**EduMaster**

**Report Manager Data Marts**

**Test Results for Attempt Star**

**Version 1.0**

**Jay Dean**

**LEX Software Systems**

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Revision Date | Revision Description |
| Draft 0.5 | Jay Dean | August 3 | Reviewed in phone conference with project leaders |
| 0.8 | Jay Dean | August 9 | Response to mgmt review |
| 0.9 | Jay Dean | Sept 8 | Reformatted  |
| 1.0  | Jay Dean | Sept 10 | Initial Distribution |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Introduction 5

A. Overview of the Software Under Test 5

B. References 5

C. Assumptions and scope 5

II. Summary of Results 5

A. Attempt Star Overview 5

B. Stored Procedures: 5

C. Fact Table: 6

D. Dimension Tables: 6

E. Null Values 6

F. Duplicate Descriptors 7

G. Inconsistent Use of “Status” or “Flag” dimensions 7

H. Outlier Values in Dimension Data 7

III. Detailed Results – Attempt Star 8

A. Test Series 1 – Stored Procedure Code Review 8

B. Test Series 2 – Stored Procedure Code Results 9

C. Test Series 3 – Attempt Star Fact Table 9

Test 1 – Identical “Grain” 9

Test 2 – Row Count Comparison 10

Test 3 – Foreign Keys 12

Test 4 – Matching values within records 16

Test 5 – Key Fields 100% Populated 20

Test 6 – Profile presence of nulls, zero-length strings and zero’s in fact fields 22

Test 7 – Profile presence of -1 values in Key fields 24

D. Test Series 4 – dimUser Dimension Table 27

Test 1 – Row Count Comparison 27

Test 2 – Profile presence of null, “Missing”, and other troublesome values 30

Test 3 – Profile composition of data fields 33

Test 4 – Verify fields populated from linked tables or lookup in LMS 33

Test 5 – Check for truncated string values 34

Test 6 – Verify Unique Descriptors 37

E. Test Series 5 – dimActivity Dimension Table 39

Test 1 – Row Count Comparison 39

Test 2 – Profile presence of null, “Missing”, and other troublesome values 41

Test 3 – Profile composition of data fields 46

Test 5 – Check for truncated string values 50

Test 6 – Verify Unique Descriptors 53

F. Test Series 6 – dimDate Dimension Table 55

Test 1 – Verify Unique Descriptors 55

Test 2 – Check for duplicate or orphaned keys 56

Test 3 – Test Logical Value 57

G. Test Series 7 – dimCompletionStatus Dimension Table 59

Test 1 – Verify Unique Descriptors 59

Test 2 – Check for duplicate or orphaned keys 60

Test 3 – Test Logical Value 61

H. Test Series 8 – dimSatisfied Dimension Table 62

Test 1 – Verify Unique Descriptors 62

Test 2 – Check for duplicate or orphaned keys 62

I. Test Series 9 – dimSuccess Dimension Table 63

Test 1 – Verify Unique Descriptors 63

Test 2 – Check for duplicate or orphaned keys 64

Test 3 – Test Logical Value 65

J. Test Series 10 – dimRegistrationStatus Dimension Table 66

Test 1 – Verify Unique Descriptors 66

Test 2 – Check for duplicate or orphaned keys 66

Test 2 – Check for duplicate or orphaned keys 67

K. Test Series 11 – vwdimStartDate Dimension Table 69

Test 1 – Verify View Definition 69

L. Test Series 12 – vwdimEndDate Dimension Table 69

Test 1 – Verify View Definition 69

M. Test Series 13 – vwdimExpirationDate Dimension Table 70

Test 1 – Verify View Definition 70

N. Test Series 14 – Star Structure 71

Test 1 – Query fact table fully joined 71

O. Test Series 15 – ETL Procedure 72

Test 1 – Verify View Definition 72

P. Test Series 16 – dimGrade 74

Test 1 – Verify Unique Descriptors 74

Test 2 – Check for duplicate or orphaned keys 76

# Introduction

## Overview of the Software Under Test

The EduMaster Report Manager utilizes a framework of stored procedures to derive data for reports. These stored procedures are structured layers, with “upper-level procedures” calling lower in the layering hierarchy.

All the reporting procedures are ultimately directed at one of two star-schema data marts, which are themselves regularly updated with data from the EduMaster LMS database.

## References

The following documents were used during this testing:

| **Documents** | **Version** |
| --- | --- |
| Report Manager Stored Procedure Framework | 0.5 |
| ST700\_FS\_ReportManager\_ReportFilter.doc | 1.1 |
| Reporting with EduMaster 7.0 | 05 |
| EduMaster 7.1 Data Dictionary |  |
| Report Manager Test Strategy and Plan – Star Schemas | 0.7 |

## Assumptions and scope

These tests are designed to discover bugs or potential errors derived from this one element of the reporting infrastructure. Apparent errors and inconsistencies will be cataloged for the attention of the Development Team.

# Summary of Results

## Attempt Star Overview

The Attempt Data Mart appears to be an accurate representation of the LMS database that underlies it. The procedures that load and query this star-schema data mart do so without altering, abridging, or adding to the data. The overall structure of the data mart is sound, and appropriate for the task.

## Stored Procedures:

There is one primary procedure that queries the Attempt star, lmssp\_GetEmpActRegistration. Much of the procedure is concerned with security and report filtering, which are being tested in a separate task, or with performance optimizations. The actual data query within the procedure is repeated four times, within minor variation based on whether User and/or Activity filtering is being applied. There is one unexpected (or unexplained) variation between these separate instances of the query.

When *both* Activity and User filtering is applied, a 31st field is returned by the procedure, that *is not returned* in other situations. The field is **attnd.name** and is labeled, “Attendance Status.” This is the 11th of 31 fields returned in that one case.

This sort of inconsistency can be very difficult to handle when designing reports. The report designer and developer are challenged to predict the number of fields returned. Even if a work-around is found, this inconsistent result set can readily lead to errors and bugs.

## Fact Table:

The central table in the star schema, factAttempt, accurately reflects the data in the underlying LMS table, TBL\_TMX\_Attempt, and returns appropriate results when queried.

