CODD'S 12 RULES

RULE -1: The Information Rule

All information in the relational database is represented in exactly one and only one way—by values in tables

This is the basis of the relational model. Data in SQL exist in the form of tables containing rows and columns, the tables have table names, and column names. The column data types are also specified in the same table within the database. The data are accessed by SQL queries.

The SELECT statement is used to access data from a database.

Consider the following query:

```
SELECT patientName, appDate FROM `appointment` WHERE status = "Confirmed";
```

It will output the name of the patients with appointment date who have "confirmed" their dental appointment.

RULE-2: The Guaranteed Access Rule

Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name

This rule specifies the importance of primary key. As per the relational theory the primary key value is guaranteed to be unique. A column (attribute) value should not be directly accessed without specifying the table and primary key. Therefore, to access data from a relational database, SQL query- SELECT must contain *primarykey_value*, *column_name* and *table_name* (to produce the accurate information).

For example; to get the next of kin phone number of patient MARY JOYCE the following query can be used:

```
SELECT n_o_k_Phone FROM patient WHERE patient_ID=105;
```

RULE-3: Systematic Treatment of Null Values

NULL values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational RDBMS for representing missing information in a systematic way, independent of data type.

This rule states how to handle the NULL values in RDBMS. SQL assigns NULL to the cell (regardless of data type), if no data exist there. NULL values in SQL is not meant for blank spaces or zero, but a distinct way to represent missing data.

The following query will output the name of the patients who have a prescription, and is an example of IS NOT NULL operator:

```
SELECT prob_ID,appNumber,patientName FROM `dentalproblem` WHERE prescription
 IS NOT NULL;
```

RULE-4: Dynamic Online Catalog Based On the Relational Model

The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data

This rule talks about data dictionary/metadata. Data dictionaries (metadata) are are composed of a set of tables, identical in properties to the tables used for data. That is the relational database must be self-describing.

MySQL provides several ways to obtain database metadata:

- SHOW statements : SHOW DATABASES or SHOW TABLES
- Tables in the INFORMATION_SCHEMA database
- Command-line programs : mysqlshow , mysqldump

The following query is an example of how to use SHOW statement to list the tables of ***dentalSurgery***:

```
SHOW TABLES FROM dentalsurgery;
```

To display a descriptive information about tables in the given database,

```
SHOW TABLE STATUS FROM dentalsurgery;
```

It is possible to retrieve metadata about the objects within a database by using the INFORMATION_SCHEMA views. INFORMATION_SCHEMA is based on the

SQL standard and some of the content is MySQL-specific. Therefore, it is more portable than the various SHOW statements. The followings is an instance of using INFORMATION_SCHEMA:

```
USE dentalsurgery; SELECT * FROM INFORMATION_SCHEMA.TABLES;
```

RULE-5: The Comprehensive Data Sub-language Rule

A relational system may support several languages and various modes of terminal use. However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible: a. data definition b. view definition c. data manipulation (interactive and by program) d. integrity constraints e. authorization f. transaction boundaries (begin, commit, and rollback)

This rule insists the existence of a relational database language, (like SQL), to manipulate data. SQL (a structured query language) uses

- CREATE to create tables / views etc.
- SELECT to access the records of tables/views,
- INSERT/DELETE/UPDATE etc to manipulate the records,
- GRANT and REVOKE to provide security by giving different level of access rights.
- For transaction control sets boundaries (BEGIN, COMMIT, ROLLBACK etc).
- It also implements **constraints (primary key, foreign key, NULL)** to provide integrity and consistency.

An example of UPDATE (applied in the ***dentalSurgery***) is shown below. It is used to update the table "patient".

```
UPDATE `patient` SET `billOverdue` = '50.00' WHERE `patient`.`patient_ID` =
102;
```

RULE-6: The View Updating Rule

All views that are theoretically updateable are also updateable by the system

A view is a virtual table. In SQL a view can be updated using CREATE OR REPLACE VIEW command, but it is possible only when we don't update more than a single table in the statement; neither can we update a derived or constant field.

[But by using INSTEAD OF triggers (SQL server 2000 onwards) such tasks can be carried out seamlessly].

For the ***dentalSurgery*** database we can create a view by using the query:

```
CREATE VIEW overdue AS SELECT patient_ID, patientName, billOverdue F
ROM patient WHERE billOverdue>0;
```

It can be updated by the following query and the resultant view will have information of patients who have an overdue of more than 100 Euro:

```
CREATE OR REPLACE VIEW overdue AS SELECT patient_ID, patientName, bi
llOverdue FROM patient WHERE billOverdue>100;
```

RULE-7: High level Insert, Update, and Delete

The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data

It emphasises the set-oriented characteristics of relational databases when using a query language. That means rows can be treated as sets when executing insert, delete, and update operations. It is also applicable to set operations like UNION/ INTERSECTION. All these operation should not be restricted to a single table or row at a time. The SQL language has INSERT, UPDATE, and DELETE statements comply with this rule.

