Pawel Zamorski, G00364553@gmit.ie

The 12 Codd's Rules using SQL and MySQL DBMS

Rule 1: The information rule. *All information in a relational data base is represented explicitly at the logical level and in exactly one way – by values in tables.*

```
SHOW TABLES IN g00364553_zamorski;
```
-- Retrieving all tables form selected databases.

Tables are the only structure of a database.

```
SELECT * FROM patient;
```
-- Retrieving all values from selected table.

The relation is a set of attributes and rows. Each value is defined by row and attribute. The order of rows or attributes should not have influence on data. Tables are the only structure of a database. It means that there can not be other structure, ie a few values in the same cell separated by comma or other sign. This may be achieved by the process of normalization.

Rule 2: The guaranteed access rule. *Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.*

```
SELECT lastName FROM patient WHERE patientNumber = 101
```
-- Selecting atomic value using the combination of table name (`patient`) primary key value (`patientNumber = 101`) and column name (`lastName`).

Only accessing an atomic value by using primary key value guarantees the correctness of a result. Every row in a table must have primary key. Primary key must by unique. It is guaranteed by the DBMS. The DMBS will not allowed to insert new data with the repeating primary key, ie:

```
INSERT INTO patient VALUES (101, 'John', 'Smith', 'Male', 'address', '+       353 
000', '+484 225', 'email@email.com', 'mobile', 1, NULL, NULL)
```
-- The above statement will produce an error: "Duplicate entry '101' for key 'PRIMARY'"

Rule 3: Systematic treatment of null values. *Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.*

```
SELECT * FROM patient WHERE address IS NULL;
```

```
-- Selecting all data from table (patient) where are the NULL values
SELECT * FROM patient WHERE address IS NOT NULL;
-- Selecting all data from table (patient) where are no the NULL values.
```

DBMS fully supports handling of NULL values in a systematic way, independently of the data type. NULL value should be consider as a missing/unknown information. Such information can not be represented as zero or empty string.

```
SELECT CONCAT(lastName, address, landline) FROM patient;
-- Concatenating strings. The result will be NULL wherever the is the NULL value.
```

Similarly, mathematic operations on NULL values result NULL value.

Rule 4: Dynamic online catalog based on the relational model. *The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.*

```
SELECT * FROM INFORMATION_SCHEMA.TABLES;
-- Selecting meta data

SELECT TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE
       FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'g00364553_zamorski';
-- Selecting particular meta data
```

The meta data are stored in the same way as users data: in tables, columns and rows. Data Dictionary Tables (system catalog) are invisible. The access to metadata is provided by `INFORMATION_SCHEMA` database containing a set of read-only views. The same language, SQL is used for accessing this data.

`INFORMATION_SCHEMA` has set of different views, ie:

- `SELECT * FROM INFORMATION_SCHEMA.TABLES` provide information about tables
- `SELECT * FROM INFORMATION_SCHEMA.COLUMNS` provide information about columns
- `SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS` provide information about table constraints
- `SELECT * FROM INFORMATION_SCHEMA.VIEWS` provide information about views

Rule 5: The comprehensive data sub language rule. *A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items: 1. Data definition. 2. View definition. 3. Data manipulation (interactive* *and by program). 4. Integrity constraints. 5. Transaction boundaries (begin, commit and rollback).*

1. Data definition

```
CREATE TABLE Employee (
   employeeNumber   SMALLINT(3) UNSIGNED NOT NULL,
   branchNumber     SMALLINT(3) NOT NULL,
   firstName        VARCHAR(30) NOT NULL,
   lastName         VARCHAR(30) NOT NULL,
   mobile           VARCHAR(30) DEFAULT NULL,
   email            VARCHAR(30) DEFAULT NULL,
   jobTitle         VARCHAR(15) NOT NULL,
   login            VARCHAR(15) NOT NULL,
   password         VARCHAR(15) NOT NULL);
```

-- An example of DDL – Data Definition Language. The statement creates a table.

2. View definition

```
CREATE VIEW MallowEmployee AS SELECT city, firstName, lastName
      FROM branch, employee
      WHERE branch.branchNumber = 1
            AND branch.branchNumber = employee.branchNumber;
```

-- An example of DDL – Data Definition Language for creating the view. The statement creates a view. It is possible to select data from tables vertically, horizontally and from more than one table.

3. Data manipulation

```
SELECT * FROM employee;
```

-- An example of DML – Data Manipulation Language. The statement selects all data from a table.

4. Integrity constraints

```
ALTER TABLE Employee ADD CONSTRAINT Employee_fk FOREIGN KEY (branchNumber) REFER
ENCES Branch (branchNumber);
```

-- An example of integrity constraint. The statements adds a foreign key to a child table and refers to a primary key of a parent table.

5. Transaction boundaries (begin, commit and rollback).

```
START TRANSACTION;
INSERT INTO branch VALUES (3, '87 New branch address', 'Cork', '+00353 87 854
85');
UPDATE employee SET branchNumber = 3 WHERE branchNumber = 2;
DELETE FROM branch WHERE branchNumber = 2;
COMMIT;
```

-- An example of transaction: starting transaction, inserting data into a table, updating data in a table, committing transaction. Above it MySQL uses InnoDB as the default engine that supports transaction.

All the above statements use SQL language. SQL may be used interactively or by application. SQL language comprehensively fulfils requirements for creating/altering/dropping tables and views, selecting/inserting/updating/deleting data, setting/altering integrity constraints and supporting transactions.