All key fields in the fact table are populated, and all values are properly found in the dimTables (no orphans). The table is close to being a factless fact table, with only one field that could be considered a true data field, “Score.” This field is sparsely populated.

We did note that a dimension foreign key value of “-1”, representing a null value in the source table, appears frequently for the minor dimensions. This is handled appropriately, and each dim table provides a matching row. This is also an accurate reflection of the source table data. However, this ambiguous data context.will make it difficult to consistently interpret reports.

## Dimension Tables:

This data mart contains both dynamic and static dimension tables. In general, we strongly prefer dynamically built dimension tables, when a source table is available. In this case, the LMS table does not store many of the dimension values in database tables, so the use of static tables may be unavoidable.

All dimension tables are properly structured and appear to accurately represent the data in the LMS system. The dynamic tables are very slightly filtered when created, and any rows dropped at that time do not link to records in the fact table. We found only a handful of possible issues, all of which derive from the state of the LMS data.

## Null Values

Many of the non-key fields in the dimension tables contain null values; often these fields nearly completely unpopulated. This includes some fields that are queried in important stored procedures. This is an accurate representation of the data, the fields are not populated in the LMS database, but this can cause confusion in reports.

A report designed around a field which is very sparsely populated— a report on activity by Employee State, for example—will profile a very small sample of the full activity for these customers. This may not be apparent to the report consumer.

Given the nature of the LMS data source, this issue is largely unavoidable—a reporting system cannot report what is not there—but it ought to make it clear to the user that the data they seek “is not there.”

## Duplicate Descriptors

The components used in the reporting web-client application will merge records with common descriptor fields. In order to keep separate records separated in the web-client, each record needs a unique descriptor field that can be used in the report.

Both the User and Activity dimensions present challenges. The name fields for Users do not provide a guaranteed unique descriptor. The stored procedure that queries this star creates a new field by concatenating to other fields, and this does appear to create a unique descriptor.

None of the fields queried from the Activity table, on the other hand, generate unique descriptors. Reports showing multiple activities on the report rows may inadvertently combine different Activities.

## Inconsistent Use of “Status” or “Flag” dimensions

Several of the dimensions are simple boolean flags representing a status indication; “success”, for example. We observed inconsistency among the three customer databases in how these fields are being populated. Again, the data in the star is an accurate representation of the LMS tables, and the issue concerns interpreting reports, not the accurate function of the system.

Even an apparently “binary” choice will result in *three* possible results. “yes”, “no”, and “null” or “no answer”. For the “Completion Status” dimension (test series 7.02), for example, a value of “Complete” is reasonably unambiguous, but one database contains no records marked “Not Completed”, the other records being null, or “-1”. Elsewhere, some records are explicitly “Not Completed” and others null, and at the third database there are no nulls. This again is an accurate picture of the underlying data, but it can make designing an accurate report a challenge.

## Outlier Values in Dimension Data

The Activity dimension table contains a few “data fields”, recording the duration of the activity. In general, the data in these fields, where present, looks reasonable, but there are a few outlier records which are clearly inaccurate, and which could impact analysis based on this field. In the ST0\_DA PreExtract database, for example, one Activity is listed with duration of over 22,000 hours; another is listed with duration of -72 hours.

As above, these values are present in the underlying LMS tables, and are not artifacts of the reporting system. We mention it only as a caution. Such values should be cleaned from the data or filtered out of reports.

# Detailed Results – Attempt Star

##  Test Series 1 – Stored Procedure Code Review

The Attempt Star Schema is queried by a single, general-purpose procedure, lmssp\_GetEmpActRegistration. This procedure is constructed in sections, with program flow controlled by nested “If…Else” structures. The outer block separates flow based on whether the query will return result records or just a record count.. The inner block of If…Else statements sends program execution through one of four different queries, depending on whether Activity Filtering, User Filtering, Both or Neither is intended.

This sort of structure can work well and is relatively easy to create and review. Maintenance of these structures, however, can be challenging. The four separate queries within the procedure (excluding the rowcount-only query) should provide identical results, except for the intended differences of filtering. If edits and fixes to one of the four separate code blocks are not properly reflected in *all* the blocks, unintended differences will result.

The SELECT Statements within the four queries should be identical—exactly the same text—so that the result set from the query is consistent, and the code easier to maintain. We found, however, that one of the four SELECT statements contained an extra line. When both Activity and User filtering are chosen, the procedure returns the field “attnd.name”, under the name “Attendance Status”, as the 11th field in the result. In the other three situations, Activity Only, User Only, and No Filtering, this field is not returned. When both filter sets are applied, the procedure returns 31 fields of data. In the other situation it returns only 30.

This creates a challenge for the report designer, who cannot predict with certainty how many fields will return, and which will be in positions 11 through 30. If there is a reason for this field to be returned in only the one situation, I would suggest it be returned as the last field in the result. Ideally, the procedure should return the same set of fields in all situations.

The extra field is matched with an extra join in the Joins sections of the queries when both Activity and User filtering is in place.

**Summary:**

|  |  |  |
| --- | --- | --- |
| **Section in Procedure** | **SELECT Differences** | **JOIN Differences** |
| 1. | Both Activity and User Filtering | Returns attnd.name  | Joins dimAttendanceStatusJoins CachedReportIds on ActivityCacheJoins CachedReportIds on UserCache |
| 2. | Activity Filtering only | No differences | Joins CachedReportIds on ActivityCache |
| 3.  | User Filtering only | No differences | Joins CachedReportIds on UserCache |
| 4. | No Filtering | No differences | No differences |

\*Items in red are unexpected differences

The Excel file, Proc\_Comparison1.xls, included with this report, contains a more complete comparison of these query elements.

##  Test Series 2 – Attempt Star Fact Table

The following tests were conducted on three different customer databases. These are identified on the test server, ENG\_Dev8 as “ST0\_DA PreExtract”, ST2\_DA\_PreExtract”, and “ST3\_DA PreExtract ” In all of the following detailed test results, the first numeral indicates the test series (generally each series is directed at a specific table), the second numeral the test number and the third numeral the database.

