Codds Rules

1) The Information Rule

SQL Command(s): SELECT * FROM `appointmentdiary`;
DESC appointmentdiary;

This command can then be applied to the other necessary tables. The rule is effectively demonstrated across all tables in this database, with all values being stored in two dimensional relations.

Rule 2: Guaranteed Access Rule

SQL Command(s): SELECT paymentReceived FROM appoinmentdiary WHERE appointmentID = 50003

The above command uses a unique primary key to identify a value from the attribute 'paymentReceived'.

These three key elements are the necessary components needed to satisfy this rule

INSERT INTO `patient`(`firstName`, `surName`, `contactNumber`, `address`, `overduePaymentSpecify`, `treatmentIDHistory`)

Rule 3: Systematic Treatment of NULL Values

SQL Command(s):

INSERT INTO `patient`(`firstName`, `surName`, `contactNumber`, `address`, `overduePaymentSpecify`, `treatmentIDHistory`)

VALUES ('John', NULL, NULL, 'Chesnut Close', 'Dublin', NULL),

('John', ' ', ' ', 'Chesnut Close', 'Dublin', ' ')

In this instance, the NULL value is supported in the DBMS, however, when I inserted a new set of values representing the lacking data by a blank space, the DBMS also accepted this and added the new row to the table. While NULL values can be implemented with good practice, the rule is violated in this case.

Rule 4: Dynamic online catalog based on the relational model

SQL Command(s)
SELECT * FROM INFORMATION_SCHEMA.TABLES
SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME LIKE 'patient'

This command displayed the system metadata at the logical level in the same format as ordinary data in the DB. In particular I chose to view the structure of the patient table within the data dictionary. This affirms the assumption that the database is self describing and Codd's Rule is met.

<u>Rule 5: Comprehensive Data Sublanguage Rule:</u>

The chosen language SQL for this case must be able to support all the central functions of a DBMS e.g. creating a database, retrieving and entering data, and setting constraints.

The DDL is supported here via the following commands:
ALTER TABLE patient
DROP COLUMN RegisteredOn;

ALTER TABLE treatment
RENAME TO treatmentAndFees;

CREATE (Previously demonstrated)

The DML is also supported as previously demonstrated via SELECT, INSERT, UPDATE commands

<u>Rule 6: View Updating Rule</u>
SQL Command(s): UPDATE newview SET treatmentType = 'Brace fittings'
SELECT * FROM treatmentandfees WHERE treatmentType = 'Brace Fittings'

This rule states that updates to views should also be reflected in the original table from which the value was obtained. An update was allowed to the view (containing the original primary key), which was correctly reflected in the treatmentandfees table. In this instance Codd's rule was met.

In addition, I created a new view of treatmentandfees excluding the primary key from the base table, and made an update to the view - which also reflected in the original table. However Codd's Rule explains that an updateable view must include the primary key of the base table. In this sense, Codd's rule is being violated as the rule definition is not being adhered to.

CREATE view view1 AS SELECT treatmentandfees.treatmentType, treatmentandfees.treatmentFee, treatmentandfees.specialistRequired FROM treatmentandfees

UPDATE view1 SET treatmentFee = 80 WHERE treatmentType = 'Composite Filling'

<u>Rule 7: High-Level Insert, Update, and Delete</u>
This rule has been demonstrated successfully in this database

Rule 8: Physical Data Independence:
While I understand the premise and logic of this rule, I unfortunately could not perform a suitable solution with SQL.

Rule 9: Logical Data Independence:
A way to challenge this rule would be to update a table within the database. If a view has been created from this table previously, the value should also update within the view. This is similar to the operation I carried out for rule six but the other way around.
SQL command(s):

```
UPDATE treatmentandfees SET treatmentFee = 90 where treatmentTypeID = 100
SELECT * FROM view1
```

The change to the table was reflected in the view also.

My initial approach to this question was to partition a table into two separate tables. Then, to further create a view, which in theory would show the result of the join of the two tables. In addition, merging two tables into one was an approach I looked into, but could not carry out these tasks with the SQL commands I tried, so unfortunately could not demonstrate this rule as I intended.

Rule 10: Integrity Independence:
This rule has been demonstrated by the inclusion of constraints when initially creating the tables, which restrict the data being entered. This satisfied Codd's Rule.

An example of a CHECK constraint was also attempted to check the payment value was greater than zero, however produced an error that I could not resolve. I cannot confirm whether this rule could be successfully implemented due to this.

SQL Command(s):

```
specialistID INT NOT NULL AUTO_INCREMENT

ALTER TABLE payment
ADD CONSTRAINT chkPrice CHECK (amount> 0);
```

Rule 11: Distribution independence:
This rule states that the distribution of data on the database out onto multiple physical systems must be possible. In addition, the end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. From research it seems that this rule generally is not met as applications need to connect to a specific DB instance, and data stored in different instances would have to be queried separately.

<u>Rule 12: The non-subversion rule:</u>
This rule requires that alternate methods of accessing the data are not able to bypass integrity constraints, which means that users can’t violate the rules of the database in any way