

[Type text]



APTECH LIMITED

Education Support Services
Academics ACE –Head Office Mumbai

[Type text]

Article Code : ACE/007

Topic : Normalization - First Normal Form

PURPOSE : To improve the performance of the Database Table

SCOPE/OUTCOMES : Ensuring that the performance of Table improves leading to better productivity.

Normalization - First Normal form

In the past, SQL has achieved success and stability due to its core foundations on the relational model. However, certain relational concepts like First Normal Form can take a wild twist when they are discussed within the context of SQL. Since the concept is so often misinterpreted, it is not easy to grasp the fundamental principle that, in SQL, a table must inevitably adhere to 1NF.

In this article, we will try to dispel some of the popular misconceptions associated with 1NF, to explain its structural underpinnings, and to provide some practical guidelines.

Understanding the norms

Why would we want to consider 1NF over any other approach in modeling SQL tables? Essentially, the principles of First Normal Form are nothing more than the properties of a relation in the relational database model ⁽¹⁾. The most important of these are :

- The table should have no duplicates -- no duplicated columns or duplicated rows.
- There must be no significance in the order of either rows or columns.
- In every row, each column must have a single value. Columns in a table, are named and typed.

A table with these properties has the benefits of having a well defined structure, as well as the features of integrity and manipulation -- the hallmarks of the relational model. While these are relatively simple goals, there are some obstacles that get in the way, such as the laxity of features in popular SQL products, the nature of the SQL language and the limited awareness of the database users who use the products and language.

The “Un-normalized”: Limiting the expressive power

Expressive power, the ease and economy with which a query can be expressed, is affected when design principles are neglected. The same is true with 1NF; poor normalization , limits

[Type text]

[Type text]

our expressive power.

Database Bias limits the expressive power of queries. Usually the bias occurs when the design of one or more tables is favored towards a certain set of queries. Consequently, whilst some queries are easy to formulate against such tables, many others may turn out to be difficult to construct. While higher normal forms tend to aggravate this situation, the symptoms are quite pronounced in a table that violates 1NF as well.

Lack of redundancy control also leads to unnecessary complexity in queries. While higher normal forms tend to exhibit the implications better, it is hard for the DBMS to enforce an integrity mechanism in order to control unnecessary duplication if the schema is not even in 1NF. Later in this article we will show some examples that will increase the complexity of query expressions.

The ambiguity of Repeating Groups

Some of the common definitions of 1NF in tutorials misuse the term "repeating groups", or "repeated groups". Usually it goes like this: *If there are no repeating groups in a table, then it is in First Normal Form.*

Many misunderstand the concept of a repeating group and use it to claim that a certain table is in violation of 1NF. Some believe that a set of columns, usually similarly named, that are placed adjacent to each other in a table, and have the same data type constitute a 'repeating group'.

Consider the following table:

A Random Table:

Customer_id	date_1	date_2	date_3	date_4
201	May 21 1961	Feb 1 1989	Sep 15 1993	Jun 6 1999
251	Feb 13 1959	Jul 15 1987	Jul 11 2001	Feb 21 2005
301	Mar 5 1973	Sep 5 1993	Jun 25 1999	May 20 2007
351	Nov 19 1977	Jan 20 1996	Dec 3 1999	Nov 13 2008
401	Jul 17 1968	Dec 19 1988	Mar 24 2003	Aug 5 2005
451	Aug 26 1970	Jun 13 1995	Oct 9 2001	Dec 2 2007

Here the similarity of names and data types of the column with dates gives the illusion of a "repeating group" which may lead people to think that the table violates 1NF. If we consider

[Type text]

APTECH LIMITED

Education Support Services

Academics ACE –Head Office Mumbai

[Type text]

the table to represent the historical data set of terminated employees, and each date represent various events, it may change our perception of the table.

Terminated Employees Table:

emp_id (key)	date_of_birth	hire_date	eligibility_date	date_of_termination
201	May 21 1961	Feb 1 1989	Sep 15 1993	Jun 6 1999
251	Feb 13 1959	Jul 15 1987	Jul 11 2001	Feb 21 2005
301	Mar 5 1973	Sep 5 1993	Jun 25 1999	May 20 2007
351	Nov 19 1977	Jan 20 1996	Dec 3 1999	Nov 13 2008
401	Jul 17 1968	Dec 19 1988	Mar 24 2003	Aug 5 2005
451	Aug 26 1970	Jun 13 1995	Oct 9 2001	Dec 2 2007

Here, we have non-key columns that are all dependent upon the key column. They are not dependent on each other but may operate under constraints that represent specific rules such as 'date_of_termination should be greater than hire_date' or maybe 'hire_date should be at least 18 years later than date_of_birth'; but no such constraints will affect the basic requirements for a table to be in 1NF. But are they still a repeating group, or are they separate columns that represent valid attributes about an employee entity?