### Test 1 – Identical “Grain”

This test is an inspection to verify that the tables contain data of the same grain.

**Test: 3.01**

**Database:** All three test databases

**Query:** (Direct inspection of tables)

**Result:** The records within the fact table and the source LMS table both represent a single “Attempt”. An Attempt record is not associated with a specific date (a timestamp in the source table is not imported) but the start and end date of the relevant Activity is recorded.

We observe that these records are updated in LMS as an employee proceeds through the training activity Attempt. Fields like “Grade” and “Success” are not relevant when the Attempt record is first created, but appear to be populated over time.

This means that existing records in the fact table will need to be updated over time. A record imported into the fact table may become out-of-synch with its corresponding record in LMS.

### Test 2 – Row Count Comparison

Differences in total row count between the fact table and the primary LMS source table are identified. These differences should be expected, explained and judged legitimate.

**Test: 3.02.01a**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT count(\*) FROM **TBL\_TMX\_Attempt**

**Result:**

509,164

**Test: 3.02.01b**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT count(\*) FROM **factAttempt**

**Result:**

509,164

**Test: 3.02.02a**

**Database:** ST2\_DA\_PreExtract

**Query:**

SELECT count(\*) FROM **TBL\_TMX\_Attempt**

**Result:**

341,021

**Test:** **3.02.02b**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT count(\*) FROM **factAttempt**

**Result:**

341,021

**Test: 3.02.03a**

**Database:** ST3\_DA\_PreExtract

**Query:**

**SELECT count(\*) FROM TBL\_TMX\_Attempt**

**Result:**

300,892

**Test:** **3.02.03b**

**Database:** ST3\_DA PreExtract

**Query:**

SELECT count(\*) FROM **factAttempt**

**Result:**

300,892

**Assessment:** The fact tables and the LMS source tables for all three databases are of matching length. No further investigation of row counts is required.

### Test 3 – Foreign Keys

All foreign keys in the fact table should have matches in the related dimension tables.

**Test: 3.03.01a**

**Database:** ST0\_DA PreExtract

**Query:** SELECT count(\*) FROM **factAttempt**

**Result:** 509,164

**Test: 3.03.01b**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT count(\*) FROM **factAttempt fact**

INNER JOIN dimUser emp ON emp.ID = fact.UserID

LEFT OUTER JOIN EmpCdDesc empcd ON empcd.EmpCd\_FK = emp.EmpCodeFK

LEFT OUTER JOIN EmpStatDesc empstat ON empstat.EmpStat\_FK = emp.EmpStatFK

INNER JOIN dimActivity act ON act.ID = fact.ActivityID

LEFT OUTER JOIN ActLabelDesc actlabel ON actlabel.ActLabel\_FK = act.ActivityLabelFK

INNER JOIN dimRegistrationStatus reg ON reg.ID = fact.RegistrationStatusID

INNER JOIN dimGrade dGrd ON dGrd.ID = fact.GradeID

INNER JOIN dimSuccess suc ON suc.ID = fact.SuccessID

INNER JOIN dimCompletionStatus compl ON compl.ID = fact.CompletionStatusID

INNER JOIN vwdimStartDate sd ON sd.DateID = fact.StartDtID

INNER JOIN vwdimEndDate ed ON ed.DateID = fact.EndDtID



**Result:** 509,164

**Test: 3.03.01c**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT

Count (\*)

FROM **TBL\_TMX\_Attempt atmpt**

 LEFT OUTER JOIN dimUser dUser ON atmpt.EmpFK = dUser.EmpFK

 LEFT OUTER JOIN dimActivity dAct ON atmpt.ActivityFK = dAct.ActivityFK

 LEFT OUTER JOIN vwdimStartDate dStartDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10), atmpt.StartDt), 0, 1)) = dStartDt.Date

 LEFT OUTER JOIN vwdimEndDate dEndDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10),

atmpt.EndDt), 0, 1)) = dEndDt.Date

 LEFT OUTER JOIN vwdimExpirationDate dExpDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10),

atmpt.ExpirationDate), 0, 1)) = dExpDt.Date

 LEFT OUTER JOIN dimCompletionStatus dCompletion ON atmpt.CompletionStatus = dCompletion.Value

 LEFT OUTER JOIN dimSuccess dSuccess ON atmpt.Success = dSuccess.Value

 LEFT OUTER JOIN dimGrade dGrade ON atmpt.GrdFK = dGrade.GradeFK

 LEFT OUTER JOIN dimAttendanceStatus dAttndStatus ON atmpt.AttndStatusFK = dAttndStatus.Value

 LEFT OUTER JOIN tmpCacheAttempt tc ON atmpt.Attempt\_PK = tc.Attempt\_PK AND

 atmpt.EmpFK = tc.EmpFK AND

 atmpt.ActivityFK = tc.Activity\_PK

 LEFT OUTER JOIN TBL\_TMX\_Registration reg ON reg.EmpFK = tc.EmpFK AND

 reg.ActivityFK = tc.Activity\_PK AND

 reg.Reg\_PK = tc.Reg\_PK

 LEFT OUTER JOIN dimRegistrationStatus dRegStatus ON reg.Status = dRegStatus.Value

**Result:** 509,164

**Note:** An extra query against the source table as an additional check.

**Test: 3.03.02a**

**Database:** ST2\_DA PreExtract

**Query:** SELECT count(\*) FROM factAttempt

**Result:** 341,021

**Test: 3.03.02b**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT count(\*)

FROM

factAttempt fact

INNER JOIN dimUser emp ON emp.ID = fact.UserID

LEFT OUTER JOIN EmpCdDesc empcd ON empcd.EmpCd\_FK = emp.EmpCodeFK

LEFT OUTER JOIN EmpStatDesc empstat ON empstat.EmpStat\_FK = emp.EmpStatFK

INNER JOIN dimActivity act ON act.ID = fact.ActivityID

LEFT OUTER JOIN ActLabelDesc actlabel ON actlabel.ActLabel\_FK = act.ActivityLabelFK

