Invers Natural Number System to Maintain User-defined Sequence of Data Records

Seyfettin Öztürk
Nokia Solutions and Networks GmbH & Co. KG, Nürnberg, Germany

Keywords: Database Indexing, Storing User-defined Row Sequence, Avoiding Reordering, Increasing Performance, Decreasing Computing Time and Energy Consumption, Reducing Internet Data Traffic.

Abstract: The objective of this paper is to present a method to insert, edit, and delete database records without affecting the sequence of existing data. Typically, databases comprise integer data fields, in this paper named sequence number, meant to determine the user-defined sequence of data records. Inserting new data records or editing the sequence number of data records might cause a resequencing of the existing data records. This resequencing can be avoided by using a numbering system that decreases the value of a number when a digit is added to its end. Such a numbering system allows to insert an infinite quantity of additional sequence numbers between two sequence numbers even if their difference is 1.

1 INTRODUCTION

Generally, the user-defined sequence of data records in database applications, such as list elements, is stored in a dedicated data field named "sequence number", "list order", "position number" etc.

Inserting new data records or editing existing data records normally requires the renumbering respectively updating of subsequent data records.

The presented method in this paper allows user-defined sequencing without the need of resequencing.

2 THE ISSUE WITH THE USER-DEFINED SEQUENCES

The following use case is discussed to explain the issue with the user-defined sequence: An application like an internet browser processes a query for bookmarks and displays the resulting list to the user (Nelson, 2018). The user can easily rearrange the positions of the bookmarks according to his needs. The sequence of the bookmarks is determined by the values in the column "sequence number" of the corresponding data record in the database. Before user's rearrangement, it can be assumed that the sequence of the available bookmarks is as shown in table 1.

Table 1: Original sequence of bookmarks displayed to the user.

Bookmark	Sequence Number
d	1
g	2
b	3
f	4
a	5

The issue of user-defined sequences appears when the user rearranges the sequence in table 1 and moves, for example, the bookmark "f" from position 4 to 1. This leads to "write" activities for the data records "d", "g" and "b" to assign them the new sequence numbers 2, 3 and 4. Write operation are resource- and time-intensive, and software architects are interested very much in minimizing these factors.

2.1 State of the Art of Maintaining the User-defined Sequences

As outlined in the previous section, the sequence number values of the bookmarks "d", "g", and "b" must be updated when the user moves the bookmark "f" from position 4 to position 1, as shown in table 2.

Öztürk, S.
Invers Natural Number System to Maintain User-defined Sequence of Data Records.
DOI: 10.5220/0010571502600266
In *Proceedings of the 10th International Conference on Data Science, Technology and Applications (DATA 2021)*, pages 260-266
ISBN: 978-989-758-521-0

Table 2: Updated sequence of bookmarks displayed to the user.

Bookmark	Sequence Number
f	1
d	2
g	3
b	4
a	5

To implement the resequencing in table 2, the state of the art uses a database API (Application Programming Interface) or an SQL (Structured Query Language, Bayer, 1970) update statement. The following pseudocode is an example to illustrate the database writing activities using an SQL statement (Nelson, 2018):

```
Input:
Database table Bookmarks: bookmarks
Attribute Bookmark: bookmark
Attribute Sequence Number: s

SQL Update Statement:
Update bookmarks
 Set s = s + 1
 Where s >= 1 and s < (Select s
from bookmarks where bookmark = 'f')

Update bookmarks
 Set s = 1
 Where bookmark = 'f'
```

The SQL statement above will initiate and perform four write operations to rearrange the database table "bookmarks"; three operations will update the data records "d", "g" and "b" and one operation will set "f" to the sequence number 1.

The quantity of necessary write operations to update the sequence will depend on the position of the inserted new data record or the updated data record. The size of the required write operation in "Big O" notation is O(n) (Wikipedia, 2021). With other words, a given sequence with n records requires at worst "n" write operations to perform the rearrangement.

