RESEARCH REPORT
OCCASIONAL PAPER

# Verifying Data Migration Correctness: The Checksum Principle

Bin Wei and Tennyson X. Chen

March 2014

RTI Press

RTI INTERNATIONAL

**About the Authors**

**Bin Wei**, MS, is a senior researcher at the Pacific Islands Fisheries Science Center, a National Oceanic and Atmospheric Administration (NOAA) research branch within the Joint Institute for Marine and Atmospheric Research, a NOAA Cooperative Institute at the University of Hawaii at Manoa.* His areas of expertise include designing systems for remote data collection and data transformation.

**Tennyson X. Chen**, MS, is a senior research analyst and software system architect in RTI International's Research Computing Division. His main focus is the National Survey of Drug Use and Health (NSDUH) project, for which he is a key system designer and database manager.

*Although Mr. Wei is employed by the NOAA Joint Institute for Marine and Atmospheric Research, this paper was produced independently and is not a work product of NOAA.

RTI Press publication OP-0019-1403

## Suggested Citation

Wei, B. and Chen, T. X. (2014). *Verifying data migration correctness: The checksum principle* (RTI Press publication OP-0019-1403). Research Triangle Park, NC: RTI Press.

This publication is part of the RTI Research Report series. Occasional Papers are scholarly essays on policy, methods, or other topics relevant to RTI areas of research or technical focus.

RTI International
3040 East Cornwallis Road
PO Box 12194
Research Triangle Park, NC
27709-2194 USA

Tel: +1.919.541.6000
Fax: +1.919.541.5985
E-mail: rtipress@rti.org
Website: www.rti.org

**doi:10.3768/rtipress.2014.op.0019.1403** **www.rti.org/rtipress**

# Verifying Data Migration Correctness: The Checksum Principle

Bin Wei and Tennyson X. Chen

## Abstract

Data migration, generally referred to as the process of reading data from their source and inserting them into a target database, is an important element of data extract, transform, and load (ETL) systems. During data migration, errors can occur during data transmission. These errors can directly affect the quality of the data in the target database. Therefore, verifying the correctness of the outcome is a critical component of a data migration operation. Current methods in data migration correctness verification have many limitations, including incompleteness and inaccuracy. This paper describes our innovative method that applies the well-proven checksum methodology to verify the correctness of the data migration outcome. This method performs a thorough and accurate verification on the correctness of the migrated data, mitigating most of the weaknesses associated with current verification methods. This method is also easy to implement and will greatly enhance the quality of data migration operations.

## Contents

RTI INTERNATIONAL

## Introduction

Many organizations, including service providers and research organizations, rely on large amounts of data to manage their operations. Moving data is a common and necessary operation in today's information systems. As a result, designing and implementing data extract, transform, and load (ETL) systems is a hot topic for programmers, developers, and the computing community.

According to Kimball and Caserta, "A properly designed ETL system extracts data from the source systems, enforces data quality and consistency standards, conforms data so that separate sources can be used together, and . . . delivers data in a presentation-ready format."[1[pxxi]] ETL systems are commonly used to transfer data from one database to another, to form data marts and data warehouses, and to convert fields in databases from one format or type to another. An important function of an ETL system is the process of data migration.

*Data migration* can be defined many ways. According to Johny Morris,[2[pxv]] data migration is "the selection, preparation, extraction, transformation, and permanent movement of appropriate data that is of the right quality to the right place at the right time and the decommissioning of legacy data stores." Data migration is also known as "the one-time movement of data from a legacy source, or multiple sources, to a new target database."[3[p4]] Although these definitions vary, data migration is generally defined as reading data from a source database and inserting them into a target database.

A number of factors can cause errors during data migration. Leaving these errors undetected during the data migration process can cause data to not be correctly transferred to the destination.

### Errors in Data Migration

Commonly, errors in data migration are caused by faulty hardware, network disturbances, format mismatches, and database definition mismatches. For example, a corrupted hard disk or computer memory can prevent the system from reading and transmitting the source table content correctly. If the system receives incorrect data from the source table, the data that are inserted into the target table cannot be correct.

