CODDs RULES

Rule 1 – The information rule.

All the tables in the database uses the structure of attribute(columns) and tuple (rowns) as per the image below. The ERD relation is base on the tables, therefore, information can be retrieve using specific queries while the relation between tables is live and linked by primary keys.

The information can be modified via the tables where the information is stored using the primary key to locate the tuple. On the three tables below 'Treatment_code',' Patient_ID' and 'Appointment_ID' are the primary keys starting from the table on the top left being the first and the top right the second.

Treatment_code	Treament_description	Fees
CC0001	CANCELATION	100
CR0350	Crown	350
FI0080	Filling	80
PE0120	Periodontal	120
PR0100	prophylaxis	100
RC0200	Root Canal	200
WE0300	Wisdom tooth Extraction	300
XR0050	X-ray	50

Patient_ID	First_name	Surname	Gender	Data_of_birth	Address	County	Phone_Number
1	Thiago	Dias	M	1983-03-01	Stonebridge	Dublin	89998889
2	Michael	Hene	M	1968-05-22	Eagle road	Galway	87998833
3	Grainne	OConnor	F	1970-12-20	Marrion Road	Dublin	85997711
4	Roisin	Dale	F	1961-02-11	Shankill lane	Dublin	83112233
5	Ronan	Carril	M	1991-07-01	Black Rock	Cork	85552200
6	Pedro	Novaes	M	1995-08-20	Cabra Park	Wicklow	683665296
7	Roger	Petrovisck	M	1993-02-01	Bray road	Wicklow	87777221

Appointment_ID	Appointment_date	Appointment_time	Patient_ID	Appointment_made_by	Treatment_code	Specialist_ID
1	2018-03-30	13:00:00.000000	1	Drop in	CC0001	9999
2	2018-03-29	13:30:00.000000	2	Phone	PR0100	6666
3	2018-02-28	14:30:00.000000	3	Post	PE0120	8888
4	2018-04-24	17:30:00.000000	4	Post	RC0200	5555
5	0000-00-00	09:30:00.000000	5	Phone	FI0080	9999
6	0000-00-00	13:00:00.000000	5	Phone	WE0300	7777
7	2018-04-15	09:00:00.000000	6	Drop in	XR0050	9999

Rule 2 – The Guarantee Access rule.

The information on can be accessed using a SQL command line SELECT column FROM *table_name* *W*HERE *condition*. Example would be if the select query below is enter on the command line this information will precise as Patient_ID is a primary key.

```
SELECT First_name, Surname, county FROM PATIENT WHERE Patient_ID ='1
```

First_name	Surname	county
Thiago	Dias	Dublin

Following this example all tables have a primary key to guarantee access to precise information.

Rule 3 – Systematic Treatment of Null Value rule.

To demonstrate rule three an incomplete set of information was entered in the PATIENT so that this rule could be demonstrated.

INSERT INTO PATIENT (First_name, Surname , Gender ,Data_of_birth, Address, County, Phone_Number)
Values
(' Paulo', '', ' ', '1960-03-07', ' ', 'Dublin', ' 089880654 ')

```
SELECT * FROM `patient` WHERE patient_ID = '11'
```

Patient_ID	First_name	Surname	Gender	Data_of_birth	Address	County	Phone_Number
11	Paulo			1960-03-07		Dublin	8988064

Mysql treats null varchar(string) as a null not adding anything, however, it adds value 0 to int and date as per the figures below.

INSERT INTO PATIENT (First_name, Surname , Gender ,Data_of_birth, Address, County, Phone_Number)
Values
(' Paulo', '', ' ', '', ' ', 'Dublin', ' ')

```
SELECT * FROM `patient` WHERE patient_ID = '12'
```

Patient_ID	First_name	Surname	Gender	Data_of_birth	Address	County	Phone_Number
12	Paulo			0000-00-00		Dublin	0

An auto increment and not null was used on the primary keys to make sure that have an unique value on the tables.

Rule 4- Dynamic online calalog based on the relational model rule.

The catalog tables are store in the information_schema database and can accessed in the same way as any other database using the SQL command line.
The catalog table for my database project can be accessed with the command line below.

