Data Transformation Process for Event Data Warehouse.

Table of Contents

Introduction 2

Step 0: Get Data Into the Standard Format 2

Input Data Structure 2

Data Map 2

SQL 3

Description of the MyLyn Map 5

Overview of CANS WAREHOUSE Data Flow and Transformation Processes 6

Extract Distinct Reference Points 6

Explode Relational Data 10

Create Dimension Records From Reference Tables 13

Create Final Fact Table 13

An example of full execution of the physical import process using data imported from the new, Oracle version of the CANS tables 14

Following the Procedures 15

References 15

# Introduction

This is a fifteen or sixteen step data transformation process. Give or take.

# Step 0: Get Data Into the Standard Format

We begin with step 0, which is getting the data from a source system into the required input format. For example, data from the MyLyn system was in this format:

## Input Data Structure

## Picture 38.png

## Data Map

We created a map to move the data from that format, to the following format:

The first explosion covers people who comment on a bug. This map creates a relationship between people commenting on a bug, and the person who is assigned that bug.

Table - MyLyn - Data map for explosion one

|  |  |  |
| --- | --- | --- |
| **First Explosion** |  |  |
| Events\_Stage Field | mylyn.research\_comments\_by\_release field | Type |
|  |  |  |
| ID | autoincrement or create | generate |
| Environement\_code | Bugzilla | literal |
| context\_name | Mylyn | literal |
| context\_id | MyLyn | map |
| context\_type | Software Engineering | literal |
| event\_ip | 0 | literal |
| event\_session | research\_comment\_by\_release.releaseid | map |
| event\_action | jforum.new | literal |
| event\_object | research\_comment\_by\_release.short\_desc | map |
| event\_url | research\_comment\_by\_release.bug\_id | map |
| event\_author\_id | research\_comment\_by\_release.author\_id | map |
| event\_author\_name | n/a | literal |
| event\_date | research\_comment\_by\_release.commentTS | map |
| object\_creator | research\_comment\_by\_release.assigned\_id | map |

The second explosion, in this case, covers people who were the original bug creators, and subsequently creates a connection between those creators and the people commenting on the bug.

Table - MyLyn, Data map for explosion two

|  |  |  |
| --- | --- | --- |
| **Second Explosion** |  |  |
| Events\_Stage Field | mylyn.research\_comments\_by\_release field |  |
|  |  |  |
| ID | autoincrement or create | generate |
| Environement\_code | Bugzilla | literal |
| context\_name | MyLyn | literal |
| context\_id | MyLyn | map |
| context\_type | Software Engineering | literal |
| event\_ip | 0 | literal |
| event\_session | research\_comment\_by\_release.releaseid | map |
| event\_action | jforum.read | literal |
| event\_object | research\_comment\_by\_release.short\_desc | map |
| event\_url | research\_comment\_by\_release.bug\_id | map |
| event\_author\_id | research\_comment\_by\_release.author\_id | map |
| event\_author\_name | n/a | literal |
| event\_date | research\_comment\_by\_release.commentTS | map |
| object\_creator | research\_comment\_by\_release.reporter\_id | map |

## SQL

This SQL is used to move the data from the Mylyn database table to the tables in the CANS\_Warehouse..

[9/9/10 9:09:11 AM] Michael Gallagher:

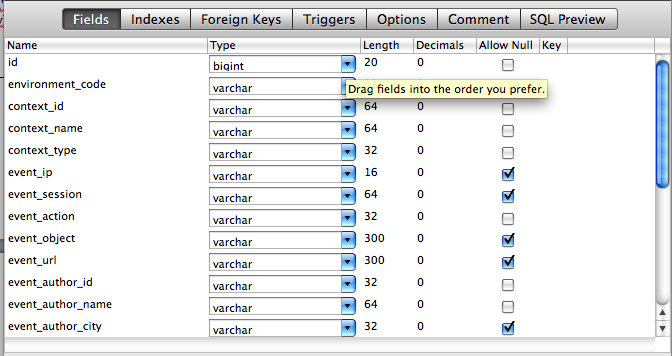
Table - Mylyn, data mapping into warehouse structure SQL