INNER JOIN dimRegistrationStatus reg ON reg.ID = fact.RegistrationStatusID

INNER JOIN dimGrade dGrd ON dGrd.ID = fact.GradeID

INNER JOIN dimSuccess suc ON suc.ID = fact.SuccessID

INNER JOIN dimCompletionStatus compl ON compl.ID = fact.CompletionStatusID

INNER JOIN vwdimStartDate sd ON sd.DateID = fact.StartDtID

INNER JOIN vwdimEndDate ed ON ed.DateID = fact.EndDtID

**Result:** 341,021

**Test: 3.03.02c**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT Count (\*)

FROM TBL\_TMX\_Attempt atmpt

 LEFT OUTER JOIN dimUser dUser ON atmpt.EmpFK = dUser.EmpFK

 LEFT OUTER JOIN dimActivity dAct ON atmpt.ActivityFK = dAct.ActivityFK

 LEFT OUTER JOIN vwdimStartDate dStartDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10), atmpt.StartDt), 0, 1)) = dStartDt.Date

 LEFT OUTER JOIN vwdimEndDate dEndDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10),

atmpt.EndDt), 0, 1)) = dEndDt.Date

 LEFT OUTER JOIN vwdimExpirationDate dExpDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10),

atmpt.ExpirationDate), 0, 1)) = dExpDt.Date

 LEFT OUTER JOIN dimCompletionStatus dCompletion ON atmpt.CompletionStatus = dCompletion.Value

 LEFT OUTER JOIN dimSuccess dSuccess ON atmpt.Success = dSuccess.Value

 LEFT OUTER JOIN dimGrade dGrade ON atmpt.GrdFK = dGrade.GradeFK

 LEFT OUTER JOIN dimAttendanceStatus dAttndStatus ON atmpt.AttndStatusFK = dAttndStatus.Value

 LEFT OUTER JOIN tmpCacheAttempt tc ON atmpt.Attempt\_PK = tc.Attempt\_PK AND

 atmpt.EmpFK = tc.EmpFK AND atmpt.ActivityFK = tc.Activity\_PK

 LEFT OUTER JOIN TBL\_TMX\_Registration reg ON reg.EmpFK = tc.EmpFK AND

 reg.ActivityFK = tc.Activity\_PK AND reg.Reg\_PK = tc.Reg\_PK

 LEFT OUTER JOIN dimRegistrationStatus dRegStatus ON reg.Status = dRegStatus.Value

**Result:** 341,021

**Test: 3.03.03a**

**Database:** ST3\_DA PreExtract

**Query:** SELECT count(\*) FROM factAttempt

**Result:** 300,892

**Test:** 3.03.03b

**Database:** ST3\_DA PreExtract

**Query:**

SELECT count(\*)

FROM

factAttempt fact

INNER JOIN dimUser emp ON emp.ID = fact.UserID

LEFT OUTER JOIN EmpCdDesc empcd ON empcd.EmpCd\_FK = emp.EmpCodeFK

LEFT OUTER JOIN EmpStatDesc empstat ON empstat.EmpStat\_FK = emp.EmpStatFK

INNER JOIN dimActivity act ON act.ID = fact.ActivityID

LEFT OUTER JOIN ActLabelDesc actlabel ON actlabel.ActLabel\_FK = act.ActivityLabelFK

INNER JOIN dimRegistrationStatus reg ON reg.ID = fact.RegistrationStatusID

INNER JOIN dimGrade dGrd ON dGrd.ID = fact.GradeID

INNER JOIN dimSuccess suc ON suc.ID = fact.SuccessID

INNER JOIN dimCompletionStatus compl ON compl.ID = fact.CompletionStatusID

INNER JOIN vwdimStartDate sd ON sd.DateID = fact.StartDtID

INNER JOIN vwdimEndDate ed ON ed.DateID = fact.EndDtID

**Result:**

300,892

**Test: 3.03.03c**

**Database:** ST3\_DA PreExtract

**Query:**

SELECT

Count (\*)

FROM TBL\_TMX\_Attempt atmpt

 LEFT OUTER JOIN dimUser dUser ON atmpt.EmpFK = dUser.EmpFK

 LEFT OUTER JOIN dimActivity dAct ON atmpt.ActivityFK = dAct.ActivityFK

 LEFT OUTER JOIN vwdimStartDate dStartDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10), atmpt.StartDt), 0, 1)) = dStartDt.Date

 LEFT OUTER JOIN vwdimEndDate dEndDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10),

atmpt.EndDt), 0, 1)) = dEndDt.Date

 LEFT OUTER JOIN vwdimExpirationDate dExpDt ON CONVERT(datetime, ROUND(CONVERT(decimal(19,10),

atmpt.ExpirationDate), 0, 1)) = dExpDt.Date

 LEFT OUTER JOIN dimCompletionStatus dCompletion ON atmpt.CompletionStatus = dCompletion.Value

 LEFT OUTER JOIN dimSuccess dSuccess ON atmpt.Success = dSuccess.Value

 LEFT OUTER JOIN dimGrade dGrade ON atmpt.GrdFK = dGrade.GradeFK

 LEFT OUTER JOIN dimAttendanceStatus dAttndStatus ON atmpt.AttndStatusFK = dAttndStatus.Value

 LEFT OUTER JOIN tmpCacheAttempt tc ON atmpt.Attempt\_PK = tc.Attempt\_PK AND

 atmpt.EmpFK = tc.EmpFK AND

 atmpt.ActivityFK = tc.Activity\_PK

 LEFT OUTER JOIN TBL\_TMX\_Registration reg ON reg.EmpFK = tc.EmpFK AND

 reg.ActivityFK = tc.Activity\_PK AND

 reg.Reg\_PK = tc.Reg\_PK

 LEFT OUTER JOIN dimRegistrationStatus dRegStatus ON reg.Status = dRegStatus.Value

**Result:**

300,892

**Assessment:** Adding the full set of table joins to the query does not change the row count returned. This means that no rows are being lost from the fact table due to missing dimension keys.

All keys present in the factTable are matched in the dimensions. This is the **Expected** and **Acceptable** result.

