NULL	1002
Edit Copy Delete	1504	600.00	2019-04-24	1004
Edit Copy Delete	1505	NULL	NULL	1005
Edit Copy Delete	1506	NULL	NULL	1006

Payments Table:

+ Options

←T→	Bill_ID	Payment_Amount	Payment_Date	Payment_ID	Payment_Type
Edit Copy Delete	NULL	0.00	2019-04-24	2001	NULL
Edit Copy Delete	1502	0.00	2019-04-24	2002	NULL
Edit Copy Delete	1504	0.00	2019-04-24	2004	NULL
Edit Copy Delete	1504	600.00	2019-04-24	2005	Credit Card
Edit Copy Delete	1505	0.00	2019-04-24	2006	NULL
Edit Copy Delete	1506	0.00	2019-04-24	2007	NULL

Doctor Schedule Table:



Code for Rule 1:

```
select * from patient
select * from appointment
select * from treatments
select * from treatment_booked
select * from treatment_provided
select * from bill
select * from payments
select * from doctor_schedule
```

Rule 2:

Each and every value is accessible by a combination of - Table name, primary key value and column name:

Taking the Doctor Schedule table above we will access the appointment details on the 23/04/2019 at 2pm.


```
select appointments from doctor_schedule where Schedule_ID= 504
```

Rule 3:

Null values are supported:

In particular we can demonstrate this with the setting up of a new patient. To set up a new patient and provide the patient with a patient chart we must create a placeholder in all tables to ensure that the patient chart updates. To do so we apply null values.

+ Options

	Appointment_ID	Patient_ID	Appointment_Status	Schedule_ID
Edit Copy Delete	1002	2	NULL	500
Edit Copy Delete	1005	4	NULL	500
Edit Copy Delete	1006	5	NULL	500
Edit Copy Delete	1007	6	NULL	500
Edit Copy Delete	1014	10	NULL	500
Edit Copy Delete	1021	14	NULL	500
Edit Copy Delete	1022	15	NULL	500
Edit Copy Delete	1025	17	NULL	500
Edit Copy Delete	1026	18	NULL	500
Edit Copy Delete	1029	20	NULL	500

The null values in the appointment table can be viewed using the below code.

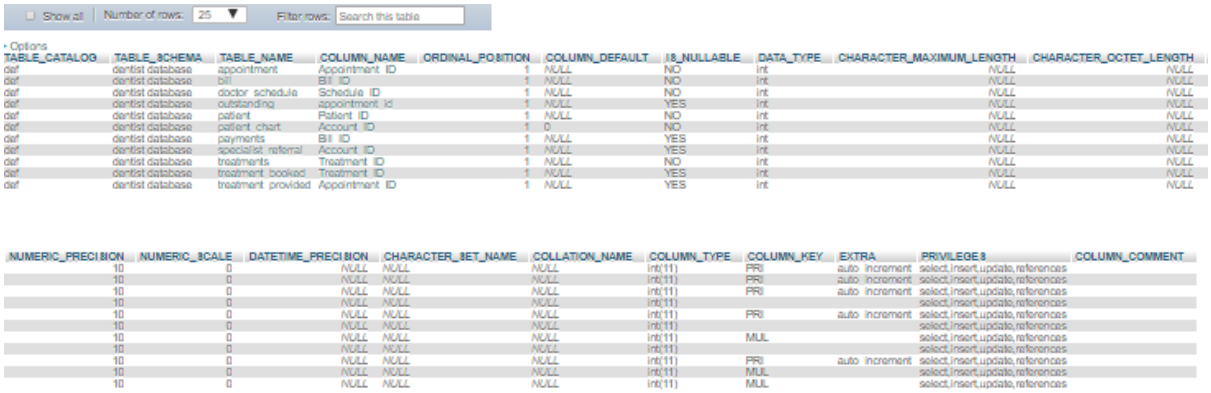
```
select * from Appointment where Appointment_Status is null
```

Rule 4:

Dynamic online catalog based on the relational model.

Automatically creating tables of defined tables. Metadata about our tables.

