CODD Rules:

Rule 1: The Information Rule: Data can 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 andcorresponding 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
```