Network transmission noise, interference, or distortion can also prevent an accurate migration. Data are migrated increasingly via the Internet, which can introduce errors to the data content. For example, in Table 1, an error during the transmission of the second record might change the City value from Amazon to Amazen. Although Internet protocols can detect and correct most transmission errors, some erroneous data can still be transmitted without being detected.[4]

Format mismatch between the source and target databases is the most common error that we have observed in designing data migration tools and managing large-scale data migration projects.[5] In Table 1, the definition of the Address field is "string" with a width of 35 characters. If the table in the target database defines the same field with a width of only 20 characters, the Address information of

**Table 1. Employees source table to be migrated**

| ID | Name | Address | City | State | Zip | Birthday | Height | Weight |
|---|---|---|---|---|---|---|---|---|
| 1 | John | 1321 IPhone Road | Apple | XX | 00011 | 1/7/1991 | 5.82 | 169 |
| 2 | David | 1322 Kindle street | Amazon | YY | 00012 | 2/7/1992 | 6.18 | 193 |
| 3 | Matthew | 1323 IPad Road | Apple | XX | 00013 | 3/7/1993 | 5.73 | 160 |
| 4 | Mark | 1324 Galaxy Lane | Samsung | ZZ | 00014 | 4/7/1992 | 6.10 | 210 |
| 5 | Luke | 1325 Pink Drive | Blue City | YY | 00015 | 5/7/1993 | 5.69 | 145 |
| 6 | Skip | 1326 Humuhumunukunukuapuaa Avenue | Fish City | NN | 00016 | 6/7/2994 | 5.32 | 136 |

the sixth record would be truncated to be "1326 Humuhumunukunuk" in certain database platforms and result in the loss of address information. Likewise, the definition of the Height field is decimal with 2-digit precision in the source table. If the height field is erroneously set up as an integer field or a decimal with 1-digit precision, the target table will round all numbers received in this field up or down, making the numbers unusable. In both scenarios, data lost during data migration result in inaccurate and unusable data.

In addition, errors are caused by different definitions between the source and target databases in primary key or constraint settings. This difference usually causes some records in the source table to be rejected from the target table. For example, before migrating data from Table 1, if the target table has ID field as its primary key and the table already has a record with value 5 in this field, trying to insert record number 5 of Table 1 into the target table would cause a key violation, and the record would not be migrated correctly.

Because of these potential errors, verifying the correctness of the data is an important component of the data migration process. Next, we discuss a few popular methods that are currently used to verify whether data have been migrated correctly and some of the limitations of each.

## Current Methods for Detecting Data Migration Errors

Once the migration is complete, most data migration systems provide only a simple report stating the number of records that have been migrated. Although this record count provides an important clue as to any records that were rejected during the migration, it does not provide any information on any problems or ensure that the data have been transferred correctly. Further verification of the data content is needed.

The first and most obvious way to verify data correctness is by manual inspection and ad hoc queries. Programmers can examine the data visually to verify that the content in the target database matches that of the source database. Programmers can also run ad hoc queries to check the quality of the transfer. While these methods work well for a small amount of data, using them in large-scale data migration operations is prohibitively expensive in terms of time and effort.

Programmers can also perform *post-migration testing*[6] or *validation testing*.[7,8] For these tests, programmers implement a series of predefined queries that examine the content of the source and target databases. The result is a summary report that provides discrepancy information once the data migration operation is complete. In a previous work,[5[p7]] we illustrated one example of validation testing as follows:

- For each field with "numeric" data type, the validation test compares maximum, minimum, average, and summary (indicating) values between the two databases. For example, before migrating the data from Table 1, we can calculate these indicating values from the Weight field in the source table. After the migration, we can perform the same calculation from the Weight field of the target table. If the results of these two calculations do not match, errors have occurred during migration.
- For each field with "string" data type, the validation test compares string length between the two databases. For example, before migration, we can make note of the total string length of the Address field of all records and compare it with the same measure after the migration. If the numbers do not match, migration errors have occurred.
- For each field with "date/time" data type, the validation converts the values into a numeric representation and compares the databases using the method applied on "numeric" fields.

In any of these comparisons, if the values in the target database do not match those of the source database, errors must have occurred during the migration process. While this method provides important information on the successfulness of the migration, it does not check the data at the record level. If programmers detect errors, they still do not know which particular records have been migrated incorrectly. Therefore, they may have to repeat the entire migration process. Even so, repeating the process does not guarantee success because the factors that caused error in the first migration may come into play during the second, or other factors may cause additional problems.

*Reconciliation testing* examines the data correctness at the record level.[9-11] This testing method retrieves and compares the content of each record in the source and target databases, making it the most comprehensive way to verify whether data were migrated correctly. Because this method performs correction verification at the record level, it will normally pinpoint the exact records that are in error. This is an improvement over the validation testing method mentioned above, but reconciliation testing has its own drawbacks:

- Depending on the data volume, reconciliation testing may be very time-consuming.
- If the source database is located remotely, this test may become difficult to perform and the testing itself may be error-prone.
- If the primary key definitions are inconsistent between the source and target tables, or if the source table does not have a primary key, matching source and target records exactly would be impossible.

