Codds Rules

Rule 1: The Information Rule

- SELECT lastName, firstName, dateOfBirth FROM patient WHERE patientId = 5;

This query demonstrates how unique values are retrieved using a primary key as an identifier.

The values should always be displayed in table format and the ordering of the data does not matter.

Rule 2: The Guaranteed Access Rule

- SELECT date FROM appointment WHERE appointmentId = 3;

This query demonstrates how the column name, table name and primary key are the only 3 items needed to retrieve data.

Rule 3: Systematic Treatment of Null Values

- SELECT patient.firstName, address.addressLine1, address.addressLine2, address.town, address.county, patientChart.specialistReportId FROM patient INNER JOIN address ON patient.address = address.addressId INNER JOIN patientchart ON patient.patientId = patientchart.patientId;

This query demonstrates how null values are supported in the database management system. Null values represent missing or unknown data in a systemic way regardless of the data type.

This query returns null in place of a char datatype in the addressLine2 column and null in place of an int datatype in the specialistReportId column.

Rule 4: Dynamic Online Catalog based on the relational model

- SELECT * FROM TABLES WHERE TABLE_SCHEMA = 'dentaldatabase';

This query demonstrates how the same tools (SQL) are used to retrieve data about the data (metadata) in the database.

The metadata is also displayed in the form of tables.

Rule 5: The Comprehensive Data Sub Language Rule

The queries below demonstrate the use of SQL within the database management system to define, manipulate and retrieve data.

DDL Queries

- CREATE TABLE examplePatient (
  examplePatientId INT NOT NULL AUTO_INCREMENT,
  firstName CHAR(30),
  lastName CHAR(30),
  PRIMARY KEY (examplePatientId));

- ALTER TABLE examplePatient ADD dateOfBirth DATE;

- CREATE TABLE exampleAppointment (
  exampleAppointmentId INT NOT NULL AUTO_INCREMENT,
  examplePatientId INT,
  date DATE,
  time TIME,
  FOREIGN KEY (examplePatientId) REFERENCES examplePatient(examplePatientId),
  PRIMARY KEY (exampleAppointmentId));

DML Queries

- INSERT INTO examplePatient (firstName, lastName, dateOfBirth) VALUES
  ('Daniel', 'Clyne', '2000-11-03'),
  ('Joseph', 'Greaney', '1995-03-17'),
  ('Michael', 'Fahy', '1998-10-08');

- INSERT INTO exampleAppointment (examplePatientId, date, time) VALUES
  (1, '2021-04-01', '14:30'),
  (2, '2021-04-05', '09:30'),
  (3, '2021-04-07', '15:00');

- UPDATE examplePatient SET dateOfBirth = '1988-12-16' WHERE examplePatientId = 1;

- SELECT firstName, lastName FROM patient WHERE patientId = 1;

- DELETE FROM exampleAppointment WHERE examplePatientId = 2;

Rule 6: The View Updating Rule

- CREATE VIEW olderPatient AS SELECT * FROM patient WHERE dateOfBirth < '1990-01-01';

- UPDATE olderPatient SET lastName = 'abcdefg' WHERE patientId = 1;

These queries demonstrate that when a view is created and then an update is made to the data in the view, that update will also be reflected in the base table.

Rule 7: High Level Insert Update and Delete Rule

- UPDATE treatment SET treatmentPrice = treatmentPrice + 500 WHERE treatmentPrice > 50 ;

This query demonstrates how multiple rows in a table can be modified using a single query.

In this case the query updates the price of all dental treatment types adding 500 to all treatments that are greater than 50.

Rule 8: Physical Data Independence

This rule states that the physical location of the data can be changed but it can still be accessed using the same queries. The relocation of the data should have no effect on the applications used to query it as the data storage is independent of the way it is accessed. A database can be transferred from one hard drive to another and the applications can still interact with the data as usual.

Rule 9: Logical Data Independence

- CREATE TABLE helloWorld (
  helloWorldId tinyint,
  hello char(5),
  world char(5),
  PRIMARY KEY(helloWorldId));

- ALTER TABLE patient ADD COLUMN age tinyint;

This rule states that if the database is changed at a logical level it will not affect the existing application or queries.

The above table helloWorld can be added to the database and this will not affect the data that already exists.

A new column can be added to an existing table and it also would not affect the application that queries the database.

Rule 10: Integrity Independence

- SELECT patient.firstName, address.county, contactInformation.email FROM patient INNER JOIN address ON patient.address = address.addressId INNER JOIN contactInformation ON patient.contactInformation = contactinformation.contactInfoId;

- CREATE TABLE patientTwo (
  patientId int not null auto_increment,
  firstName char(30),
  lastName char(30),
  dateOfBirth date,
  address int,
  contactInformation int,
  primary key (patientId),
  foreign key (address) references address(addressId),
  foreign key (contactInformation) references contactInformation(contactInfoId));

This rule states that the integrity constraints of the database must be defined by the data sublanguage and stored in the data dictionary and not the applications.

For each non null foreign key in a database there must be a matching primary key in the same domain and no primary key can have a null value.

The SELECT query above demonstrates referential integrity where each non null foreign key references a primary key.

When the tables were created a not null constraint was put on the primary keys which enforces entity integrity. The patient table above demonstrates this.

Rule 11: Distributed Independence

This rule states that the database language and applications will not be affected should the data be stored all on one system or split up and stored over multiple systems.

The user of the database should not know the difference.

The distribution of the data is independent of the applications and how they function.

Rule 12 Non-Subversion Rule

The database should not allow any low-level language to alter or bypass any integrity constraints that were defined by a higher-level language.

SQL should be the only method of changing any data in the database.

This can protect a system from any unwanted corruption of data or attacks from a malicious source such as a hacker.