### Test 4 – Matching values within records

In addition to containing the correct number of rows, the fact table should contain matching data within those rows. Aggregating across the full set of records, using sums, averages and\or hash totals of common fields, is a high-level test to identify where differences exist.

The queries use the TRANSACT-SQL function CHECKSUM\_AGG.

From the MSDN Documentation of CHECKSUM: “CHECKSUM satisfies the properties of a hash function: CHECKSUM applied over any two lists of expressions returns the same value if the corresponding elements of the two lists have the same type and are equal when compared using the equals (=) operator. For the purpose of this definition, NULL values of a given type are considered to compare as equal. If one of the values in the expression list changes, the checksum of the list also usually changes. However, there is a small chance that the checksum will not change.”

**Test: 3.04.01a**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT

 CHECKSUM\_AGG(factAttempt.Score) AS Score\_CheckSum,

 CHECKSUM\_AGG(factAttempt.SuccessID) AS SuccessID\_CheckSum,

 SUM(factAttempt.Score) AS Score\_Sum2,

 SUM(factAttempt.SuccessID) AS SuccessID\_Sum,

 Count(ALL factAttempt.Score) AS Score\_Count,

 Count(All factAttempt.SuccessID) AS SuccessID\_Count,

 Count(factAttempt.UserID) AS [Rowcount]

FROM

 **factAttempt**

**Result:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Score\_CheckSum | SuccessID\_CheckSum | Score\_Sum2 | SuccessID\_Sum | Score\_Count | SuccessID\_Count | Rowcount |
| 36 | -2 | 3,231,000 | -507,658 | 44,451 | 509,164 | 509,164 |

**Test: 3.04.01b**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT

 CHECKSUM\_AGG(cast(TBL\_TMX\_Attempt.Score as int)) AS Score\_CheckSum,

 CHECKSUM\_AGG(cast(TBL\_TMX\_Attempt.Success as int)) AS Success\_Checksum,

 SUM(TBL\_TMX\_Attempt.Score) AS Score\_Sum,

 SUM(TBL\_TMX\_Attempt.Success) AS Success\_sum,

 COUNT(All TBL\_TMX\_Attempt.Score) AS Score\_Count,

 COUNT(All TBL\_TMX\_Attempt.Success) AS Success\_Count,

 COUNT(TBL\_TMX\_Attempt.EmpFK) AS [RowCount]

FROM

 **TBL\_TMX\_Attempt**

**Result:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Score\_CheckSum | SuccessID\_CheckSum | Score\_Sum2 | SuccessID\_Sum | Score\_Count | SuccessID\_Count | Rowcount |
| 36 | 1 | 3,231,000 | 705 | 44,451 | 801 | 509,164 |

**Test: 3.04.02a**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT

 CHECKSUM\_AGG(factAttempt.Score) AS Score\_CheckSum,

 CHECKSUM\_AGG(factAttempt.SuccessID) AS SuccessID\_CheckSum,

 SUM(factAttempt.Score) AS Score\_Sum2,

 SUM(factAttempt.SuccessID) AS SuccessID\_Sum,

 Count(ALL factAttempt.Score) AS Score\_Count,

 Count(All factAttempt.SuccessID) AS SuccessID\_Count,

 Count(factAttempt.UserID) AS [Rowcount]

FROM

 **factAttempt**

**Result:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Score\_CheckSum | SuccessID\_CheckSum | Score\_Sum2 | SuccessID\_Sum | Score\_Count | SuccessID\_Count | Rowcount |
| 122 | -1 | 6,130,940 | -340,881 | 64,386 | 341,021 | 341,021 |

**Test: 3.04.02b**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT

 CHECKSUM\_AGG(cast(TBL\_TMX\_Attempt.Score as int)) AS Score\_CheckSum,

 CHECKSUM\_AGG(cast(TBL\_TMX\_Attempt.Success as int)) AS Success\_Checksum,

 SUM(TBL\_TMX\_Attempt.Score) AS Score\_Sum,

 SUM(TBL\_TMX\_Attempt.Success) AS Success\_sum,

 COUNT(All TBL\_TMX\_Attempt.Score) AS Score\_Count,

 COUNT(All TBL\_TMX\_Attempt.Success) AS Success\_Count,

 COUNT(TBL\_TMX\_Attempt.EmpFK) AS [RowCount]

FROM

 **TBL\_TMX\_Attempt**

**Result:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Score\_CheckSum | SuccessID\_CheckSum | Score\_Sum2 | SuccessID\_Sum | Score\_Count | SuccessID\_Count | Rowcount |
| 122 | 0 | 6131062.3535 | 70 | 64,386 | 80 | 341,021 |

**Test: 3.04.03a**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT

 CHECKSUM\_AGG(factAttempt.Score) AS Score\_CheckSum,

 CHECKSUM\_AGG(factAttempt.SuccessID) AS SuccessID\_CheckSum,

 SUM(factAttempt.Score) AS Score\_Sum2,

 SUM(factAttempt.SuccessID) AS SuccessID\_Sum,

 Count(ALL factAttempt.Score) AS Score\_Count,

 Count(All factAttempt.SuccessID) AS SuccessID\_Count,

 Count(factAttempt.UserID) AS [Rowcount]

FROM

 **factAttempt**

**Result:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Score\_CheckSum | SuccessID\_CheckSum | Score\_Sum2 | SuccessID\_Sum | Score\_Count | SuccessID\_Count | Rowcount |
| 69 | -1 | 12141755 | -300675 | 129320 | 300892 | 300892 |

**Test: 3.04.03b**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT

 CHECKSUM\_AGG(cast(TBL\_TMX\_Attempt.Score as int)) AS Score\_CheckSum,

 CHECKSUM\_AGG(cast(TBL\_TMX\_Attempt.Success as int)) AS Success\_Checksum,

 SUM(TBL\_TMX\_Attempt.Score) AS Score\_Sum,

 SUM(TBL\_TMX\_Attempt.Success) AS Success\_sum,

 COUNT(All TBL\_TMX\_Attempt.Score) AS Score\_Count,

 COUNT(All TBL\_TMX\_Attempt.Success) AS Success\_Count,

 COUNT(TBL\_TMX\_Attempt.EmpFK) AS [RowCount]

FROM

 **TBL\_TMX\_Attempt**

**Result:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Score\_CheckSum | SuccessID\_CheckSum | Score\_Sum2 | SuccessID\_Sum | Score\_Count | SuccessID\_Count | Rowcount |
| 69 | 0 | 12141755. | 0 | 129320 | 217 | 300892 |

**Assessment:** There are differences between the fact table and LMS table on the Success ID field. Examining this field shows that there are null values in the LMS table that are converted to “-1” when imported into the factTable.