Below is a screen shot of the metadata from the schema.



TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	CHARACTER_OCTET_LENGTH
def	dentist database	appointment	Appointment ID	1	NULL	NO	int	NULL	NULL
def	dentist database	bill	Bill ID	1	NULL	NO	int	NULL	NULL
def	dentist database	doctor schedule	Schedule ID	1	NULL	NO	int	NULL	NULL
def	dentist database	outstanding	appointment id	1	NULL	YES	int	NULL	NULL
def	dentist database	patient	Patient ID	1	NULL	NO	int	NULL	NULL
def	dentist database	patient chart	Account ID	1	0	NO	int	NULL	NULL
def	dentist database	payments	Bill ID	1	NULL	YES	int	NULL	NULL
def	dentist database	specialist referral	Account ID	1	NULL	YES	int	NULL	NULL
def	dentist database	treatments	Treatment ID	1	NULL	NO	int	NULL	NULL
def	dentist database	treatment booked	Treatment ID	1	NULL	YES	int	NULL	NULL
def	dentist database	treatment provided	Appointment ID	1	NULL	YES	int	NULL	NULL

NUMERIC_PRECISION	NUMERIC_SCALE	DATETIME_PRECISION	CHARACTER_SET_NAME	COLLATION_NAME	COLUMN_TYPE	COLUMN_KEY	EXTRA	PRIVILEGES	COLUMN_COMMENT
10	0	NULL	NULL	NULL	int(11)	PRI	auto_increment	select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)	PRI	auto_increment	select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)	PRI	auto_increment	select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)			select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)	PRI	auto_increment	select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)			select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)	MUL		select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)			select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)	PRI	auto_increment	select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)	MUL		select,insert,update,references	
10	0	NULL	NULL	NULL	int(11)	MUL		select,insert,update,references	

```
select * from information_schema.`columns` where table_schema = 'dentist database' group by table_name
```

Rule 5:

The comprehensive data sub language.

To show data definition, view definition, data manipulation and integrity constraint I have code below from the schema and queries which demonstrates this:

Data Definition where we create the table.

View Definition where we create the view.

Integrity constraint where we set constraints such as size of value in payment amount and primary key:

```
/*This table record the many payments that can take place for any given bill*/
Create table if not exists Payments (
Bill_ID int,
Payment_Amount decimal(10,2),
Payment_Date date,
Payment_ID int Auto_increment,
Payment_Type varchar(12),
Foreign key (Bill_ID) references Bill(Bill_ID),
Primary key (Payment_ID));
/*To ensure that each primary key starts from a different point I will specify the starting point.
Greater distance between starting points would be required in a commercial environment.*/
```

```
Alter table Payments auto_increment = 2000;

/*The secretary can view all treatments marked are specialist to process the referral
Notes are included for the specialist; these are updated when the specialist returns the
referral*/
Create view Specialist_referral as
select
appointment.patient_id as 'Account_ID',
patient.Patient_name as 'Patient_Name',
treatment_provided.Treatment_Notes as 'Notes',
treatment_provided.Treatment_Status as 'Consultation_Status',
Treatment_provided.Treatment_Date as 'Referral_Date'
from appointment,treatment_provided,patient
where
treatment_provided.appointment_id=appointment.Appointment_ID
and patient.patient_id = appointment.patient_id
and treatment_provided.treatment_status = "Specialist";

Data Manipulation where we insert data.
insert into patient values (null,"Con Kirwan","5 Waterford","0851234568",'1979-07-
05','1980-01-01',"Male","No");
insert into appointment values (null,(select patient_id from patient where patient_name =
"Con Kirwan"
),null,500);
insert into treatment_booked values((select treatment_id from treatments where
Treatment_name = "No Booking Yet"),
(select appointment_id from appointment where patient_id=(select patient_id from patient
where patient_name = "Con Kirwan"
) and schedule_id = 500));
insert into treatment_provided values((select appointment_id from appointment where
patient_id=(select patient_id from patient where patient_name = "Con Kirwan")
 and schedule_id = 500),null,null,null,null,null);
insert into Bill values (null,null,null,(select appointment_id from appointment where
patient_id=(select patient_id from patient where patient_name = "Con Kirwan")
and schedule_id = 500));
insert into payments values((select bill_id from bill where bill.appointment_id=(select
appointment_id from appointment
where patient_id=(select patient_id from patient where patient_name = "Con Kirwan")
and schedule_id = 500)),0,curdate(),null,null);

Transaction boundaries:
Each transaction is defined by the ; at the end.
If we take the first transaction:
insert into patient values (null,"Con Kirwan","5 Waterford","0851234568",'1979-07-
05','1980-01-01',"Male","No");
```