Because of the drawbacks of the methods discussed so far, we have implemented a new data migration correctness verification method using the checksum methodology. This method verifies data content at the record level, so it is more comprehensive than validation testing. It also does not require the system to retrieve the source data again, so it avoids the weaknesses of reconciliation testing. Our method is novel in that it integrates the well-known checksum methodology used in network data transmission error detection into data migration correctness verification. Before we provide additional detail on how to implement this method, we first briefly review the checksum methodology and how it is used in network data transmissions.

## Checksum Methodology in Network Transmission Error Detection

Parity bit check and its extension called checksum are popular methods of detecting network data transmission errors. There are variations on how to implement a checksum. Bolton[12] describes this methodology as follows:

> The movement of digital data from one location to another can result in transmission errors. For example, the transmitted sequence 1001 may be incorrectly received as 1101. In order to detect such errors a *parity bit* is often used. A parity bit is an extra 0 or 1 bit attached to a code group at transmission. In the *even parity* method the value of the bit is chosen so that the total number of 1s in the code group, including the parity bit, is an even number. For example, in transmitting 1001 the parity bit used would be 0 to give 01001, and thus an even number of 1s. In transmitting 1101 the parity bit used would be 1 to give 11101, and thus an even number of 1s. With *odd parity* the parity bit is chosen so that the total number of 1s, including the parity bit, is odd. Thus if at the receiver the number of 1s in a code group does not give the required parity, the receiver will know that there is an error and can request that the code group be retransmitted.
>
> An extension of the parity check is the *checksum* in which a block of code may be checked by sending a series of bits representing their binary sum.

For example, for each binary packet sent, an 8-bit checksum can be attached. The checksum bits can hold $2^8$ different values, so the checksum's value ranges from 0 to 255. If the value of the packet is less than 255, the checksum is identical to the packet value; otherwise, the checksum value is the remainder of the packet value divided by 256.

In summary, checksum methodology basically derives extra content from the original information, attaches this content to the original information, and transmits the combined information to the receiving end. Once the information is received, the receiver re-derives the checksum content and compares it with the information received. If the checksum values have discrepancies, errors have occurred. In the next section, we discuss how to apply the checksum methodology to verify whether data have migrated correctly.

## A New Data Migration Correctness Verification Method That Applies the Checksum Methodology

We first discuss the implementation details of this correctness verification method. Then we discuss its strengths and weaknesses.

### Implementation of the Method

The checksum methodology uses extra content to verify the correctness of the original data. This methodology is the basic mechanism of the data migration correctness verification method discussed in this paper. In this method, additional content is attached to each record to verify the correctness of the record after migration. Therefore, the first step is to add a new checksum field to each table that is to be migrated from one database to another. This field holds the extra content for each record that is going to be used to verify the record. The second step is to decide what content to put into the checksum field.

A common and reliable method for creating content for the checksum field that uses fixed-length and shortened output to reference the original data is *hash function*. A hash function can generate a code for each record, based on the content of the record, for the checksum field of the source table. Once the table arrives at the receiving end, programmers apply the same hash function algorithm that was used to encode the source to generate a *hash code* for each record received. For any records received, if the calculated hash codes on the receiving end do not match the hash codes transferred with the record, errors have occurred in these particular records during migration.

There are several mature and well-known hash code algorithms, such as MD5 and SHA-1.[13] For our example, we use the popular MD5 algorithm to generate the hash codes. To illustrate this process, we used the Employees information from Table 1. Table 2 shows the source table to be migrated, with the new field containing the checksum hash codes calculated for each record.

To generate the checksum hash codes in the source table, we retrieve the content of each record, convert the value in every field into alpha character type, concatenate these values into a single string, and then pass the string as a parameter (inputString) into a programming code module (written in C# in this example) as follows:

```
private string GetHashString(inputString)
{
  //calculate MD5 hash from input
  System.Security.Cryptography.MD5 md5;
  md5 = System.Security.Cryptography.MD5.
      Create();
  string contentstr = inputString;
  if (contentstr == null) return null;
  byte[] inputBytes = System.Text.Encoding.
      ASCII.GetBytes(contentstr);
  byte[] hash = md5.ComputeHash(inputBytes);

  //convert byte array to hex string
  StringBuilder sb = new StringBuilder();
  for (int i = 0; i < hash.Length; i++) {
      sb.Append(hash[i].ToString(“x2”));
  }
  return sb.ToString();
}
```

Implementing this hash code method is simple and straightforward. This same module is used on the receiving end to calculate the checksum hash codes in the target table. This method is very effective in detecting most data migration errors, as discussed next.