|  |
| --- |
| Step 1:  insert into cans\_warehouse.events\_stage\_mylyn(environment\_code,           context\_id,           context\_name,           context\_type,           event\_ip,           event\_session,           event\_action,           event\_object,           event\_url,           event\_author\_id,           event\_author\_name,           event\_author\_city,           event\_author\_state,           event\_author\_zip\_code,           event\_author\_country,           event\_author\_latitude,           event\_author\_longitude,           event\_date,           event\_object\_type,           object\_creator,           weight\_in\_minutes,           row\_type,           parent\_id)  select 'bugzilla',’mylyn’,'mylyn','Software Engineering',  '0',mylyn.research\_comment\_by\_release.releaseid,'jforum.new',  mylyn.research\_comment\_by\_release.short\_desc,mylyn.research\_comment\_by\_release.bug\_id,  mylyn.research\_comment\_by\_release.author\_id,' ',' ',' ',' ',' ',' ',' ',  mylyn.research\_comment\_by\_release.commentts,' ',mylyn.research\_comment\_by\_release.assigned\_id,  ' ',' ',' '  from mylyn.research\_comment\_by\_release  Step 2:  insert into cans\_warehouse.events\_stage\_mylyn(environment\_code,           context\_id,           context\_name,           context\_type,           event\_ip,           event\_session,           event\_action,           event\_object,           event\_url,           event\_author\_id,           event\_author\_name,           event\_author\_city,           event\_author\_state,           event\_author\_zip\_code,           event\_author\_country,           event\_author\_latitude,           event\_author\_longitude,           event\_date,           event\_object\_type,           object\_creator,           weight\_in\_minutes,           row\_type,           parent\_id)  select 'bugzilla',’mylyn’,'mylyn','Software Engineering',  '0',mylyn.research\_comment\_by\_release.releaseid,'jforum.read',  mylyn.research\_comment\_by\_release.short\_desc,mylyn.research\_comment\_by\_release.bug\_id,  mylyn.research\_comment\_by\_release.author\_id,' ',' ',' ',' ',' ',' ',' ',  mylyn.research\_comment\_by\_release.commentts,' ',mylyn.research\_comment\_by\_release.reporter\_id,  ' ',' ',' '  from mylyn.research\_comment\_by\_release |

## Description of the MyLyn Map

The Mylyn data differs from CANS data in one important way – there are two types of object creators, and the map between the event author and the object creator is already in the file. We don’t need to go get it. The data is also less rich than CANS data, because we do not get an event each time something is read. You can see from the map above that we refer to the connection between the bug creator and the bug commenter as a “read” connection. This approximates the same concept of a “read” in CANS.

The data is now in the warehouse table “Events\_Stage”:



At this point the data that is being brought in, begins the procedural part written for CANS data.

The figure, below, lists the items that change for each load procedure. Mike’s next goal is to handle deletion of dimension data better when a load fails. Also, there are two data sets that remain to be loaded.