Within comprehensive databases, such rearrangements might cause numerous read and write actions, consuming considerable computing capacity. In the worst-case scenario, all existing data records in the sequence must be updated, as e.g. when inserting a new data record in the first position of the sequence.

The implementation of such data access transactions, in particular via the internet, is usually complex and time-consuming.

Using the innovative method presented in this paper, this comprehensive updating can be avoided since the insertion will not affect the remaining sequences anymore and the size of the write operations will be limited to O(1).

3 OPTIONS FOR USER-DEFINED SEQUENCES WITHOUT THE NEED OF RESEQUENCING

3.1 The Consideration of Fixed-point or Floating-point Arithmetic

In theory, it is possible to generate sequence numbers in fixed-point or floating-point format to avoid the resequencing (Schäfer, 1989). However, these data types have limited precision and after certain iteration the rightmost digits will be skipped, with the effect that the accuracy and resolution of results might be affected. This is due to the well-known machine "overflows" respectively "underflows" that can occur in electronic fixed- and floating arithmetic. Furthermore, electronic computers might have serious performance limitations when fixed-point and floating-point arithmetic is used (Schmid, 1974). Therefore, fixed- or floating-point numbers are rarely used in applications to maintain the user-defined sequences.

In contrast, the presented alternative method is based on integer data type that is interpreted as inverse natural numbers. The inverse natural number system keeps the original sequence of data records valid after new data record insertion and edition. The renumbering is avoided even if two sequence numbers only differ by 1 and a new data record is inserted in between.

3.2 The Inverse Natural Number System

When using the inverse natural numbering system, resequencing is avoided by interpreting the value of the sequence number inversely. In this context inverse means that the maximum value of the number is absolutely limited to the value of the leftmost digit. Consequently, the value of a sequence number decreases as the number of its digits increases. For example, if the leftmost digit of a number is 9, the value of this number will never exceed 9, no matter how many digits are appended at the right of the number.

With other words, the value of a sequence number can be reduced by adding further digits to the right of the existing digits of the number. Consequently, the sequence number 100 is interpreted as being smaller

than the sequence number 10. Accordingly, the sequence number 10 is assigned a smaller value than the sequence number 1.

The value of positive integer numbers, i.e. natural numbers, results from adding the place value multiplied with the digit, as shown below in the summation equation (1) (Sleator, 1988).

$$\begin{aligned} &\textstyle\sum_{i=0}^{n} a_i * 10^i \\ &= a_n\, a_{n-1} \dots a_1\, a_0 \quad (a \in \mathbb{N}, a_i \in \{0,1,\dots 9\}) \end{aligned} \tag{1}$$

This leads to the commonly known inequation (2):

$$10 < 100 \tag{2}$$

In contrast, the place value assignment of the inverse natural numbering system is determined using the following summation equation in (3):

$$\begin{aligned} &a_n * 10^0 + \sum_{i=0}^{n-1} -1 * \begin{cases} (9 - a_i) * 10^{i-n}, a_i < 9 \\ 9 * 10^{i-n-1}, a_i = 9 \end{cases} \\ &= a_n\, a_{n-1} \dots a_1\, a_0 \quad (a \in \mathbb{N}, a_i \in \{0,1,\dots 9\}) \end{aligned} \tag{3}$$

This leads to the inequation of the inverse natural numbers 10 and 100 (4):

$$10 > 100 \tag{4}$$

Due to this formula, it is possible to insert an infinite quantity of sequence numbers between two numbers that differ by 1. This is because the value of a sequence number decreases as the number of its digits increases.

4 THE COMPARATOR OF THE INVERSE NATURAL NUMBER SYSTEM

For the purpose of the inverse natural number system, the comparator can be used to determine the greater respectively the smaller of two given inverse natural numbers without applying the summation formula in (3). Hereinafter, the comparator's algorithm to compare two given inverse natural numbers will be presented.

The first inverse natural number's value is interpreted as being greater than the second number's value if:

a. all digits of the first number - from left to right - are identical to the corresponding digits of the second number and
b. the second number has at least one additional digit. In other words, the second number has more digits than the first number.