**Table 2. Employees source table to be migrated using the checksum verification method**

| ID | Name | Address | … | Checksum Hash Code |
|---|---|---|---|---|
| 1 | John | 1321 IPhone Road | … | 652adc7cc073fe69b94a48b3704e9f12 |
| 2 | David | 1322 Kindle street | … | 745f6845ff3ebf33ab618aae567f3926 |
| 3 | Matthew | 1323 IPad Road | … | 3d88779f639bb5e2bc32441009e7bb00 |
| 4 | Mark | 1324 Galaxy Lane | … | a88798f5a9025645020026f11c35c93f |
| 5 | Luke | 1325 Pink Drive | … | 131ac65ce64cacbd3168afed2a504a30 |
| 6 | Skip | 1326 Humuhumunukunukuapuaa Avenue | … | 17c126a5e3c06f20a8f36a3f1703778c |

## Discussion

We previously listed four main causes of potential errors: faulty hardware, network transmission noise, database format mismatch between source and target databases, and database primary key and constraint definition differences in source and target databases. The checksum verification method proposed in this paper can be used effectively in detecting the first three types of errors, particularly the errors caused by database format mismatch between source and target databases, the most common in data migration. We use this error type to illustrate the effectiveness of checksum verification.

In Table 3, the scenario is that the target table defines the Address field with a width of 20 characters rather than 35.

Because of this 20-character definition, the Address information in record 6 is truncated due to the target table's field format. When we calculate the hash code for each record in the target table, the hash code of record 6 would be 9448b731deacdf2942c85bc8d5a0af66, which is different from the checksum hash code received. Therefore, this comparison reveals that a data migration error occurred and also pinpoints the exact record that was migrated erroneously.

In the scenario in which the Height field (from Table 1) is defined with 1-digit decimal precision in the target table, Table 4 illustrates the comparison between the received checksum values and the calculated hash codes based on the target table content.

Other than the fourth record, which has value 6.10 in the Height field of the source table (and is therefore not affected by the new format definition of the target table), the calculated hash codes are different from those that were received. Table 4 reveals that data migration error has occurred in almost every record.

Based on the above analysis on detecting data migration errors, one can easily infer that this method would be useful in identifying errors caused by faulty hardware as well as by network transmission noise. This method checks the calculated hash code of each record and therefore provides a comprehensive examination of the data migration outcome. In addition, this method performs correctness verification solely on the target database and does not require the repeated retrieval of the source data, which makes it more feasible and easier to implement than validation testing or reconciliation testing.

While checksum verification is effective and can be applied to detect most of the data migration errors,

**Table 3. Employees target table received with error**

| ID | Name | Address | … | Checksum Hash Code Received |
|---|---|---|---|---|
| 1 | John | 1321 IPhone Road | … | 652adc7cc073fe69b94a48b3704e9f12 |
| 2 | David | 1322 Kindle street | … | 745f6845ff3ebf33ab618aae567f3926 |
| 3 | Matthew | 1323 IPad Road | … | 3d88779f639bb5e2bc32441009e7bb00 |
| 4 | Mark | 1324 Galaxy Lane | … | a88798f5a9025645020026f11c35c93f |
| 5 | Luke | 1325 Pink Drive | … | 131ac65ce64cacbd3168afed2a504a30 |
| 6 | Skip | **1326 Humuhumunukunuk** | … | 17c126a5e3c06f20a8f36a3f1703778c |

**Table 4. Comparison of received and calculated hash codes**

| ID | Checksum Hash Code Received | Hash Code Calculated After Receipt |
|---|---|---|
| 1 | 652adc7cc073fe69b94a48b3704e9f12 | dcc120f003d4c49f6f94bb0fb6810df3 |
| 2 | 745f6845ff3ebf33ab618aae567f3926 | 48e83caff5f7a3805c4abd68855e523e |
| 3 | 3d88779f639bb5e2bc32441009e7bb00 | 58168cfa64cbbeee27c0b0a82d0b0a8d |
| 4 | a88798f5a9025645020026f11c35c93f | a88798f5a9025645020026f11c35c93f |
| 5 | 131ac65ce64cacbd3168afed2a504a30 | ddc375d542d35eca0a4dc15e5c037cc9 |
| 6 | 17c126a5e3c06f20a8f36a3f1703778c | 3f2ce06d09d7bf5835041c1eece0a0b3 |

programmers do need to be mindful of its limitations and drawbacks. First, checksum verification increases the data volume being transferred because of the extra checksum field in each table. However, with improving technologies in data communications and with storage becoming less expensive, this concern becomes less of an issue almost daily.