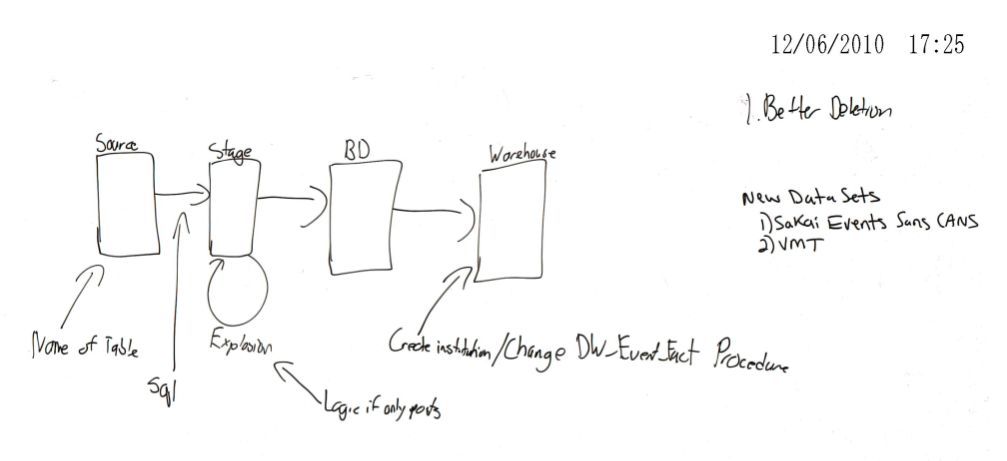


Figure - Overview of what changes during loads of different data

# Overview of CANS WAREHOUSE Data Flow and Transformation Processes

Figure one provides a conceptual overview of all of the steps to follow. First, we get events from source, which is already described in the first part of “Step 0”. Second, data is moved to the staging area, which for now, is simply the “EVENTS STAGE” table, also described in step 0.

## Extract Distinct Reference Points

When we go to “Extract Distinct Reference Points”, we are extracting the data into a set of “BD” tables (which is an acronym for the project code name, ‘Bucky Dome 2’. This step is about inventorying all of the components, about which we have event data. The “BD” tables are a staging area. They are truncated at the conclusion of every successful load process. The critical components are enumerated in the box. These are:

1. Context –
   1. Context ID – For example, with CANS, the context id is the course ID. The context ID for Bugzilla is “MyLyn”
   2. Context Name – For example, the context with CANS is “sakai” and for BugZilla it is “Bugzilla”
2. Context Type for Sakai is “Group” and for Bugzilla it is “Software Engineering”.
3. Event Action in Sakai is determined by CANS. We map Bugzilla events to jforum.new and jforum.read for now, as it makes subsequent analysis more consistent. In the future we should develop a more abstract name for these event types. Some of the logic to do this is in the procedures that reference the table “event\_action\_mapping”, but we have not fully tested this logic. This should be tested in the future; we think its done…

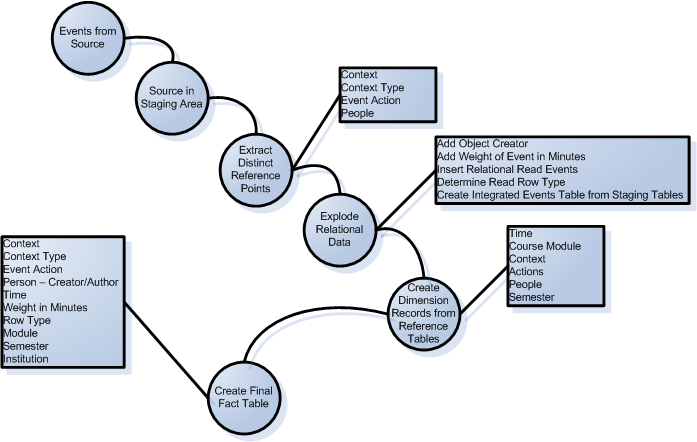


Figure - Overall Flow Diagram, Conceptual Level

Figure two shows “extract distinct reference points” as four procedures: Insert\_Context, Insert\_EventActions, Insert\_Persons\_withcreators, and Insert\_context\_type.

In the case of the MyLyn loading, which was our first attempt at loading a data set other than CANS, we took a careful, stepwise approach and did not run the “loading master program” that performs all steps automatically.