Filtering out -1 keys in the dimension table returns the same values as the query against TMX\_TBL\_Attempt.

**Query A:**

SELECT

 CHECKSUM\_AGG(factAttempt.SuccessID) AS SuccessID\_CheckSum,

 SUM(factAttempt.SuccessID) AS SuccessID\_Sum

FROM

 **factAttempt**

WHERE

factAttempt.SuccessID != -1

**Result:**

|  |  |
| --- | --- |
| SuccessID\_CheckSum | SuccessID\_Sum |
| 1 | 705 |

**Query B:**

SELECT

 CHECKSUM\_AGG(cast(TBL\_TMX\_Attempt.Success as int)) AS SuccessID\_CheckSum,

 SUM(TBL\_TMX\_Attempt.Success) AS SuccessID\_Sum

FROM

TBL\_TMX\_Attempt

WHERE

TBL\_TMX\_Attempt.Success is not null

**Result:**

|  |  |
| --- | --- |
| SuccessID\_CheckSum | SuccessID\_Sum |
| 1 | 705 |

The difference is therefore explained, and the result is judged acceptable.

### Test 5 – Key Fields 100% Populated

In the factTable, the key fields must be fully populated (no nulls.) Rather than count the null values, we have counted the non-null values, which gives a more concise query.

**Test: 3.05.01a**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT

 COUNT(ALL factAttempt.UserID) AS UserID\_Count,

 COUNT(ALL factAttempt.ActivityID) AS Activity\_Count,

 COUNT(ALL factAttempt.StartDtID) AS StartDtID\_Count,

 COUNT(ALL factAttempt.EndDtID) AS EndDtID\_Count,

 COUNT(ALL factAttempt.ExpirationDtID) AS ExpirationID\_Count,

 COUNT(ALL factAttempt.CompletionStatusID) AS CompletionStatusID\_Count,

 COUNT(ALL factAttempt.AttendanceStatusID) AS AttendanceStatusID\_Count,

 COUNT(ALL factAttempt.SuccessID) AS SuccessID\_Count,

 COUNT(ALL factAttempt.RegistrationStatusID) AS StatusID\_Count,

 COUNT(ALL factAttempt.Score) As Score\_Count,

 COUNT(\*) as 'RowCount'

FROM **factAttempt**

**Result:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| UserID\_Count | Activity Count | StartDtID\_Count | EndDtID\_Count | ExpirationID\_Count | CompletionStatusID\_Count | AttendanceStatusID\_Count | SuccessID\_Count | StatusID\_Count | Score\_Count | Row count |
| 509,164 | 509,164 | 509,164 | 509,164 | 509,164 | 509,164 | 509,164 | 509,164 | 509,164 | 44,451 | 509,164 |

**Test: 3.05.02a**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT

 COUNT(ALL factAttempt.UserID) AS UserID\_Count,

 COUNT(ALL factAttempt.ActivityID) AS Activity\_Count,

 COUNT(ALL factAttempt.StartDtID) AS StartDtID\_Count,

 COUNT(ALL factAttempt.EndDtID) AS EndDtID\_Count,

 COUNT(ALL factAttempt.ExpirationDtID) AS ExpirationID\_Count,

 COUNT(ALL factAttempt.CompletionStatusID) AS CompletionStatusID\_Count,

 COUNT(ALL factAttempt.AttendanceStatusID) AS AttendanceStatusID\_Count,

 COUNT(ALL factAttempt.SuccessID) AS SuccessID\_Count,

 COUNT(ALL factAttempt.RegistrationStatusID) AS StatusID\_Count,

 COUNT(ALL factAttempt.Score) As Score\_Count,

 COUNT(\*) as 'RowCount'

FROM **factAttempt**

**Result:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| UserID\_Count | Activity\_Count | StartDtID\_Count | EndDtID\_Count | ExpirationID\_Count | CompletionStatusID\_Count | AttendanceStatusID\_Count | SuccessID\_Count | StatusID\_Count | Score\_Count | Row count |
| 341021 | 341021 | 341021 | 341021 | 341021 | 341021 | 341021 | 341021 | 341021 | 64386 | 341021 |

**Test: 3.05.03a**

**Database:** ST3\_DA PreExtract

**Query:**

SELECT

 COUNT(ALL factAttempt.UserID) AS UserID\_Count,

 COUNT(ALL factAttempt.ActivityID) AS Activity\_Count,

 COUNT(ALL factAttempt.StartDtID) AS StartDtID\_Count,

 COUNT(ALL factAttempt.EndDtID) AS EndDtID\_Count,

 COUNT(ALL factAttempt.ExpirationDtID) AS ExpirationID\_Count,

 COUNT(ALL factAttempt.CompletionStatusID) AS CompletionStatusID\_Count,

 COUNT(ALL factAttempt.AttendanceStatusID) AS AttendanceStatusID\_Count,

 COUNT(ALL factAttempt.SuccessID) AS SuccessID\_Count,

 COUNT(ALL factAttempt.RegistrationStatusID) AS StatusID\_Count,

 COUNT(ALL factAttempt.Score) As Score\_Count,

 COUNT(\*) as 'RowCount'

FROM

 **factAttempt**

**Result:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| UserID\_Count | Activity\_Count | StartDtID\_Count | EndDtID\_Count | ExpirationID\_Count | CompletionStatusID\_Count | AttendanceStatusID\_Count | SuccessID\_Count | StatusID\_Count | Score\_Count | Row count |
| 300,892 | 300,892 | 300,892 | 300,892 | 300,892 | 300,892 | 300,892 | 300,892 | 300,892 | 129,320 | 300,892 |

**Assessment:** All fields in the fact table are filled except the “Score” field, which is the only “fact” field. This is the expected and acceptable result.