The transaction begins at insert. Should an individual component violate any of the conditions for the insert then it will fail and rollback to the beginning.

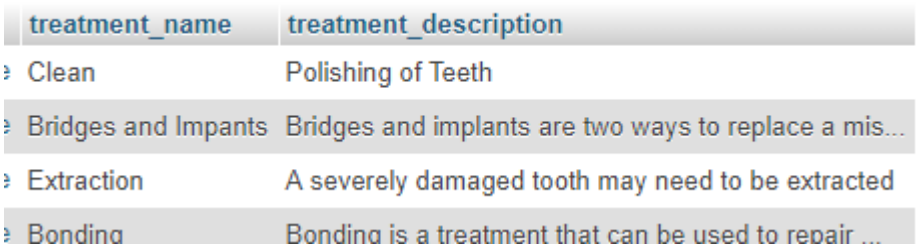
Rule 6:

An update on a view will update the base table:

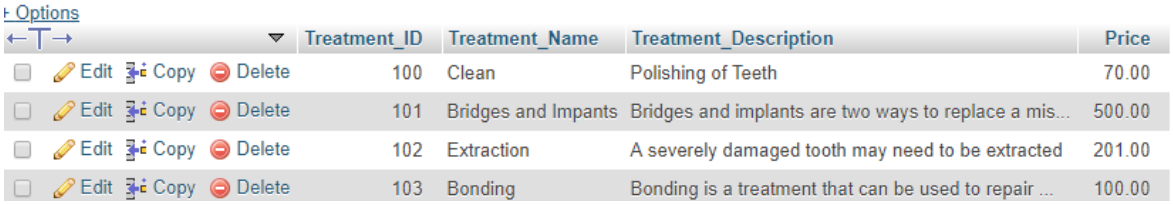
To demonstrate this, I will walk through the process.

Create a view of treatments Table:

Update of Treatment View Table:



Updated on Treatments Table:




```
create view Treatment_View as select treatment_name,treatment_description from treatments
select * from treatment_view
update treatment_view set treatment_description = "Polishing of Teeth" where treatment_name = "Clean"
Select * from treatment_view
select * from treatments
```

Rule 7:

The capability to handle a base relation or a derived relation as a single operand.

```
insert into treatment_booked values
((select treatment_id from treatments where Treatment_name = "Gum
Surgery"),
(select appointment_id from appointment where patient_id=2 and schedule_id
= 504));
```

To demonstrate this rule the above command is inserting, into the table called treatment_booked, two values. The first value is the treatment_id from another table where the name is Gum Surgery. The second value is an appointment id from another table where the appointment id = a value and the schedule id = a value.

Here we are selecting from multiple rows to update one row.

Rule 8:

Physical Data Independence:

The data is available to be moved as the data is exported in SQL to a SQL document and can then be uploaded to the new storage.

This ensures that the data is independent of the storage.