Table - SQL Used during the MyLyn Data Import

|  |
| --- |
| update events\_stage\_mylyn set context\_id = 'mylyn';  truncate table events\_stage;  truncate table bd\_context;  truncate table bd\_events;  delete from DW\_Events\_Fact where institution\_id=2; -- This deletes from previous runs in the fact table for MyLyn. We had to do this as we experimented with data. Ensuring that each data set has a distinct institution ID, or some group of data that we can use for this process is critical.  delete from DW\_Context\_Dim where DW\_Context\_Dim.Context\_name != 'sakai'; -- this is a similar function for deleting context during trial runs.  --Here, we see the events\_stage  insert into events\_stage  select \* from events\_stage\_mylyn;  ---Here, Mike is just redoing the necessary parts after an initial run. These parts had to be changed because we mapped the context wrong initially.  call insert\_context();  call insert\_into\_events\_stage();  call update\_rows\_for\_sna\_type();  call insert\_events\_withcreators();  call DW\_insert\_context\_dim();  call DW\_insert\_events\_fact(); |

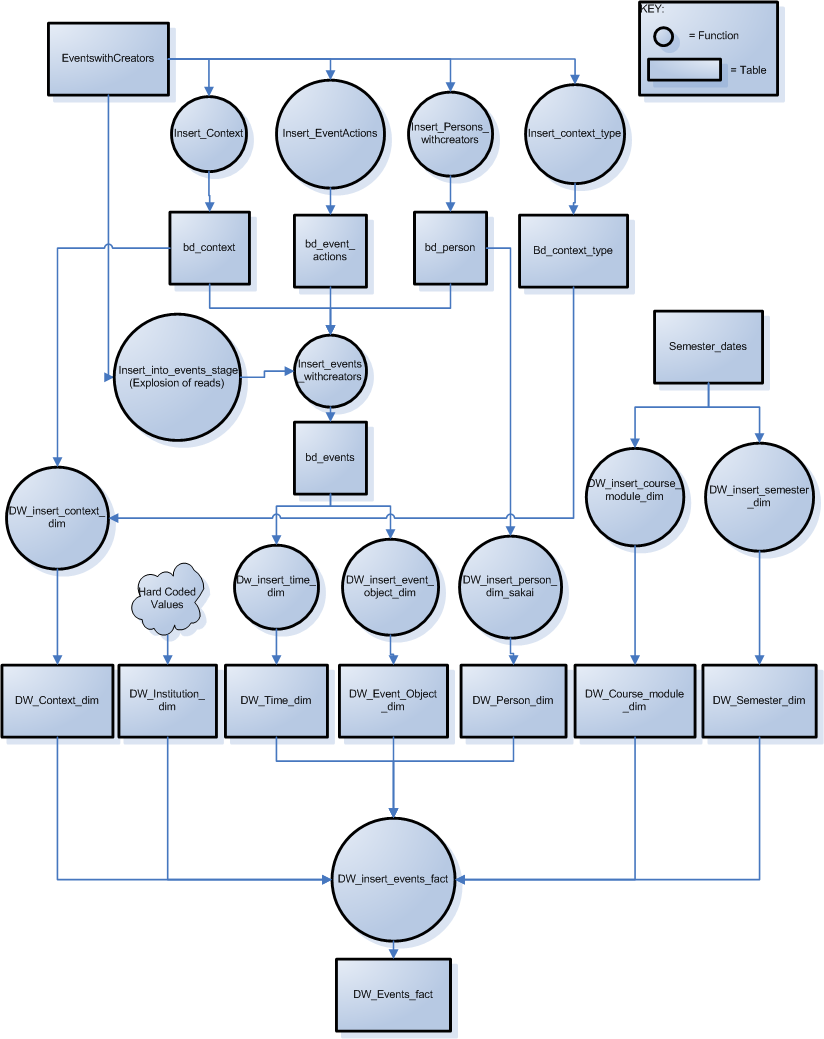


Figure - Technical Description of Procedures Executed

## Explode Relational Data

Conceptually, there are five main steps in this process: Add the object creator, add weights to the events based on minutes that transpired, inserting relational read events, determining the read row type and creating integrated events table from staging tables.

