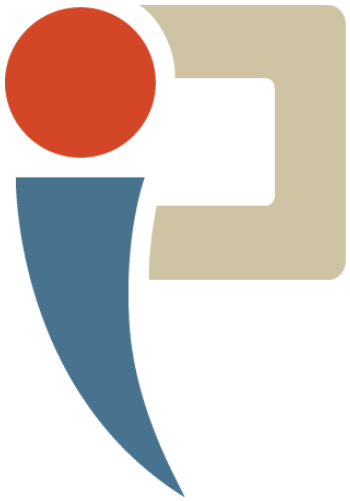
POLUDO INSTITUTE OF TECHNOLOGY & MEDIA



SQL TRAINING

Module 2

Data Integrity and Normalization

Jesse Silva – jessetsilva@gmail.com

2018

Module 2: Data Integrity and Normalization.

The students will be introduced to the basic commands to retrieve data.

Lessons

- Data integrity
- Database Normalization

After completing this module, students will be able to:

- Understand the importance of the data normalization and how to keep the data integrity

DATA INTEGRITY

Enforcing data integrity ensures the quality of the data in the database. Data integrity refers to the overall completeness, accuracy and consistency of data and can be indicated by the absence of alteration between two instances or between two updates of a data record, meaning data is intact and unchanged. Data integrity is usually imposed during the database design phase through the use of standard procedures and rules and can be maintained through the use of various error checking methods and validation procedures. According with Microsoft, “Two important steps in planning tables are to identify valid values for a column and to decide how to enforce the integrity of the data in the column”. Data integrity can be defined into these categories:

- **Entity Integrity**

  There are no duplicate row (registry) in a specific table.

- **Domain Integrity**

  Enforces valid entries for a given column by restricting the type, the format, or the range of values (remember the constraints).

- **Referential Integrity**

  Rows that are used by other records can’t be deleted.

- **User-Defined Integrity**

  Enforces some specific business rules that do not fall into entity, domain, or referential integrity. For example default values or checks.

One of the techniques used to keep the data integrity is the data normalization.

DATA NORMALIZATION

Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data and ensuring data dependencies make sense (only storing related data in a table) and ensures that data is logically stored. The logical design of the database is the core of an optimized relational database. A good logical database design can lay the foundation for optimal database and application performance (Microsoft, 2008).

Basic, Normalize a database design involves using formal methods to separate the data into multiple, related tables. One of the clues that a database is normalized is when you find several tables with few columns each. The Microsoft lists us several benefits that can be get for a normalized database, as example:

- Faster sorting and index creation.
- A larger number of clustered indexes.
- Narrower and more compact indexes.
- Fewer indexes per table.
    - This improves the performance of the INSERT, UPDATE, and DELETE statements.
- Fewer null values and less opportunity for inconsistency.
    - This increases database compactness.

There is a set of normal forms that you can use to organize the database structure. We will aim here in the 3 main ones:

- 1NF - First Normal Form

    It is the most basic set of rules for an organized database and defines the data items required as they will be the columns in a table, avoiding repeating groups of data and making sure

that there will be always a primary key. It is based on three rules:

First Rule: Define the data items. What you must do is understand the that you want to store, organize it into columns, defining what type of data each column contains and put the related columns into their own table. For example, look at the data to be stored below, organize it into columns, define what type of data each column contains and finally put the related columns into their own table (use the fictional tables customer and address).

Column name	Table	Data Type
fullName		
streetName		
gender		
birthDate		
city		
zipCode		
number		
age		

Second Rule: Avoid repeating groups of data, always trying to break the table into others tables and join them using a key if it is possible. The table below shows us what we need to avoid, because there is a lot of repeated information that can easily drive us to a data corruption problem.

ID	NAME	AGE	ADDRESS	ORDERS
1	Jesse	32	1325 maple street	Smartphone
1	Jesse	32	1325 maple street	TV
1	Jesse	32	1325 maple street	LapTop
1	Jesse	32	1325 maple street	Mouse
1	Jesse	32	1325 maple street	Memory
1	Jesse	32	1325 maple street	Drone xy

We can solve it by analyzing and applying the first rule and then create a proper table for each set of information. If you look at the original table and follow the rules, you can solve the problem. One plausible solution: We can create a table to carry only the customer information …

ID	NAME	AGE	ADDRESS
1	Jesse	32	1325 maple street

…and other just to carry the orders

ID	CUSTOMER_ID	ORDERS
1	1	Ipad
2	1	Drone xy
3	1	Mouse Wireless
4	1	TV 43

If we go further we can optimize this tables even more!

Third Rule: Create a primary key for each table. This is a very simple and straight forward rule. Basically, when you apply the first ones you are forced to create the primary keys for its relationships to work.

- 2NF - Second Normal Form

Must follow all the rules from the 1NF and there must be no partial dependencies of any of the columns on the primary key.

- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key.

From *Relational Database Normalization Basics* at http://www.databasedev.co.uk/ we have the following example: “Consider a customer's address in an accounting system. The address is needed by the Customers table, but also by the Orders, Shipping, Invoices, Accounts Receivable and Collections tables. Instead of storing the customer's address as a separate entry in each of these tables, store it in one place, either in the Customers table or in a separate Addresses table”.

In the example below we need to use our knowledge to put this table in accordance with the second normal form (the solution will be presented during the class).

STUD_ID	STUD_NAME	COURSE_ID	COURSE_DETAIL	COURSE_DATE
1	jesse	1	Java	2015-02-13 00:00:00.000
1	jesse	2	SQL	2015-08-07 00:00:00.000
1	jesse	3	QC	2015-12-12 00:00:00.000
1	jesse	4	PHP	2015-04-15 00:00:00.000

- 3NF - Third Normal Form

That Normal form respect all the 2NF + all non-primary fields are dependent on the primary key. To be in accordance with the 3NF the first thing to do is to remove transitive dependencies. Doing that we reduce the amount of duplicated data, simplify the database maintenance and keep the data integrity, just for start!

As Example, Imagine a customer table with the following fields.

1. custID -> Customer identifier
2. custName -> Customer name
3. birthDate -> Customer Birth date
4. street -> Customer address (# included)
5. city -> Customer City
6. state -> Customer State
7. zipCode -> Address ZIP Code
8. emailID -> Customer email identification

Notice that the field ZIP and STREET have a transitive dependency. It's against the Third normal form, so we need to fix it. Work with a partner and try to solve this problem by creating two different tables that can together keep the information and ensure the data integrity. The solution will be given in the classroom.

PRACTICING

1. We discussed here about 3 normal Forms. What are the other ones? Is it important to know how they work? Do people use it very often?

2. Design a set of tables that can store information about a student, his personal information, where he lives, his courses, his grades, his tuition fees and the total duration of his courses. (everything you think is necessary to know about a student)
    - You need only to describe the tables, columns names, types and its relations (no code is necessary for now)

        Try to design it following the 3 normalization rules. Be creative and don't be afraid of errors, we will fix them together as needed.

REFERENCES

- **Books**
  - Fundamentals of database systems – 6th edition
  - Head First SQL - Lynn Beighley
- **Internet**
  - W3Schools @ http://www.w3schools.com/sql
  - Structured Query Language @ http://home.hit.no/~hansha/documents/database/documents/Structured%20Query%20Language.pdf