Rule 6: The view updating rule. *All views that are theoretically updatable are also updatable by the system.*

```
CREATE VIEW BasicPatient AS SELECT patientNumber, firstName, lastName FROM patient
;
```
-- Creating an updatable view: the view is created from one base table, it is not created from another view which is not updatable, includes primary key column of the base table, consist of columns of the base table, not of a constant, expression or aggregate function, doesn't have a group by or having clause, includes all columns that requires the data (doesn't have a specified default value).

```
INSERT INTO basicpatient VALUES (200, 'Gerald', 'Kean');
```
-- Inserting values to the view table

DBMS fulfils the view updating rule. However, not every table view is updatable. It must fulfil requirements that are described above.

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.*

```
INTO Branch (branchNumber, address, city, phone)
      VALUES (1, '22 Sundown Drive', 'Athlone', '00353-86-777-8789'),
             (2, '988 Harbort Trail', 'Cork', '+86 176 487 6799');
```
-- Inserting two rows into a table by a single statement.

```
UPDATE treatmentoffered SET price = price +(price * 0.1) WHERE employeeNumber = 1;
```
-- Updating a table, adding 10.00 to the price of all treatments. The statements effects all rows.

```
DELETE FROM workschema WHERE workDateTime BETWEEN '2018-04-23 08:00:00' AND '2018-
04-24 11:00:00';
```
-- Deleting rows from a table using a single statement.

All the above operations have effect on more than one row.

Rule 8: Physical data independence. *Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.*

The phpMyAdmin is an application for MySLQ DBMS that enables exporting and importing databases:

- the following will export a database:

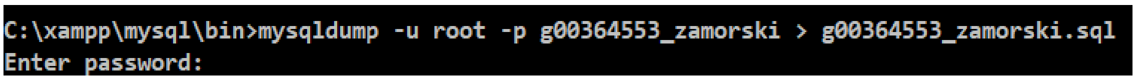
phpMyAdmin → choose the database → Export → choose the file format → Go

- the following will import a database:

phpMyAdmin → Import → choose the file → Go

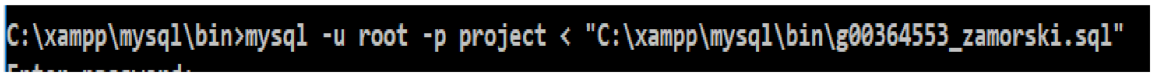
MySQL provides the following programs for making backup:

- mysqldump for a logical backup. It produces SQL statements that reproduce the original database (database structure and content). This backup is machine independent.



```
C:\xampp\mysql\bin>mysqldump -u root -p g00364553_zamorski > g00364553_zamorski.sql
Enter password:
```

  -- Exporting the database.



```
C:\xampp\mysql\bin>mysql -u root -p project < "C:\xampp\mysql\bin\g00364553_zamorski.sql"
```

-- Importing the database.

- MySQL Enterprise Backup that produces a physical backup by coping the original files. A physical backup is faster than logical backup. However, it may be restored only on the machine that has similar hardware.

It is possible to change a device, ie a hard drive, during the process of exporting and importing. This process doesn't have impact on conceptual level or external level.

Moving the folder to the new location doesn't have impact on application programs. It requires changing a path.

There are new version of MySQL, which may result in changing the access method doesn't have impact on application programs.

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.*

```
ALTER TABLE branch ADD branchInfo VARCHAR(255) DEFAULT NULL;
-- Adding new column to a table.
```

Adding new column to a table doesn't affect external level – application programs and terminal activities. Similarly, adding new relation doesn't have impact on external level. The above are information preserving changes. Contrary, removing table or column may not be an information-preserving change and can't be consider as logical data independence.

Rule 10: Integrity independence. *Integrity constraints specific to a particular relational data base must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.*

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS WHERE TABLE_SCHEMA = 'g00364553
_zamorski';
```
- - Selecting information about table constraints.

```
SELECT * FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE TABLE_SCHEMA = 'g00364553_
zamorski';
```
-- Selecting information about column constraints.

All the information about constraints are stored in a system data dictionary (catalog) that is accessible via `INFORMATION_SCHEMA` database.

```
ALTER TABLE branch ADD PRIMARY KEY (branchNumber);
```
-- Adding primary key to the table

```
ALTER TABLE Employee ADD CONSTRAINT Employee_fk FOREIGN KEY (branchNumber) REFEREN
CES Branch (branchNumber);
```
-- Adding foreign key to the table and linking it with a primary key from a parent table

It is possible to change the integrity constraints by using SQL.

MySQL supports the following constraint:
- PRIMAR KEY that is both UNIQUE and NOT NULL, and uniquely defines each row in a table
- FOEIGN KEY that references to a row in a parent table
- UNIQUE, that prevents from inserting non unique values
- NOT NULL that prevents from not inserting a value
- DEFAULT that inserts the default value in case there was no such on specify

MySQL doesn’t support CHECK. However, it is possible to simulate the CHECK condition by using triggers. Triggers are not a part of SQL.

MySQL supports rule 10, besides CHECK.

Rule 11: Distribution independence. *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.*

Distribution independence means that from a user point of view the data should be accessible exactly the same independently they are store in the same physical location or in many.
As an example based on the project, the two dental clinics could have they own databases. The DBMS should provide feature to manage this two databases as one.

Rule 12: Nonsubversion rule. *If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).*

MySQL provides SLQ as a data definition and data manipulation language. There is no low-level language that is supported by MySQL DBMS.
There is phpMyAdmin application that enables modification of single cell without using SQL language by the user. However, phpMyAdmin application creates an appropriate SQL statement.
It is possible to modify the exported file that is not protected ie. encrypted. However, dealing with exported files is out of scope of DBMS. Database administrator should protect such files.