With “Add Object Creator” we are accounting for a shortcoming in the original CANS data set, where the person who created the object being read by the event author is not known. In CANS data, however, we can determine who created the object by looking at the object name, and searching for the original jforum.new event – the oldest event for that particular object. With reference to the CANS data, “event\_url” represents the object. There is a unique event\_url for each discussion board and resource. We did not need to do this in the MyLyn data because the original source data included the event\_object\_creator.

The presence or absence of an event\_object\_creator in the source data is important to understand, with reference to figure four. In figure four, we see the procedure insert\_into\_events\_stage is called. This procedure has two sources in the database: insert\_into\_events\_stage\_cans and insert\_into\_events\_stage\_mylyn. It is **vital** to make sure that the base procedure; insert\_into\_events\_stage is substituted with the appropriate procedure, depending on the nature of the source data. In the future, this logic should be handled as an input parameter.

Adding weights to each event in minutes incorporates traversing the original events from the “events\_stage” table and calculating the minutes between each read or post event and the prior read or post event in a sequence (discussion board). The cardinality of this data is the same as the original event data loaded from the source.

In the next step, we account for the fact that many reads and posts in CANS include reference, at least tacitly, to more than one previous event. We have tuned an algorithm to first identify these events, then apply a timestamp difference weight to them based on the distance from each prior post. This is illustrated in two recently published papers (Goggins, Laffey, Amelung, & Gallagher, 2010; Goggins, Galyen, & Laffey, 2010) in figure 3.

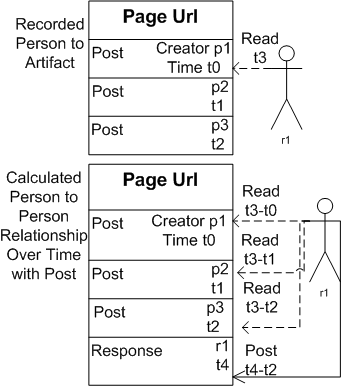


Figure - Conceptual view of the insert relational read events logic, with the top view showing what is originally recorded, and the bottom view showing what happens in the explosion

To understand how this fits with the rest of “Explode relational data”, a more detailed logical view is presented in figure 4.

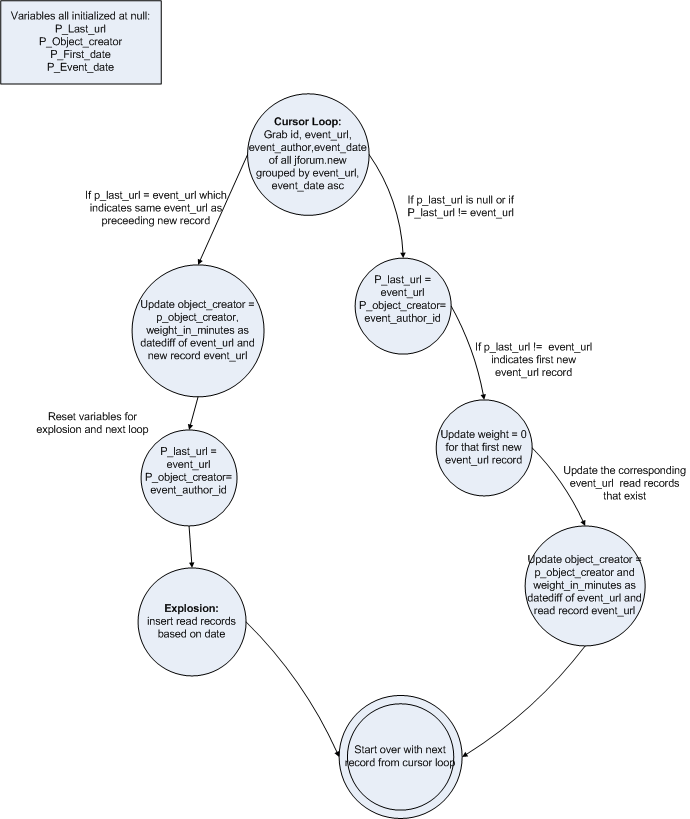


Figure - This is a detailed view of the "Explode Relational Data", including the "insert relational read events"; but it shows the steps in context with everything else involved in "Explode Relational Data"

The fourth step in “Explode Relational Data” is “Determine Read Row Type”. The purpose of this step is to provide metadata in the fact table, DW\_EVENT\_FACT, that shows us which rows were originally in the source data and which rows were added. This step is the same for either data input type (CANS or, the other current example, MyLyn).