SELECT * FROM information_schema.tables Where table_schema = 'projectg000364674';

This command line will give access to the data dictionary of 'projectg000364674' database tables. Example below:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	ENGINE	VERSION	ROW_FORMAT	TABLE_ROWS	AVG_ROW_LENGTH
def	projectg000364674	appointment	BASE TABLE	InnoDB	10	Compact	7	2340
def	projectg000364674	treatment	BASE TABLE	InnoDB	10	Compact	8	2048
def	projectg000364674	specialist	BASE TABLE	InnoDB	10	Compact	5	3276
def	projectg000364674	payment	BASE TABLE	InnoDB	10	Compact	9	1820
def	projectg000364674	patientbills	VIEW	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
def	projectg000364674	patientappointments	VIEW	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*
def	projectg000364674	patient	BASE TABLE	InnoDB	10	Compact	9	1820
def	projectg000364674	bill	BASE TABLE	InnoDB	10	Compact	6	2730

SELECT * FROM information_schema.columns Where table_schema = 'projectg000364674'; This command line will give access to the data dictionary of 'projectg000364674' database columns. Example below:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE
def	projectg000364674	appointment	Appointment_ID	1	*NULL*	NO	int
def	projectg000364674	appointment	Appointment_date	2	*NULL*	YES	date
def	projectg000364674	appointment	Appointment_time	3	*NULL*	YES	time
def	projectg000364674	appointment	Patient_ID	4	*NULL*	YES	int
def	projectg000364674	appointment	Appointment_made_by	5	*NULL*	YES	varchar
def	projectg000364674	appointment	Treatment_code	6	*NULL*	YES	varchar
def	projectg000364674	appointment	Specialist_ID	7	*NULL*	YES	int
def	projectg000364674	bill	Bill_ID	1	*NULL*	NO	int
def	projectg000364674	bill	Bill_Date	2	*NULL*	NO	date
def	projectg000364674	bill	Appointment_ID	3	*NULL*	YES	int
def	projectg000364674	bill	Patient_ID	4	*NULL*	YES	int
def	projectg000364674	bill	Treatment_code	5	*NULL*	YES	varchar
def	projectg000364674	bill	Fees	6	*NULL*	YES	int
def	projectg000364674	bill	Total_due	7	*NULL*	YES	int
def	projectg000364674	patient	Patient_ID	1	*NULL*	NO	int
def	projectg000364674	patient	First_name	2	*NULL*	YES	varchar
def	projectg000364674	patient	Surname	3	*NULL*	YES	varchar
def	projectg000364674	patient	Gender	4	*NULL*	YES	varchar
def	projectg000364674	patient	Data_of_birth	5	*NULL*	YES	date
def	projectg000364674	patient	Address	6	*NULL*	YES	varchar
def	projectg000364674	patient	County	7	*NULL*	YES	varchar
def	projectg000364674	patient	Phone_Number	8	*NULL*	YES	int
def	projectg000364674	patientappointments	First_name	1	*NULL*	YES	varchar
def	projectg000364674	patientappointments	Surname	2	*NULL*	YES	varchar
def	projectg000364674	patientappointments	Appointment_Date	3	*NULL*	YES	date

05 – The comprehensive data sub language rule.

The DATABASE projectg000364674 was created using a data definition language language or a set of SQL commands. These set of commands allowed us to create and manipulate the structure of the data base as per examples below.

CREATE TABLE

Run SQL query/queries on database information_schema:

```
CREATE TABLE PATIENT (
  Patient_ID int UNIQUE NOT NULL AUTO_INCREMENT,
  First_name varchar(255),
  Surname varchar(255),
  Gender varchar(1),
  Data_of_birth Date,
  Address varchar(255),
  County varchar(255),
  Phone_Number int(9),
  Primary Key (Patient_ID)
);
```

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	Patient_ID	int(11)			No	None	AUTO_INCREMENT
2	First_name	varchar(255)	latin1_swedish_ci		Yes	NULL	
3	Surname	varchar(255)	latin1_swedish_ci		Yes	NULL	
4	Gender	varchar(1)	latin1_swedish_ci		Yes	NULL	
5	Data_of_birth	date			Yes	NULL	
6	Address	varchar(255)	latin1_swedish_ci		Yes	NULL	
7	County	varchar(255)	latin1_swedish_ci		Yes	NULL	
8	Phone_Number	int(9)			Yes	NULL	

The DDL(Data Definition Language) allows the database to be created and manipulated. It allows the database administrator to add integrity constrains such as PRIMARY KEY, UNIQUE and NOT NULL values or modify information using a command line as per example below.

```
alter table bill modify column bill_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
```

