Derek Higgins
g00398357@gmit.ie

Demonstration of Codd’s Rules

Rule #1 - The Information Rule

E.g. Demonstrated by: `SELECT address FROM patients WHERE PatientID = '1';`

The essence of this rule is that every item of data stored in the database is represented only in one way, that is by values in tables. To adhere to this rule it is not possible to store data or information in any way other than data in tables.

This is demonstrated by queries such as `SELECT * from patients;` which retrieves data of patients from the patients table or by using an individual patients key their address can be retrieved `SELECT address FROM patients WHERE PatientID = '1';` No information such as patient photos, xrays etc are stored directly in the database.

Rule #2 - The Guaranteed Access Rule

E.g. Demonstrated by: `SELECT name FROM patients;`

This rule implies that all data can be accessed in the table in a logical and uniform manner, typically in the format of combining the Table Name, Primary Key and Column name with the possibility of either or both Primary Key and Column Name being omitted for the retrieval of multiple rows of data. An example of this is the retrieval of patient names `SELECT name FROM patients;`

Rule #3 - Systematic Treatment of Null Values

E.g Demonstrated by: `SELECT cancelled FROM appointments WHERE patientID = '5' AND DATE = '2022-04-28' AND TIME = '14:00:00';`

The rule requires the database to distinguish between empty data values and unknown data values so that missing information is represented in a systematic manner. An empty or unknown data point should not be given the value ‘0’ for example a this may imply a state whereas the value null would indicate that the data is unknown.

For example the statement above returns a value of ‘0’ indicating that an appointment has not been cancelled. This ‘0’ is a default value created when the appointment is made however, were the value to be NULL it would imply that it is unknown whether the appointment is cancelled or not.

Rule #4 - Dynamic Online Catalog based on the Relational Model

E.g. Demonstrated by: `` SHOW TABLE STATUS FROM `information_schema`; ``

This rule requires that a relational database contains tables describing the data, i.e. metadata and that this data is stored and accessed using the same relational data.

Rule #5 - The Comprehensive Data Sub Language Rule

E.g. Demonstrated by: `INSERT INTO patients (NAME) VALUES ('Bruce');` (i.e. DML)

This rule requires that the database implements a language like SQL that has a set syntax, supporting data definition and manipulation, security and integrity contraints among other things.

Rule #6 - The View Updating Rule

E.g. Demonstrated by: `UPDATE next_Week SET staffID = '2';`

Views are virtual tables which can represent data from different underlying tables. This rule infers that data can be updated in a view and this will update data in the corresponding tables. For example, consider the following query `UPDATE next_Week SET staffID = '2';` This acts on the view of next week's appointments and would change the dentist working on all of the appointments to '2' updating the value in the appointments table. This may be useful if the dentist was unexpectedly ill and a stand in was taking his appointments.

Rule #7 - High Level Insert Update and Delete Rule

E.g. Demonstrate by: `INSERT INTO patients (NAME) VALUES ('Jess'), ('Freddy')`

This rule emphasises that modification of data using insert, update and delete should be permitted to apply to sets of rows as opposed to individual entries and that implementations which act on a single row are prohibited. As an example the SQL given for Rule #5 above can be modified to create multiple rows as so `INSERT INTO patients (NAME) VALUES ('Jess'), ('Freddy')`;

Derek Higgins
g00398357@gmit.ie

Rule #8 - Physical Data Independence

This rule infers that changes at a physical or low level as to how the data is stored, moving of hard drive, moving of folder etc would not change how data is accessed on the logical level, i.e. tables, columns, rows will remain the same.

Rule #9 - Logical Data Independence

E.g. Demonstrated by: `` CREATE TABLE `bills`; ``

This rule means that if new columns are added or even columns removed from a table, that the user view or programs which do not rely on the columns would remain unchanged. This would also apply to the addition of new entities and relationships such as the creation of a new table demonstrated above not affecting existing tables and data.

This would imply that references to data are absolute in nature and not relative to other columns or rows.

Rule #10 - Integrity Independence

E.g. Demonstrated by:

```
CREATE TABLE `bills` ( `billno` INT(11) NOT NULL
AUTO_INCREMENT, `amount` FLOAT NULL DEFAULT NULL,
PRIMARY KEY (`billno`) );
```

This rule requires that integrity constraints must be declared as part of the database structure, i.e. that they can be defined and stored as part of the data dictionary.

An example of this is providing integrity constraints when a new table is created as demonstrated above where the PRIMARY KEY billno is constrained to be not null.

Rule #11 - Distributed Independence

This rule infers that the distribution of the database and its parts should not be apparent to end users. An example of this would be using the PARTITION command which can be used to separate data on tables but that queries on the table be handled normally.

Rule #12 - Non Subversion Rule

This rule prohibits database systems from providing a low level language which can be used to edit data potentially bypassing security or integrity constraints. Some RDMBS' allow constraints to be turned off for quicker bulk importing or copying of data which violates this rule.