Codd's 12 rules

Rule 1 – Information Rule

The values in the database are represented consistently and in one way only by inserting a single value per tuple per column in the table identified by a Primary Key. By using a SELECT statement, a every value van be retrieved in exactly one way.

SELECT fname FROM patients WHERE patientID = 1;

Rule 2 – Guaranteed access rule

Every atomic piece of data in the database is related to a table, a Primary Key and a column name. Thus, using a SELECT statement which requires a table and column name in conjunction with a Primary Key statement, this rule is satisfied.

SELECT fname FROM patients WHERE patientID = 1;

Rule 3 – Systematic treatment of null values

The database supports null values for all data types and are distinct from 0 or any other values.

Let there be a database as per the following:

```
CREATE TABLE appointmentExample (
   appointmentID INT(11) AUTO_INCREMENT PRIMARY KEY,
   appointmentDate DATETIME,
   patientID INT,
   dateCreated DATE DEFAULT CURDATE(),
   notes VARCHAR(255)
);
```

The database will be able to handle the following query with null values for the table appointment which has DATETIME, INT, DATE and VARCHAR datatypes and each null is handled and represented in the same way.

```
INSERT INTO appointmentExample (appointmentdate, patientID, dateCreated, notes)
   VALUES
   (NULL, NULL, NULL, NULL);
```

Rule 4 – Dynamic online catalog based on relational model

This rule states that the data about the database must be stored, retrieved and manipulated in the same way other information in database. By using INFORMATION_SCHEMA, we can retrieve metadata about the database such as the following query

```
SELECT * FROM information_schema.TABLES  WHERE TABLE_SCHEMA = 'dentistoffice';
```

This returns information such as the table names, the row format, number of rows, the time it was created as well as the underlying Database engine.

Rule 5 – Comprehensive sub data rule

SQL satisfies the five items described in this rule.

Data Definition: at the point of creating a table, the data type is defined as seen in the following example:

```
CREATE TABLE billExample (
   billID INT(11) AUTO_INCREMENT PRIMARY KEY,
   patientID INT(11),
   issued DATE);
```

View Definition: The View creation query uses a select statement to define the output and is demonstrated with the following example:

```
CREATE VIEW sumPaymentsExample
   AS SELECT billID, SUM(installmentAmount) AS paymentTotal
   FROM payment
   GROUP BY billID;
```

Data Manipulation: This can be performed by inserting data or updating data:

Inserting data can be performed with the following query as an example:

```
INSERT INTO treatment (treatmentName, treatmentPrice, dateChanged)
   VALUES
   ('Referral', 40, '2018-03-20');
```

Data can be updated with the following example:

```
UPDATE bill
set patientID = 3
where billID = 7;
```

Integrity Constraints: The main method of integrity constraints is to use Primary Keys. They are set at the time the table is created as such:

```
CREATE TABLE primaryKeyExample (
   billID INT(11) AUTO_INCREMENT PRIMARY KEY,
   patientID INT(11),
   issued DATE);
```

Transaction Boundaries: Transactions can be created using a START TRANSACTION and COMMIT statement as per the following example:

```
START TRANSACTION;
INSERT INTO treatment(treatmentName, treatmentPrice, dateChanged)
   VALUES
```

```
   ('Late Cancellation fee', 10, '2018-03-20');
COMMIT;
```

Rule 6 – View updating rule

A view created from a single table can be updated in SQL. When updating the data in the view, it will also update the table as per the following:

View creation:

```
CREATE VIEW patientLastName
        AS SELECT patientID, lName, patientEirCode
        FROM patient;
```

The following queries can be run

```
SELECT patientEirCode from patientLastName where lName = 'Doe';

SELECT patientEirCode from patient where lName = 'Doe';
```

The below will be returned

patientEirCode
AX3 FD48

Updating the view:

```
UPDATE patientLastName
set patientEirCode = ‘LH6 DFGH’
where lName = ‘Doe’;
```

Running the aforementioned SELECT queries will show that the UPDATE query has been processed which will then return:

patientEirCode
LH6 DFGH

Rule 7 – High level insert, update, delete

SQL allows for the user to insert, update and delete one or more rows with a single command.

Insert allows for multiple tuples to be inserted into the table by delimiting rows with commas:

```
INSERT INTO primaryKeyExample (patientID, issued)
   VALUES
   (2, '2018-03-20'),
   (4, '2020-04-04'),
   (6, '2019-09-01');
```

Update allows for multiple entries to be updated using the WHERE clause:

```
UPDATE primaryKeyExample
SET patientID = 10
WHERE patientID < 5;
```

Delete allows for multiple entries to be deleted with the WHERE clause, similarly to the UPDATE query:

```
DELETE FROM primaryKeyExample WHERE billid < 2;
```

Rule 8 – Physical data independence

When using the Database Create Commands script with this submission, the logical integrity of the database will not be affected, no matter whether it is installed on my machine or any person who imported it on theirs. Running the following query will return the same results on any machine with the database inserted:

```
SELECT * FROM information_schema.TABLES  WHERE TABLE_SCHEMA = 'dentistoffice';
```

Rule 9 – Logical data independence

Information preserving changes, such as the addition of columns to a table or adding a table to the do not affect the ability to use the database. The following two queries will not affect the user's ability to use the database will not affect the results of any subsequent query

Adding a column to a table:

```
ALTER TABLE patient
        ADD emailAddress varchar(255)
        AFTER phoneNumber;
```

Adding a new table:

```
CREATE TABLE newTable (
   billID INT(11) AUTO_INCREMENT PRIMARY KEY,
   patientID INT(11),
   issued DATE);
```

Rule 10 – Integrity Independence

The database creates and checks integrity constraints at the creation stage. When a table is created, integrity constraints are put in place and are checked when any update is made. The Primary Key from one table can be referenced in another table through defining a Foreign Key. Thus, a Primary Key must be unique and cannot be changed.

For example, a Primary Key must be unique as per the example below.

Table creation:

```
CREATE TABLE primaryKeyExample (
   billID INT(11) AUTO_INCREMENT PRIMARY KEY,
   patientID INT(11),
   issued DATE);
```

Inserting a null into the Primary Key value will not work and will assign a Primary Key automatically

```
INSERT INTO primaryKeyExample (billID, patientID)
        VALUES
        (NULL, 4);
```

Similarly, there will be an error with the following insert statement as the same Primary Key will be used.

```
INSERT INTO bill (billID)
        VALUES
        (3),
        (3);
```

Rule 11 – Distributed Independence

This situation arises when a database is located in more than one place, or multiple databases are merged together. By using network connections as well as choosing the location of the information should be stored (whether based upon security or access speeds), and selecting an appropriate RDBMS and front end, the distribution of data across several locations can be achieved.

Rule 12 – Non-subversion rule

Although it is possible to change each atomic value in the database through SQL, there is no way to bypass the integrity constraints.