### Test 6 – Profile presence of nulls, zero-length strings and zero’s in fact fields

These values can be troublesome in constructing queries and reports. There is only one fact field in this table, “Score.”

**Test: 3.06.01**

**Database:** ST0\_DA PreExtract

**Query:**

SELECT

 COUNT(\*) AS Score\_Count, factAttempt.Score as 'Score'

FROM

 **factAttempt**

WHERE

 (factAttempt.Score = 0) OR

 (factAttempt.Score = '') OR

 (factAttempt.Score IS NULL)

GROUP BY

 **factAttempt.Score**

**Result:**

|  |  |
| --- | --- |
| **Score\_Count** | **Score** |
| 464,713 | null |
| 582 | 0 |

**Test: 3.06.02**

**Database:** ST2\_DA PreExtract

**Query:**

SELECT

 COUNT(\*) AS Score\_Count, factAttempt.Score as 'Score'

FROM

 **factAttempt**

WHERE

 (factAttempt.Score = 0) OR

 (factAttempt.Score = '') OR

 (factAttempt.Score IS NULL)

GROUP BY

 **factAttempt.Score**

**Result:**

|  |  |
| --- | --- |
| **Score\_Count** | **Score** |
| 1,734 | 0 |
| 276,635 | null |

**Test: 3.06.03**

**Database:** ST3\_DA PreExtract

**Query:**

SELECT

 COUNT(\*) AS Score\_Count,

 factAttempt.Score as 'Score'

FROM

 **factAttempt**

WHERE

 (factAttempt.Score = 0) OR

 (factAttempt.Score = '') OR

 (factAttempt.Score IS NULL)

GROUP BY

 factAttempt.Score

**Result:**

|  |  |
| --- | --- |
| **Score\_Count** | **Score** |
| 466 | 0 |
| 171,572 | null |

**Assessment:** The score field is very frequently null, and occasionally zero. Both are appropriate values under the correct circumstances. As we have seen in earlier tests, this field exactly matches the score field in LMS.

### Test 7 – Profile presence of -1 values in Key fields

These values are often troublesome and must be handled correctly. The results of this test, as with the one above, are helpful as alerts for later testing of the dimension tables and the procedures.

**Test: 3.07.01a**

**Database:** ST0\_DA PreExtract

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt.**UserID** = -1)

**Result:** Count=0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **ActivityID**= -1)

**Result:** Count=0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **StartDtID** = -1)

**Result:** Count= 34,989

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **EndDtID** = -1)

**Result:** Count= 128,808

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **ExpirationDtID** = -1)

**Result:** Count= 470,872

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **CompletionStatusID**= -1)

**Result:** Count= 499

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **AttendanceStatusID**= -1)

**Result:** Count= 3,327

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **SuccessID**= -1)

**Result:** Count= 508,363

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **RegistrationStatusID**= -1)

**Result:** Count= 166,096

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **GradeID**= -1)

**Result:** Count= 464,644

**Test: 3.07.02**

**Database:** ST2\_DA PreExtract

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt.**UserID** = -1)

**Result:** Count=0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **ActivityID**= -1)

**Result:** Count=0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **StartDtID** = -1)

**Result:** Count= 27,250

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **EndDtID** = -1)

**Result:** Count= 30,655

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **ExpirationDtID** = -1)

**Result:** Count= 341,021

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **CompletionStatusID**= -1)

**Result:** Count= 30,609

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **AttendanceStatusID**= -1)

**Result:** Count= 0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **SuccessID**= -1)

**Result:** Count= 340,951

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **RegistrationStatusID**= -1)

**Result:** Count= 84,443

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **GradeID**= -1)

**Result:** Count= 253,215

**Test: 3.07.03**

**Database:** ST3\_DA PreExtract

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt.**UserID** = -1)

**Result:** Count=0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **ActivityID**= -1)

**Result:** Count=0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **StartDtID** = -1)

**Result:** Count= 79,923

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **EndDtID** = -1)

**Result:** Count= 87,686

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **ExpirationDtID** = -1)

**Result:** Count= 300,892

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **CompletionStatusID**= -1)

**Result:** Count= 0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **AttendanceStatusID**= -1)

**Result:** Count= 0

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **SuccessID**= -1)

**Result:** Count= 300,675

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **RegistrationStatusID**= -1)

**Result:** Count= 217

**Query:** SELECT COUNT(\*) AS NegOneRows FROM factAttempt WHERE (factAttempt. **GradeID**= -1)

**Result:** Count= 128236

**Assessment**: Significant numbers of key fields are populated with “-1”. This value is used in the Star Schema where the field was null in the source table. For the “major” dimension, User and Attempt, where filtering will be applied, the field is always filled with a meaningful value.

##  Test Series 4 – dimUser Dimension Table

### Test 1 – Row Count Comparison

Differences in total row count between the dim table and the primary LMS source table are identified.

**Test: 4.01.01a**

**Database:** ST0\_DA PreExtract

**Query:** SELECT count(\*) FROM **dimUser**

**Result:** Count= 222,791

**Query:** SELECT count(\*) FROM dbo.**tblEmp**

**Result:** Count= 222,802

**Query:** SELECT count(\*) FROM dbo.**tblEmp** WHERE tblEmp.Emp\_IsDeleted = 0

**Result:** Count= 222,791

**Query:** SELECT \* FROM dbo.tblEmp WHERE tblEmp.Emp\_IsDeleted <> 0

**Result:** (see file **Test\_4.01.01a.CSV**)

**Query:**

SELECT SELECT COUNT(\*) FROM TBL\_TMX\_Attempt

 INNER JOIN tblEmp ON (TBL\_TMX\_Attempt.EmpFK=tblEmp.Emp\_PK)

WHERE (tblEmp.Emp\_IsDeleted <> 0)

**Result:** Count = 0

**Assessment:** Dimension table and LMS table differ only by the number of records for Employees flagged as deleted. These employees do not have records in the fact table. This is an acceptable result.