The final step in “explode relational data” integrates the work of the previous steps to produce a single staging table with all exploded events and associated metadata. This is accomplished with the “insert\_events\_withcreators” procedure, noted in figure four. Also noted in figure two is the resulting table at the conclusion of this process, which is the “bd\_events” table.

## Create Dimension Records From Reference Tables

With BD\_EVENTS created, we are now ready to add rows, as necessary, to the dimension tables in the warehouse. The key dimensions are time, course module, context, actions, people and semester. Figure two illustrates the six procedures used to populate these dimensions. It is critical to note that, before populating new rows in the dimensions, the procedures check for existing rows with the same values. A seventh dimension for institution is hard coded. This seldom changes. For example, it is a single value for the CANS data, and also a single value for the MyLyn data.

## Create Final Fact Table

Rows are added to the “Final fact table”, DW\_EVENT\_FACT, by the procedure Dw\_insert\_events\_fact, which inserts one row per row in the BD\_EVENTS table, and creates a corresponding link with the dimensions. This results in a star schema, or “data mart”/”data warehouse” structure for events data that is optimized for analysis purposes. The key elements of the event fact, noted in figure two, are context, context type, event action, person – creator/author, time, weight in minutes, row type, module, semester and institution.

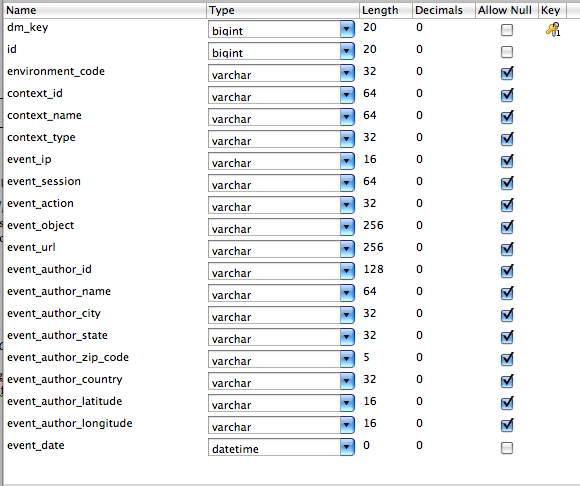
Since event fact

Table - Event Fact Data Description

|  |  |
| --- | --- |
| Event Fact Column | Description of Data |
| Context | Context is constituted by the context name, the context type and the events\_context\_id. In the MyLyn data, these elements are “Bugzilla”, “Software Engineering” and “MyLyn” respectively. In the CANS data, these elements are “Sakai”, “Group” and the course id (there are nearly 100). |
| Context Type | Software engineering, or “Group” What type of interaction context is this, exactly. |
| Event Action | The type of action. For now, this is “read” or “create”; jforum.read or jforum.new. |
| Person – Creator | Person who created the event record |
| Person – Author | Person who created the object the event record measures |
| Time | Timestamp of the event, which points to a time dimension, allowing for systematic temporal slicing of the data. |
| Weight in Minutes | Time distance from the previous event |
| Row Type | Whether the event row is from source data, or inferred from source data based on approximations of user behavior (i.e., in Sakai, a user is likely to view the five previous posts to some diminishing extent when reading or posting). |
| Module | Can be mapped to a module in a course, if this data is known. |
| Semester | Can be mapped based on the dates of the events. An imperfection in the current mapping of mylyn data is that we use the event\_session for release metadata. It would be more optimal to leverage the semester as a boundary for release.  This is something to explore in future releases. In the future, we may wish to connect institution id and semester in a single version of the fact table. |
| Institution | Hard coded. Currently the CANS host institution or “Drexel”, for the MyLyn data. |