See table 3 for the comparison between 10 and 100.

Table 3: Example inverse natural numbers to compare.

Comparison 1:	a_n	a_{n-1}	a_{n-2}	a_{n-3}
First Number	1	0		
Second Number	1	0	0	
The digits a_n and a_{n-1} are identical, but second number has additionally the digit a_{n-2}, therefore: 10 > 100				
Comparison 2:	a_n	a_{n-1}	a_{n-2}	a_{n-3}
First Number	1	2	3	
Second Number	1	2	4	7
The digit a_{n-2} of the first number is smaller than the a_{n-2} of the second number, therefore: 123 < 1247				

The first two digits of 10 and 100 from left are identical in table 3. Since the integer number 100 has a third digit 0, it has, according to the new methods' comparator, less value than the integer number 10.

In summary, if all digits of the first number are processed and identical to the corresponding digits of the second number, and the second number still has further digits, then the second number is smaller. This is explained with the shown additional examples in table 4.

Table 4: Comparison of two inverse natural numbers P1 and P2 by the comparator.

P1	P2	Comparator
134	1350	P2 > P1
2783	2785	P2 > P1
221	22022	P1 > P2
66	667	P1 > P2
2595	25	P2 > P1
1330	133	P2 > P1

As presented in table 4, the algorithm enables to interpret the greater value between two of given inverse natural numbers just by comparing their digits from the left to the right.

5 MAINTAINING THE USER-DEFINED SEQUENCE USING THE INVERSE NATURAL NUMBER SYSTEM

This section briefly describes the insertion algorithm of a new section number:

- In the first position of a sequence
- Between two existing sequence numbers
- In the last position of the sequence.

The proposed insertion algorithm is also able to preserve the uniqueness of sequence numbers in the list order. This is even more relevant when the inverse natural number system is utilised for indexing and for key fields of the database records.

In order to maintain the uniqueness of sequence numbers, the method in this section avoids 9 as last rightmost digit of the newly inserted sequence numbers.

This is a key functionality to create index files that allow to insert new index items without the need to update the index tree (Bayer, 1970, Schäfer, 1989).

5.1 Insertion of Data Records in the First Position of the Sequence

This section outlines the methodology to add a new row in the beginning of the ordered list.

An additional data record is inserted in the first position of the sequence by calculating the sequence number P_n (n stands for new) of the additional data record as 10 times the sequence number P_1 of the subsequent data record in the sequence (see equation (5)):

$$P_n = 10 * P_1 \quad (5)$$

Here, the subsequent data record in the sequence is the data record that originally occupied the first place in the sequence of the data records.

This enables insertion of an additional data record in the first position of the sequence by reading just the sequence number of one further data record. The sequence numbers of the already existing data records do not need to be changed (see table 5).

Table 5: Inserting a row in the first position of the sequence.

Before insertion	After insertion
1 (P_1)	10 (P_n)
2	1 (P_1)
3	2
etc.	3
	etc.

5.2 Insertion of Data Records in the Sequence between a Preceding Data Record and a Succeeding One

To insert an additional data record in the sequence between a preceding data record and a succeeding one, the sequence numbers of the preceding and succeeding data record must be read.

In the event that the sequence numbers of the preceding data record and the succeeding data record differ by 1, the sequence number P_n of the additional data record is calculated as being 10 times the sequence number P_{i+1} of the succeeding data record (see equation (6) and table 6).

$$P_n = 10 * P_{i+1} \quad (6)$$

Table 6: Inserting a row between sequence numbers 1 and 2.

Before insertion	After insertion
1 (P_i)	1 (P_i)
2 (P_{i+1})	20 (P_n)
3	2 (P_{i+1})
etc.	3
	etc.

If the sequence numbers of the preceding data record and the succeeding data record differ by more than 1, a number of steps need to be performed in order to determine the sequence number of the additional data record.