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	Bill_ID	int(11)			No	None	AUTO_INCREMENT
2	bill_date	timestamp			No	CURRENT_TIMESTAMP	
3	Appointment_ID	int(11)			Yes	NULL	
4	Patient_ID	int(11)			Yes	NULL	
5	Treatment_code	varchar(255)	latin1_swedish_ci		Yes	NULL	
6	Fees	int(11)			Yes	NULL	
7	Total_due	int(11)			Yes	NULL	

On the view definition the structure query language and retrieve the information required and display in an organized way for the user view. Example below:

```
SELECT * FROM `bill`
```

Bill_ID	bill_date	Appointment_ID	Patient_ID	Treatment_code	Fees	Total_due
1	2018-03-30 00:00:00.000000	1	1	CC0001	100	100
2	2018-03-29 00:00:00.000000	2	2	PR0100	100	100
3	2018-02-28 00:00:00.000000	3	3	PE0120	120	120
4	2018-04-24 00:00:00.000000	4	4	RC0200	200	200
5	2018-03-01 00:00:00.000000	5	5	FI0080	80	80
6	2018-04-15 00:00:00.000000	6	5	WE0300	300	300

```
Create view VIEW_1 as Select PATIENT.First_name, PATIENT.Surname, APPOINTMENT.Appointment_Date, APPOINTMENT.Appointment_time from PATIENT inner join APPOINTMENT on PATIENT.Patient_ID = APPOINTMENT.Patient_ID
```

```
SELECT * FROM `view_1`
```

First_name	Surname	Appointment_Date	Appointment_time
Thiago	Dias	2018-03-30	13:00:00.000000
Michael	Hene	2018-03-29	13:30:00.000000
Grainne	OConnor	2018-02-28	14:30:00.000000
Roisin	Dale	2018-04-24	17:30:00.000000
Ronan	Carril	0000-00-00	09:30:00.000000
Ronan	Carril	0000-00-00	13:00:00.000000
Pedro	Novaes	2018-04-15	09:00:00.000000

Moreover, the SQL permits the user to manipulate data.

```
UPDATE `projectg000364674`.`patient` SET `First_name` = 'Pedro' WHERE `patient`.`Patient_ID` = 1;
```

Patient_ID	First_name	Surname	Gender	Data_of_birth	Address	County	Phone_Number
1	Pedro	Dias	M	1983-03-01	Stonebridge	Dublin	89998889

Finally, it preserves the transaction boundaries (begin, commit and rollback)

12	Paulo	Diniz		0000-00-00		Dublin	89949002

```
update patient set phone_number ='123123123123123' where patient_id ='12'
```

The tuple in this case is not update. The transaction rolls back as the data type for it is integer of length 9. However it is committed when the follow is enter on the SQL command line.

12	Paulo	Diniz		0000-00-00		Dublin	88888899

Rule 6 – View Updating Rule.

This rule is demonstrated on the images below as a proof that the view is updatable and propagates the updated to the base tables guarantying the integrity of the data.

```
SELECT * FROM `view_1`
```

First_name	Surname	Appointment_Date	Appointment_time
Thiago	Dias	2018-03-30	13:00:00.000000
Michael	Hene	2018-03-29	13:30:00.000000
Grainne	OConnor	2018-02-28	14:30:00.000000
Roisin	Dale	2018-04-24	17:30:00.000000
Ronan	Carril	0000-00-00	09:30:00.000000
Ronan	Carril	0000-00-00	13:00:00.000000
Pedro	Novaes	2018-04-15	09:00:00.000000

```
update view_1 set First_name ='Peter' where First_name ='Thiago'
```

```
SELECT * FROM `view_1`
```

First_name	Surname	Appointment_Date	Appointment_time
Peter	Dias	2018-03-30	13:00:00.000000
Michael	Hene	2018-03-29	13:30:00.000000
Grainne	OConnor	2018-02-28	14:30:00.000000
Roisin	Dale	2018-04-24	17:30:00.000000
Ronan	Carril	0000-00-00	09:30:00.000000
Ronan	Carril	0000-00-00	13:00:00.000000
Pedro	Novaes	2018-04-15	09:00:00.000000

```
SELECT * FROM `patient
```

Patient_ID	First_name	Surname	Gender	Data_of_birth	Address	County	Phone_Number
1	Peter	Dias	M	1983-03-01	Stonebridge	Dublin	899503918

Rule 07 – High level insert, update and delete.