# An example of full execution of the physical import process using data imported from the new, Oracle version of the CANS tables

The events from oracle are saved using MySQL Import from an Oracle Database restore. The table is events\_from\_oracle. It has the following structure:



This structure mirrors the events structure in the current version of the CANS database. There is a significant flaw in the person data in the Oracle database for CANS: Some people have more than one unique event\_author\_id. This appears to be an artifact of an authentication conversion in the Sakai environment. We write the following SQL Script to map duplicates back to the shortest event\_author\_id string, which is the one most likely to exist in previous CANS data sets. Table six illustrates the SQL we used to correct this issue.

Table - Author Correction SQL

|  |
| --- |
| update events\_from\_oracle a, events\_from\_oracle b  set a.event\_author\_id = b.event\_author\_id  where a.event\_author\_name = b.event\_author\_name  and length(b.event\_author\_id) < length(a.event\_author\_id) |

## Following the Procedures

The next steps involve following a set of procedures that we have developed over the past year to import the data. Logically, those procedures are described above. The physical execution follows this precise path:

1. Execute clean\_events\_stage
2. Execute start\_to\_finish … which inside of itself, executes these procedures in sequence. During the execution, the cans\_warehouse.import\_status table is updated as each step is completed. In this way, if there is a failure, we have one place to go to evaluate the nature of that failure.
   1. Execute insert\_from\_events\_to\_events\_stage – this procedure has a line in it that specifies the table where your source data is in the **from** clause.
   2. Analyze Table – this makes sure the table has the right statistics, which is a performance enhancing step (a big performance enhancing step)
   3. Populate BD tables
      1. Execute Insert\_context\_type
      2. Execute insert\_context
      3. Execute insert\_event\_actions
      4. Execute insert\_person\_withcreators\_nonames
   4. Explode the data
      1. Execute insert\_into\_events\_stage – here, like in the case of MyLyn, you want to make sure you are using the correct version of the procedure. There is one version for source data that already has event\_creators, and another for data that does not have event\_creators
         1. Insert\_into\_events\_stage\_cans – no event creators
         2. Insert\_into\_events\_stage\_mylyn – with events creators
   5. Analyze events\_stage
   6. Update rows for SNA type
      1. Execute update\_rows\_for\_sna\_type
   7. Insert data into the BD\_Events table
      1. Execute insert\_events\_withcreators
   8. Execution scripts to add rows to dimensions as needed
      1. Execute DW\_insert\_context\_dim
      2. Execute dw\_insert\_time\_dim
      3. Execute dw\_insert\_event\_object\_dim
      4. Execute dw\_insert\_person\_dim\_sakai
      5. Execute dw\_insert\_events\_fact – there is an institutional id that might be updated for each set of data imported. For example, we called the oracle data from Mizzou “institution 3”; allowing for easier backup and identification of corpus origin within the data warehouse.

# UCern Mappings

## Input Data Structure & Data Map

Contained in the warehousemap.xlsx file in the directory.



## Explosions

1. The most important thing to understand about the uCern and Coffee Party data is that it reflects read only data, and this read only data requires a different explosion procedure: insert\_into\_events\_stage\_posts\_only
2. Everything else is the same.

# References

Goggins, S., Galyen, K., & Laffey, J. (2010). *Network Analysis of Trace Data for the Support of Group Work: Activity Patterns in a Completely Online Course*. Proceedings from ACM Group 2010, Sanibel Island, FL.

Goggins, S. P., Laffey, J., Amelung, C., & Gallagher, M. (2010). *Social Intelligence In Completely Online Groups*. Proceedings from IEEE International Conference on Social Computing, Minneapolis, MN.