Firstly, the sequence number P_n of the additional data record is calculated as being 1 plus the sequence number P_i of the preceding data record (see equation (7)):

$$P_n = P_i+1 \quad (7)$$

If the last digit of the preceding sequence number is equal to 8, then – in addition thereto – the obtained sequence number of the additional data record needs to be multiplied by 10 (see equation (8)).

$$P_n = 10 * P_n \quad (8)$$

This ensures that the last, i.e. the rightmost digit of the sequence number of the additional data record is not equal to 9 and the uniqueness of the sequence is guaranteed (see table 7).

Table 7: Inserting a row between sequence numbers 28 and 2.

Before insertion	After insertion
28 (P_i)	28 (P_i)
2 (P_{i+1})	290 (P_n)
3	2 (P_{i+1})
etc.	3
	etc.

Then, the sequence number of the additional data record is compared by means of the comparator (see Section 4) with the sequence number of the succeeding data record. The sequence number of the new data record must be smaller than the sequence number of the succeeding data record.

$$P_n < P_{i+1} \quad (9)$$

If the sequence number of the succeeding data record is not interpreted as being greater than the sequence number of the additional data record as in inequation (9), then the sequence number of the additional data record must be multiplied by 10 and compared with the sequence number of the succeeding data record again until the sequence number of the succeeding data record is interpreted as being greater than the resulting sequence number of the additional data record (see equation (10) and table 8).

$$P_n = 10 * P_n \quad (10)$$

Table 8: Inserting a row between sequence numbers 28 and 290.

Before insertion	Intermediate	After insertion
28 (P_i)	28 (P_i)	28 (P_i)
290 (Pi+1)	290 (P_n)	2900 (P_n)
3	290 (P_{i+1})	290 (P_{i+1})
etc.	3	3
	etc.	etc.

5.3 Insertion of Data Records in the Last Position of the Sequence

For insertion of additional data record in the last position of the sequence, it is necessary to distinguish whether the last, i.e. the rightmost, digit of the sequence number of the preceding data record is equal or not to 8.

In the event that the last digit of the sequence number of the preceding data record is not equal to 8, then the sequence number P_n of the additional data record is calculated as being 1 plus the sequence number Pi of the preceding data record (see equation (11) and table 9):

$$P_n = P_i+1 \quad (11)$$

Table 9: Inserting a data record at the end of the sequence.

Before insertion	After insertion
83	83
84	84
85 (P_i)	85 (P_i)
	86 (P_n)

If the last digit of the sequence number of the preceding data record is 8, then the sequence number P_n of the additional data record is determined in such a way that 1 is added to the sequence number of the preceding data record, and the sum obtained is multiplied by 10 (see equation (12)).

$$P_n = 10 * (P_i +1) \quad (12)$$

This is to ensure that the rightmost digit of the sequence number of the additional data record is not equal to 9, thus guaranteeing the uniqueness of the sequence.

The preceding data record here is the data record that occupied the last position in the sequence of data records originally, i.e. before the addition step (see table 10).

Table 10: Inserting a data record at the end of the sequence.

Before insertion	After insertion
6	6
7	7
8 (P_i)	8 (P_i)
	90 (P_n)

6 THE NEXT VALUE ALGORITHM FOR THE INVERSE NATURAL NUMBER SYSTEM

Software applications require a standard functionality that track the actual sequence and provide the last sequence number increment as the next value (IBM, 2021). The next value of integer numbers Pi is calculated as being 1 plus its selves (see equation (13)).

$$P_{nextValue} = 1 + P_i \quad (13)$$

As a matter of fact, the equation (14) will not result in the correct next value considering the summation equation (3) of the inverse natural number system. For instance, the next value of 1 applying equation (14) will result in 2. According to new method's comparator (see section 4), 2 as the next value of 1 would waste all numbers between 1 and 2 (see inequation (14) and table 11).

$$1 < 2000 < 2 \quad (14)$$

Table 11: Skipping sequence numbers between 1 and 2.