Consider the following query:

INSERT INTO `visitcard` (`patientName`, `appNumber`, `problemName`, `appDate`, `appTime`, `medication`, `prescription`, `spName`, `work_No`) VALUES ('Patric Kennedy', 'MAR/12', 'Toothache', '2019-03-15', '10:30:00', NULL, 'Panadol', NULL, '1001'), ('James PJ', 'APR/10', 'Toothache', '2019-04-04', '15:00:00', 'Panadol', 'amoxicillin', NULL, '1010'),

('Anna Pitt', 'APR/11', 'Toothache', '2019-04-24', '09:00:00', NULL, NULL, NULL, NULL),

('Brian Cowel', 'APR/13', 'Toothache', '2019-04-24', '10:45:00', NULL, NULL, NULL, NULL),

('James PJ', 'APR/14', 'Cavity', '2019-04-25', '11:30:00', NULL, NULL, NULL, NULL),

('Mary Joyce', 'APR/15', 'Toothache', '2019-04-25', '15:00:00', NULL, NULL, NULL, NULL);

It will add 6 rows to the table “visitcard”.

Similarly, the query:

DELETE FROM ‘visitcard’;

It will delete all records from the table “visitcard”, without affecting its structure, attributes, and indexes.

RULE-8: Physical Data Independence

Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods

Physical data independence is meant by the ability to change the physical schema without changing the logical schema.

This rule specifies with the lowest level of data abstraction- the physical level data independence. Users need not be concerned about how the data is stored and how it’s accessed. This is possible, since DBMS maps the conceptual schema to the host system; the internal layer of the database. The storage management is the OS’s task, but users access the database through DBMS (uses queries).

SQL use **indexes** for data retrieval.

The following is an example of how to create an index to “patientName” in “patient” table:

CREATE INDEX indx_patientName ON patient (patientName);

It is possible to create indexes on multiple columns by listing column names in parentheses and separated by commas.

RULE-9 Logical Data Independence

Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables

This is similar to rule 8. Any change (such as adding a table) to the logical schema, should not reflect the physical schema (user’s view). That is, the ability to change the logical schema without affecting the physical schema.

The tables in relational database are logically independent of each other, thus no impact on any other table and users or applications. If we split tables, we can use UNION and produce a view to the user.

The following SQL code demonstrates the use of ALTER TABLE to create a FOREIGN KEY CONSTRAINT to the table “work”:

```
ALTER TABLE work ADD FOREIGN KEY (treat_Number) REFERENCES treatment (treat_
Number);
```

RULE-10: Integrity Independence

Integrity constraints must be definable in the RDBMS sub-language and stored in the system catalogue and not within individual application programs

It states that database should be able to apply integrity rules by using its query languages. The keys and constrains must be strong enough to handle the integrity without depending on external applications. A RDBMS should support at least the entity integrity, and referential integrity.

Data integrity- assuring the accuracy and consistency of data over its entire life cycle - is maintained in SQL (possible to specify data types- **integer, char, varchar, boolean,** etc).

The queries below are an instance of application of this rule in the database created as part of this project.

1. CREATE TABLE treatment (treat_Number varchar(10) NOT NULL, patientName varchar(50) NOT NULL, problemName varchar(50) NOT NULL, treat_Date date NOT NULL, treat_Time time NOT NULL, dentist_ID int(10) NOT NULL, PRIMARY KEY (treat_Number));
2. CREATE TABLE work (work_No int(10) NOT NULL, process varchar(50) NOT NULL, notes varchar(300) NOT NULL, status varchar(50) NOT NULL,treat_Number varchar(50) NOT NULL,CONSTRAINT PK_work PRIMARY KEY (work_No, process));
3. `ALTER TABLE work ADD FOREIGN KEY (treat_Number) REFERENCES treatment ( treat_Number);`

Here it is ensured that treat_Number is NOT NULL and UNIQUE (A PRIMARY KEY constraint automatically has a UNIQUE constraint). The similar case is applied to the table "work".

RULE-11: Distribution Independence

A relational DBMS has distribution independence

According to this rule the database language should have the ability to manipulate data located on other computer systems.

RULE-12: Non Subversion Rule

If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity or constraints expressed in the higher level relational language (multiple-records-at-at-time)

As per this rule users can not be able to bypass integrity constraints of the database in any manner.

References:

1. The definitions are taken from: Database Design and Development (Lecture Notes, Week 11, 12) by Dr. Owen Foley, GMIT, Galway
2. https://www.tutorialcup.com/dbms/codds-rule.htm
3. http://www.informit.com/articles/article.aspx?p=2036581&seqNum=7
4. https://www.mssqltips.com/sqlservertutorial/179/sql-server-informationschema-views-tutorial/
5. https://www.mssqltips.com/sqlservertip/1804/using-instead-of-triggers-in-sql-server-for-dml-operations/
6. https://www.w3schools.com/sql/sql_view.asp
7. https://www.w3schools.com/sql/sql_create_index.asp
8. https://www.webopedia.com/TERM/C/Codds_Rules.html