Below is a snippet of the database after exporting:

```
-- phpMyAdmin SQL Dump
-- version 4.8.4
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1
-- Generation Time: Apr 25, 2019 at 12:48 AM
-- Server version: 10.1.37-MariaDB
-- PHP Version: 7.3.0

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `dentist database`
--

-- --------------------------------------------------------

--
-- Table structure for table `appointment`
```

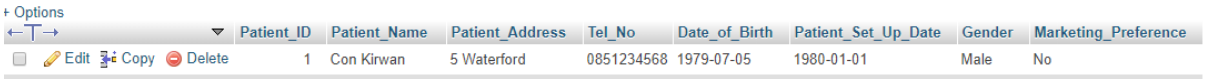
```
--

CREATE TABLE `appointment` (
  `Appointment_ID` int(11) NOT NULL,
  `Patient_ID` int(11) DEFAULT NULL,
  `Appointment_Status` varchar(12) DEFAULT NULL,
  `Schedule_ID` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Rule 9:

Logical Data independence

To demonstrate this, I want to show that splitting a table into two tables and creating a view where from both tables will display the original table.

To do so I will use the patient Table:



I create a new table:

```
Create table if not exists Marketing_Preferences ( Patient_ID int auto_increment, Marketing_Preference varchar(3), foreign key (Patient_ID) references patient(patient_id))
```

And insert the values from Patient_ID 1:

```
insert into marketing_preferences values (1,"No")
```

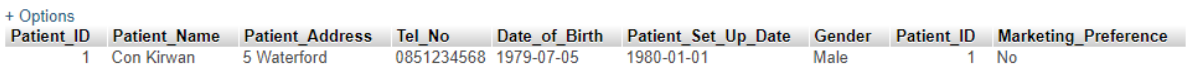
I delete the marketing preference column:

```
alter table patient drop column Marketing_Preference
```

Then when I select from both tables:

```
select * from patient,marketing_preferences where patient.patient_id = marketing_preferences.patient_id
```

I get the same display:

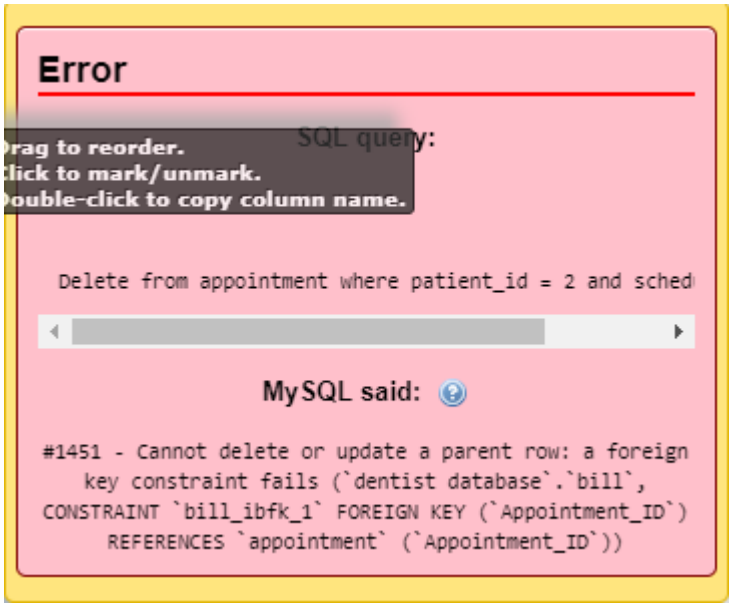


Rule 10:

Integrity Independence:

Here we will try to delete an appointment but only on the appointment table. The primary key on the appointment table is also a foreign key on 4 other tables so a relationship exists with those 4 tables and to delete just the appointment table will damage the integrity of those relationships:

We get the following error



When we enter this code:

```
Delete from appointment where patient_id = 2 and schedule_id = 500
```

Rule 11:

Distributed Independence:

My understanding of this rule is that, once the integrity of the database is maintained, the physical storage of the tables will not impact on the running of the database. For example, if the patient table is stored in one location and the appointment table is stored in another location, once the foreign key patient_id references patient(patient_id) in the correct way, the storage location of patient is irrelevant.

Rule 12:

Non Subversion Rule:

A critically important rule especially now with the implementation of GDPR. Management of the database must follow strict rules to ensure compliance to GDPR. What is important here is that you want to avoid a situation where any individual can by-pass integrity constraints (for example) and insert data which does not have the correct relationships in place. If you had a situation where compliance to GDPR required those relationships to be in place, then the data would be compromised, and you may have a data breach.