List Order	Unused sequence numbers
1 (P_i)	
	…
	2000
	…
	etc.
2 ($P_{nextValue}$)	

To ensure that the range of values of the new method are utilized effectively, it is proposed to start

the next value at a so-called turning number if the current sequence number is a single-digit number. The turning number is calculated by multiplying the current sequence number with a default power of 10. As an example, the possible valuation flow of the inverse natural numbering system using 3 as default power of 10 for the turning number is shown in table 12:

Table 12: Next value series of sequence numbers using the inverse natural number system.

Next value series of numbers using 3 as default power of the 10 for the turning number
1
2000, 2001, 2002, …, 2008,
200
2010, 2011, 2012, …, 2018,
201
2020, 2021, 2022, …, 2028,
…
2090, 2091, 2092, …, 2098
20
2100, 2101, 2102, …, 2108
…
2190, 2191, 2192, …, 2198
21
2200, 2201, 2102, …, 2208
…
2290, 2291, 2292, …, 2298
22
…
2990, 2991, 2992, …, 2998
2
3000, 3001, 3002, …, 3008
…

The pseudocode to calculate the next sequence number of the inverse natural number system is illustrated below:

```
Program Module to get the next value
for a given sequence number, written
in pseudocode

Input:
Given sequence number: v ∈ ℕ

Output:
Next sequence number value: x ∈ ℕ

Algorithm:
Initialize default power of 10 of the
most-left digit of the turning
number: p ∈ ℕ

Initialize power of 10 of the
most-left digit of the given
sequence number value v. This is
calculated using the logarithm base
10 to get the exponent and the
function Floor()to get the integer
part of a given decimal number:
q = Floor(Log10(v))

If rightmost digit of v is 8 and
 p smaller or equal q Then

 v ← (v - 8) / 10

   While rightmost digit of v is 9
        v ← (v - 9) / 10
   Loop

Else

   If p greater q Then
        v ← (v + 1) * 10^(p - q)

 Else
     v ← (v + 1)

Return x ← v
```

7 CONCLUSIONS

As a result, insertion of further data records in any position within the sequence of data records is performed effectively and easily when using the described method.

The new method allows management of databases without the need for recalculating already allocated sequence numbers of data records when inserting further data records in any position into the sequence of these records.

Using this method, a sequential renumbering of the sequence numbers of all succeeding data records in the sequence is avoided. It remains valid and relevant, even following changes of the sequence numbers or deletion of any data records.

Consequently, the computation time for managing a database is reduced and the data volume required when accessing the database, particularly via the internet, is minimized.

REFERENCES

Bayer R. and McCreight E. (1970). Organization and Maintenance of Ordered Indices, Boeing Scientific Research Laboratories.

Cowlishaw, M. F. (2021). Decimal Arithmetic FAQ. http://speleotrove.com/decimal/decifaq.html.

Cowlishaw, M. F. (2003). Decimal Floating-Point: Algorism for Computers. IBM UK.
IBM (2021). db2 product hub. https://www.ibm.com/support/producthub/db2/docs/content/SSEPGG_11.5.0/com.ibm.db2.luw.sql.ref.doc/doc/r0023464.html.
IBM (2021). Sequence objects. https://www.ibm.com/docs/en/db2-for-zos/11?topic=programs-sequence-objects.
Microsoft (2021). ALTER SEQUENCE (Transact-SQL). https://docs.microsoft.com/de-de/sql/t-sql/statements/alter-sequence-transact-sql?view=sql-server-ver15.
Nelson, J (2018). User-defined Order in SQL. https://begriffs.com/posts/2018-03-20-user-defined-order.html.
Sleator, D. D. and Dietz, P. F. (1988). Two Algorithms for Maintaining Order in a List.
Schäfer, G. (1989). Datenstrukturen und Datenbanken.
Schmid, H. (1974). Decimal Computation, General Electric Company, New York.
Horn, T. (2007). SQL-Grundlagen. https://www.torsten-horn.de/techdocs/sql.htm.
Wikipedia (2021). Decimal. https://en.wikipedia.org/wiki/Decimal.
Wikipedia (2021). Big O Notation. https://en.wikipedia.org/wiki/Big_O_notation.