**Test: 4.01.01b**

**Database:** ST2\_DA PreExtract

**Query:** SELECT count(\*)FROM **dimUser**

**Result:** Count= 16,773

**Query:** SELECT count(\*)FROM dbo.**tblEmp**

**Result:** Count= 16,779

**Query:** SELECT count(\*)FROM dbo.**tblEmp** WHERE tblEmp.Emp\_IsDeleted = 0

**Result:** Count= 16,773

**Query:** SELECT \* FROM dbo.tblEmp WHERE tblEmp.Emp\_IsDeleted <> 0

**Result:** (see file **Test\_4.01.01b.CSV**)

**Query:**

SELECT SELECT COUNT(\*) FROM TBL\_TMX\_Attempt

 INNER JOIN tblEmp ON (TBL\_TMX\_Attempt.EmpFK=tblEmp.Emp\_PK)

WHERE (tblEmp.Emp\_IsDeleted <> 0)

**Result:** Count = 0

**Assessment:** Dimension table and LMS table differ only by the number of records for Employees flagged as deleted. These employees do not have records in the fact table. This is an acceptable result.

**Test: 4.01.01c**

**Database:** ST3\_DA PreExtract

**Query:** SELECT count(\*)FROM **dimUser**

**Result:** Count= 19,183

**Query:** SELECT count(\*)FROM dbo.**tblEmp**

**Result:** Count= 19,193

**Query:** SELECT count(\*)FROM dbo.**tblEmp** WHERE tblEmp.Emp\_IsDeleted = 0

**Result:** Count= 19,183

**Query:** SELECT \* FROM dbo.tblEmp WHERE tblEmp.Emp\_IsDeleted <> 0

**Result:** (see file **Test\_4.01.01c.CSV**)

**Query:**

SELECT SELECT COUNT(\*) FROM TBL\_TMX\_Attempt

 INNER JOIN tblEmp ON (TBL\_TMX\_Attempt.EmpFK=tblEmp.Emp\_PK)

WHERE (tblEmp.Emp\_IsDeleted <> 0)

**Result:** Count = 0

**Assessment:** Dimension table and LMS table differ only by the number of records for Employees flagged as deleted. These employees do not have records in the fact table. This is an acceptable result

**Overall Assessment:** The dimUser tables are accurate representations of the tblEmp tables.

### Test 2 – Profile presence of null, “Missing”, and other troublesome values

**Test: 4.02.01**

**Database:** ST0\_DA PreExtract

**Query:** Select Count(\*) from dimUser where **EmpLName** Like '%Missing%'

**Result:** Count= 26 rows

**Query:** SELECT Count(\*) from dimUser where **EmpLName** is null

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCode** is null

**Result:** Count= 218,856 rows

**Query:** SELECT Count(\*) from dimUser where dimUser. **PrimaryJobName** is null

**Result:** Count= 22,712 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**PrimaryOrgName** is null

**Result:** Count= 20,328 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**PrimaryDomName** isnull

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**MgrEmpFullName1** isnull

**Result:** Count= 5,686 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpStartDt** isnull

**Result:** Count= 222,388 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpEmail** isnull

**Result:** Count= 127,424 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCity** isnull

**Result:** Count= 221,952 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpState** isnull

**Result:** Count= 221,952 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCntry** isnull

**Result:** Count= 221,946 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpZip** isnull

**Result:** Count= 221,956 rows

**Test: 4.02.02**

**Database:** ST2\_DA PreExtract

**Query:** Select Count(\*) from dimUser where **EmpLName** Like '%Missing%'

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where **EmpLName** is null

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCode** is null

**Result:** Count= 256 rows

**Query:** SELECT Count(\*) from dimUser where dimUser. **PrimaryJobName** is null

**Result:** Count= 9,179 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**PrimaryOrgName** is null

**Result:** Count= 8,312 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**PrimaryDomName** isnull

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**MgrEmpFullName1** isnull

**Result:** Count= 3,010 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpStartDt** isnull

**Result:** Count= 258 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpEmail** isnull

**Result:** Count= 16,773 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCity** isnull

**Result:** Count= 257 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpState** isnull

**Result:** Count= 257 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCntry** isnull

**Result:** Count= 4846 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpZip** isnull

**Result:** Count= 257 rows

**Test: 4.02.03**

**Database:** ST3\_DA PreExtract

**Query:** Select Count(\*) from dimUser where **EmpLName** Like '%Missing%'

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where **EmpLName** is null

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCode** is null

**Result:** Count= 121 rows

**Query:** SELECT Count(\*) from dimUser where dimUser. **PrimaryJobName** is null

**Result:** Count= 19,183 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**PrimaryOrgName** is null

**Result:** Count= 7,411 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**PrimaryDomName** isnull

**Result:** Count= 0 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**MgrEmpFullName1** isnull

**Result:** Count= 4,174 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpStartDt** isnull

**Result:** Count= 2,391 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpEmail** isnull

**Result:** Count= 19,183 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCity** isnull

**Result:** Count= 2,297 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpState** isnull

**Result:** Count= 3,280 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpCntry** isnull

**Result:** Count= 1,315 rows

**Query:** SELECT Count(\*) from dimUser where dimUser.**EmpZip** isnull

**Result:** Count= 7,542 rows

**Assessment:** Whilethe key fields for this dim table are clean, and the last name field contains a small number of “Missing Name” records (which appear on visual inspection to be test records), there are many fields in the table which will return NULL (especially in ST0\_DA PreExtract.). The fields reviewed here are those that are returned by the reporting Stored Proc, “lmssp\_GetEmpActRegistration”.

These NULLS are reflected in the base LMS tables, so there presence in the dimTable is not an error per se, but the reports generated from this procedure can be affected. In the ST0\_DA PreExtract database some fields, the City and State fields for example, contain very little usable data. Reports designed from this procedure should account for the many nulls returned from this table.

This result is acceptable in that it is not a bug in the Star Schema, but it is a potential source of trouble in the creation of reports and should be noted.

*[This document continues in this fashion for another 50 pages]*