Here is another example:

Client	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Jon	852	45	652	455	150	652	154	454	55	658	753	85
Kate	245	900	321	0	15	850	655	513	214	654	455	75
Shyam	256	855	65	160	55	985	745	500	500	565	152	650
Young	655	25	325	350	150	500	32	451	75	55	159	987

This table isn't well-designed and has some expressive limitations but the columns named after the months are not exactly repeating groups and that, by itself, does not cause a violation of first normal form.

So then, what is a repeating group? Well, simply put a repeating group is a column that can accommodate multiple values. The columns in a base table in SQL are explicitly named and typed and therefore can accommodate only a single value of that type (or a null in case of a nullable column which we will discuss in a while). Therefore strictly speaking, a base table in SQL, cannot have a repeating group. A SQL column with an integer data type, for example, cannot contain a repeating group containing a set of several integers.

[Type text]

[Type text]

Interpreting Atomicity

In the past, most people based their understanding of 1NF on the concept of atomicity -- more specifically data value atomicity. That is, each value in a column must be a "single unit of data". In fact some earlier database pioneers have defined atomic data, paraphrased, as "*data that cannot be decomposed into smaller pieces*".

While this was considered appropriate for some time, it has turned out to be very vague and imprecise. For instance, how do we perceive an atomic value? Is a value declared as VARCHAR(10) considered atomic when we can decompose the string into individual characters? How can a value declared as DATETIME, which has year, month and day components and the time portion, have further decomposable elements? Well, the answer is "it depends". Data value atomicity by itself is not an objective yardstick and has no absolute meaning. It depends on how we want to deal with the data. In other words, whether or not a data value is atomic is in the eye of the beholder.

Consider the following table:

key	date	day	month	year
201	19881219	19	12	1988
501	20081021	21	10	2008
801	19610513	13	5	1961

A single date can be viewed as an "atomic" value or as separate day, month, and year values each of which are "atomic" by themselves. If we treat the date as a whole and if we can declare it to the DBMS using a data type that can represent this value, then obviously it can be treated as a single unit of data or a scalar value.

In the same way, any value can be treated as atomic regardless of what the underlying "structure" of the value is -- whether it is an integer, string, array, image, list, collection, audio, movie etc. -- as long as the DBMS supports a type system that allows such values. This is important. If the DBMS has no facility to support such values, obviously the designers will have to invent work around method to "get it done". Now to elaborate this a bit, what is a type

[Type text]

[Type text]

system? In loose terms, it is a facility within a DBMS that allows the user to declare values to be of specific data types.

'...every row and column intersection in that table contains exactly one value of the applicable type, nothing more and nothing less. (The value in question can be arbitrarily complex, it can even be a table...)'

In most cases the database designer, knowingly or unknowingly, is forced to use an existing type to represent a value that should have otherwise been declared using an appropriate data type. Some have mistakenly stated that certain data types such as arrays themselves are the source for repeating groups. After all, a string is nothing more than an array of characters defined as a finite sequence. For instance, *"The definition of 1NF is that the table has no repeating groups and that all columns are scalar values.*

NULLS and 1NF

On a similar note, let us look at another set of rows to see if it violates 1NF. Would the table A be a better representation than Table B?

Table A

client_id	client_name
3256	Jamie Zacharias
2345	Ellen Edward Szeles
9514	Vaikom Muhammad Basheer
8524	Mr T
3698	Bilal M. Azharuddin
7532	Madonna

Table B

client_id	first_name	middle_name	last_name
3256	Jamie	NULL	Zacharias
2345	Ellen	Edward	Szeles
9514	Vaikom	Muhammad	Basheer
8524	Mr T	NULL	NULL
3698	Bilal	M.	Azharuddin
7532	Madonna	NULL	NULL

[Type text]



APTECH LIMITED

Education Support Services

Academics ACE –Head Office Mumbai

[Type text]

Some people might think that Table B on the right would be more accurate representation of reality. That may be correct to some extent, *if the business relies on the significance of each part of the name* and they are used in the model. If the underlying model does not care about the individual parts of the name, then Table A on the left is obviously better designed. Then we have the introduction of NULLs, which is an issue with respect to Table B which we rarely talk about in SQL circles -- in strict terms, NULLs violate First Normal Form. Remember the underlying principle that each value is of the declared type of the column. So this boils down to what a NULL is, in this regard. Most database experts consider NULL to be a marker rather than a value and SQL has certain special rules with regard to addressing NULLs in columns. NULLs also break many known mathematical identities on simple relational operations. In many cases, we have to use special functions like COALESCE and ISNULL to work around certain issues. Some have proposed the concept of "*typed NULLs*", but the baggage that comes with such ideas is too heavy.

This is somewhat of a controversy in the database circles, in fact anything that has to do with NULLs is a controversy among the experts. However the general consensus among the database professionals is that NULLs are a part of all DBMS products that use SQL; in fact, SQL cannot function without NULLs and the accompanying three valued logic (3VL). Therefore, most people tend to discount the implications of NULLs while making sure that a table is in 1NF. However, considering the logical implications of NULLs that we encounter in practice, in general, only minimal use of NULLs is recommended.