Second, if data migration errors occur in transmitting the checksum values but not in transmitting the original record content, the checksum method may actually cause false alarms.

Third, the checksum method can only detect errors on the records that are received in the target table. If records are rejected because of differences in primary key and constraint definition between the source and target tables, the checksum method is not effective. Fortunately, the error caused by records being rejected is easy to detect. A simple record-count comparison before and after the migration, which almost all data migration tools perform, is sufficient to detect this sort of error, overcoming the method's biggest shortcoming.

Fourth, the checksum method described here can leave errors undetected, just like the parity bit and checksum methods in network data transmission can. This is because converting the content of a record to a hash code is not a 1-to-1 relationship. Although the possibility is extremely low, errors could alter the record content while the resulting hash code remains identical to that of the original record. In such a scenario, the errors will escape the detection of the checksum method.

However, these potential problems should rarely occur because our proposed method is based on the well-tested and commonly used checksum methodology. The limitations are merely the inheritance of this methodology. Given the success of the checksum methodology, and the multiple advantages of the checksum data migration verification method as illustrated in our examples, we believe this method provides an easy and effective way to verify the correctness of a data migration operation. Implementing the checksum correctness verification method in transmitting data from one database to another can greatly enhance the quality of a data migration outcome.

## Future Work

Much work has been done on not only detecting errors with the parity bit and checksum method in network data transmission, but correcting those errors as well. Further research and exploration is needed to expand the checksum correctness verification method described here to detect and correct errors more effectively and overcome the slight possibility that some errors pass undetected. Further research in studying the error-missing rate will be helpful in establishing the tolerance level when applying this method.

## References

1. Kimball R, Caserta J. The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data. Indianapolis (IN): Wiley Publishing; 2004.
2. Morris J. Practical data migration. 2nd ed. Swindon (UK): BCS Learning & Development, Ltd.; 2012.
3. A roadmap to data migration success: approaching the unique issues of data migration [Internet] San Jose (CA): SAP Business Objects; 2008. Available from: http://fm.sap.com/data/upload/files/A_Road_Map_to_Data_Migration_Success_2010.3.17-17.29.55.pdf
4. White C. Data communications and computer networks: a business user's approach. Boston: Course Technology; 2011.
5. Wei B, Chen TX. Criteria for evaluating general database migration tools (RTI Press publication No. OP-0009-1210). Research Triangle Park (NC): RTI Press; 2012. Available from: http://www.rti.org/publications/rtipress.cfm?pubid=20204
6. Katzoff D. How to implement an effective data migration testing strategy [Internet]. Data Migration Pro; 2009. Available from: http://www.datamigrationpro.com/data-migration-articles/2009/11/30/how-to-implement-an-effective-data-migration-testing-strateg.html
7. Singh I. How to do database migration testing effectively and quickly? [Software testing blog on the Internet]. Delhi: Inder P. Singh; 2010. Available from: http://inderpsingh.blogspot.com/2010/03/how-to-do-database-migration-testing.html
8. Paygude P, Devale P. Automated data validation testing tool for data migration quality assurance. Int J Mod Eng Res (IJMER). 2013 Jan-Feb;3(1) 599-603.
9. Matthes F, Schulz C, Haller K. Testing & quality assurance in data migration projects. In proceeding of: IEEE 27th International Conference on Software Maintenance, ICSM 2011; Williamsburg (VA), September 25-30, 2011.
10. Haller K. Towards the industrialization of data migration: concepts and patterns for standard software implementation projects. In: Eck P, Gordijn J, Wieringa R, editors. 21st International Conference on Advanced Information Systems Engineering (CAiSE) Proceedings; 2009; Amsterdam, The Netherlands; June 8–12, 2009; pp. 63-78.
11. Manjunath T, Hegadi R, Mohan H. Automated data validation for data migration security. Int J Computer App. 2011 Sep;30(6):41-6.
12. BoltonW. Mechatronics: electronic control systems in mechanical and electrical engineering. 3rd ed. Longman (NY): Prentice Hall; 2004.
13. Mironov I. Hash functions: theory, attacks, and applications. Mountain View (CA): Microsoft Research; 2005.

## Acknowledgments

We would like to thank Craig Hollingsworth and Joanne Studders of RTI International for making substantial editorial contributions to this paper. We would also like to thank our colleagues at NOAA and RTI's Research Computing Division for helping us shape and improve the idea of the method discussed in this paper.

**www.rti.org/rtipress** **RTI Press publication OP-0019-1403**