The rows were inserted using a high level insert data allowing multiple information being inserted on the different columns with a single command . example below:

INSERT INTO APPOINTMENT (Appointment_date, Appointment_time, Patient_ID, Appointment_made_by, Treatment_code, Specialist_ID)

Values

('2018-03-30', '13:00:00', '1','Drop in', 'CC0001', '9999'),

('2018-03-29', '13:30:00', '2','Phone', 'PR0100', '6666'),

('2018-02-28', '14:30:00', '3','Post', 'PE0120', '8888'),

('2018-04-24', '17:30:00', '4','Post', 'RC0200', '5555'),

('2018-04-31', '09:30:00', '5', 'Phone', 'FI0080', '9999'),

('2018-04-31', '13:00:00', '5', 'Phone','WE0300', '7777'),

('2018-04-15', '09:00:00', '6', 'Drop in','XR0050','9999');

Mysql database also allow data to be delete or updated on the same way as demonstrate below.

Updating a single table or view with multiple columns .

```
update appointment set Appointment_date = '2018-05-01', Appointment_time = '10:00:00', Appointment_made_by = 'Post', Treatment_code = 'CR0350' whereappointment_id ='7'
```

7	2018-05-01	10:00:00.000000	6	Post	CR0350	9999

```
UPDATE Patientbills SET payment_date = '2018-05-03', amount_paid='40' where payment_id ='10'
```

First_name	Surname	Bill_ID	Bill_Date	Total_due	Patient_ID	Payment_ID	Payment_Date	Amount_paid
Peter	Dias	1	2018-03-30 00:00:00.000000	100	1	10	2018-05-03	40

Rule 8 – Physical data independence.

In order to demonstrate this rule a new table would be created on the new server database where all information would moved using the export and then the import so that a copy of the database could be inserted. After copy is completed is complete used would use the database on the new server in the same way as on the previous server.

Rule 9 – Logical data independence.

After dropping a primary key at table patient the DMBS does allow user to modify tuple in which attributes are being used as foreign keys on other tables.

```
ALTER TABLE patient drop primary KEY
```

Error

SQL query:

```
ALTER TABLE patient drop column patient_ID
```

MySQL said:

```
#1829 - Cannot drop column 'Patient_ID': needed in a foreign key constraint 'appointment_ibfk_2' of table 'projectg000364674.appointment'
```

Error

SQL query:

```
ALTER TABLE appointment drop FOREIGN KEY treatment_code
```

MySQL said:

```
#1091 - Can't DROP 'treatment_code'; check that column/key exists
```

Run SQL query/queries on database projectg000364674:

```
1 ALTER TABLE appointment drop FOREIGN KEY treatment_code
```

Error

SQL query:

```
update patient set patient_id = '3' = 'John' where patient_id = '1'
```

MySQL said:

```
#1451 - Cannot delete or update a parent row: a foreign key constraint fails (`projectg000364674`.`appointment`, CONSTRAINT `appointment_ibfk_2` FOREIGN KEY (`Patient_ID`) REFERENCES `patient` (`Patient_ID`))
```

```
ALTER TABLE patient ADD Constraint PK_Surname Primary key(first_name, Surname)
```

Error

SQL query:

```
alter table appointment
add foreign key (Patient_ID) references patient (First_name, Surname)
```

MySQL said:

```
#1239 - Incorrect foreign key definition for 'foreign key without name': Key reference and table reference don't match
```

As the table are linked logically the DBMS avoid inconsistency on the database not allowing the constraint to be changed.

Rule 10 – Integrity independence.

The database was created using a data integrity rule with PRIMARY KEYS, FOREIGN KEY AND NOT NULL avoiding inconsistency on the data as demonstrated on the rule 9.Examples below.

CREATE TABLE PATIENT (
Patient_ID int UNIQUE NOT NULL AUTO_INCREMENT,
Primary Key (Patient_ID)

CREATE TABLE APPOINTMENT (
Appointment_ID int UNIQUE NOT NULL AUTO_INCREMENT,
Primary key (Appointment_ID),
Foreign key (Treatment_code) REFERENCES TREATMENT(Treatment_code),
Foreign key (Patient_ID) REFERENCES PATIENT(Patient_ID),
Foreign Key (Specialist_ID) REFERENCES SPECIALIST(Specialist_ID)