Why is 1NF important?

Often people talk about rows in a table with varying number of columns being a violation of 1NF. While this is true as a concept, this does not apply in SQL because SQL tables have all rows with the same set of columns. So why would this be an issue in defining 1NF? The reason is again, mostly because there is no suitable data type and partly due to a poor grasp of effective design principles, people will try to cram multiple values in to a single column, usually by

[Type text]



APTECH LIMITED

Education Support Services

Academics ACE –Head Office Mumbai

[Type text]

delimiting with a comma or a semi-colon. For instance, consider a table from a hypothetical financial firm that represents the account numbers their clients:

client_id	account_numbers
3256	54165452
2345	95184753, 68537142, 85693125
9514	26159483, 85632914
8524	62561981
3698	56321479, 864126208, 84095632
7532	36987412, 5698112

Here, the table does not accurately represent what the user wants. Since there is no specific "account number" data type available, the user is using a VARCHAR type to represent a account numbers. More specifically, since there is no available type to represent a "list of account numbers", the user has to use, again, a VARCHAR type.

Now what would be the practical implications of such a representation? Of course, it makes it easier to display, if the display requires the list of account numbers.

However, consider how one might construct a query that returns the clients who share the same account number. Or how about even a simple query: 'Which client has the account number 864126208?' Can it be done without parsing out the values in the account _numbers column? How about even a simpler query to see how many accounts does the client 9514 have?

Now it should not be hard for us to imagine the potential difficulties associated with adding, removing and changing account numbers.

There are further problems, as well, that relate to the integrity of the data: For instance, how do we avoid duplicating the account numbers for a particular client? How about enforcing a restriction on the number of account numbers a client can have? Obviously, in such multi-valued representations, all such constraints will have to be implemented outside the database without taking advantage of the existing integrity enforcement mechanism within the DBMS⁽⁵⁾.

So far, we haven't mentioned a fundamental flaw in representation that the same column has two kinds of values -- a single account number and a list of account numbers. In fact, because the appropriate constraints are missing, any VARCHAR value the user wants can be represented

[Type text]



APTECH LIMITED

Education Support Services

Academics ACE –Head Office Mumbai

[Type text]

in that column! In this case, if the business model deals with individual accounts, then each account number is a scalar value. Therefore having a column with multiple account number values can be deemed as a violation of 1NF.

How do we go about fixing this? One alternative often suggested is to "split" the list of account numbers into individual columns of VARCHAR type. It may result in a table, usually with column names conveying some additional information, as follows:

client_id	checking_acct_nbr	savings_acct_nbr	brokerage_acct_nbr
3256	54165452	NULL	NULL
2345	95184753	68537142	85693125
9514	26159483	85632914	NULL
8524	62561981	NULL	NULL
3698	56321479	864126208	84095632
7532	36987412	56981312	NULL

Obviously, we would have to use NULLs or some default values for columns that do not have a corresponding account number value. The fact that NULLs exist in the second table makes it a violation, but as said earlier, in SQL many people tend to discount NULLs from being a restriction. In that sense, this table seems to have met the other criteria for 1NF.

This representation is not as bad as the one before and helps with certain constraints that could not be applied before, like limiting the number of accounts. Moreover, some additional information is included, such as whether the account number belongs to a checking, savings or a brokerage account.

However, it still has some serious limitations with certain queries, especially if the same account number is used by more than one client. The duplication of account numbers is yet another problem. Furthermore, if a requirement arises to record more than three accounts for a client, additional schema alterations would be needed.

Ideally, this table can be represented better with three columns specifically identifying the account number and the account type.

[Type text]



APTECH LIMITED

Education Support Services

Academics ACE –Head Office Mumbai

[Type text]

client_id	account_number	account_type
3256	54165452	checking
2345	95184753	checking
2345	68537142	savings
2345	85693125	brokerage
9514	26159483	checking
2345	85632914	savings
8524	62561981	checking
3698	56321479	checking
2345	86426208	savings
2345	84095632	brokerage
7532	36987412	checking
7532	56981312	savings

Not only would this make the underlying predicates clearer, it allows for simpler queries. Though there are still some limitations from insufficient type support, there is little ambiguity as to whether the values are repeated across the columns. As an added benefit, there is no need for special handling of NULLs or default values and most constraints can be expressed directly. The uniqueness constraint will have to be declared by using a composite key.

Once the schema starts to develop, and requires additional reference constraints on this table, perhaps we could introduce a surrogate key, for reasons of simplicity, depending on the business requirements.

Conclusion

The effects of violating 1NF are sometimes considered harmless, though most of them compromise the structural soundness and integrity of the schema. You will do little more than impose added burden on the overall stability of a database if you try to "patch" up the problem by using complex routines that parse and pivot values or rely on external applications to enforce sufficient integrity. In a nutshell, any perception that you have achieved simplicity of design by keeping away 1NF is merely an illusion. On the other hand, there is much to gain by simply embracing it as the most foundational dictum of integrity in data management.