

## Team Process Data Warehouse – High-Level Schema for Core Data

As described in the “roadmap” document, an iterative development strategy has been used to add new types of data to the warehouse over time. The initial iterations focused on a small, core set of data. Additional elements have been added over time.

This document describes the data elements that are currently present in the data warehouse.

### Audience and Purpose of this Document

The primary goal of this document is to provide a high-level overview of the data elements in the warehouse, along with their relationships to each other. This document aims to provide simple, high-level descriptions that are accessible to a nontechnical audience.

This document focuses on data elements that are meaningful to an individual who is retrieving data from the warehouse (for example, to generate a report). To minimize distractions for the nontechnical audience, this document does not describe supplementary elements that are primarily of interest to DBAs (for example, indexes) or to the developers of ETL logic (for example, hashcode columns for performing quick comparisons).

This document strives to be database agnostic. As a result, this document uses abstract data type names (such as “Integer” or “String”) to describe the types of various data elements. These abstract type names map to SQL storage types in a straightforward way, but that mapping is inherently database-specific. The mappings for a particular database could be provided in a separate document if needed.

This document also strives to be tool-agnostic. Details that are specific to a particular TSP/TPI\* tool can be found in a separate document.

Development on the warehouse schema is still actively underway. As a result, the table and column names in this document are subject to change.

---

\* TSP<sup>SM</sup> is used by teams working in a wide variety of problem domains (e.g. software, hardware, services). Since these activities are not limited to software, the name “Team Process Integrated” and the acronym “TPI” are used in this document to describe the full range of TSP-inspired high-maturity processes, and to avoid improper use of Carnegie Mellon service marks. TSP is a service mark of Carnegie Mellon University. Carnegie Mellon University has neither contributed to nor evaluated the contents of this document.

## Dimensions

The following dimensions will be included in the warehouse:

- ETL Audit Log
- Organization
- Team
- Person
- Project
- EV Schedule
- WBS Element
- Task
- Process
- Plan Item
- Process Enactment
- Data Block
- Size Metric
- EV Metric
- Defect Type
- Measurement Type
- Dependency Type
- Attribute
- Text
- Date
- Baseline

## Fact Tables

The following fact tables will be included in the warehouse:

- Time Log
- Defect Log
- Task Status
- Task Dates
- EV Schedule Periods
- EV Metric Values
- Size
- Plan Item Attribute
- Plan Item Dependency
- Plan Item Note
- Process Enactment

## Conventions

Within this document, several important conventions are used.

### Versioned Data

To support the complete spectrum of enterprise analysis needs, it is crucial for the warehouse to preserve the history of changes to project data. This history makes it possible to analyze trends, evaluate the ROI of process changes, and more.

That power notwithstanding, the vast majority of analyses are likely to be focused on current data. To strike a balance between power and complexity, versioned historical data will be placed into tables with the suffix “\_HIST.” Then, views will be provided that constrain the data to include only “current” rows.

### Hierarchical Columns

Some columns contain data which is naturally hierarchical. When this occurs, the levels of the hierarchy are stored in numbered columns such as SOME\_COLUMN\_1, SOME\_COLUMN\_2, SOME\_COLUMN\_3, etc. In these scenarios, an additional column is generally provided that contains the full hierarchical path. This “full path” column has the same name, without the final number (for example, SOME\_COLUMN).

This naming convention was selected because it will allow a clean migration path for columns to become hierarchical over time. For example, if the initial schema contains a column called TASK\_NAME, and future analysis determines that task names contain hierarchical information, the schema could be extended with new columns TASK\_NAME\_1, TASK\_NAME\_2, etc. This change could be made to the schema without breaking any existing reports or analyses.

## Dimension: ETL Audit Log

The ETL Audit dimension captures important metadata about the flow of data into the data warehouse.

Column	Type	Description
ETL_BATCH_KEY	Integer	A uniquely assigned number for this row
ETL_BATCH_APP_NAME	String	The name of the ETL batch application that ran; for example, "Process Dashboard"
ETL_BATCH_APP_VERSION	String	The version number of the ETL application logic
ETL_BATCH_START_TIME	Timestamp	The date/time when the batch run started
ETL_BATCH_FINISH_TIME	Timestamp	The date/time when the batch run finished
ETL_BATCH_ELAPSED_SECONDS	Number	The number of seconds of elapsed time
ETL_BATCH_SOURCE, ETL_BATCH_SOURCE_IDENTIFIER	String	Descriptions of the external data source that the ETL process drew its data from ( <i>See note below</i> )
ETL_BATCH_SOURCE_DATE	Timestamp	The effective date/time of the source data; for example, the time it was extracted from the source system
ETL_BATCH_ERROR_TEXT_KEY	Integer	If the batch process encountered an error during its operation, this field will point to a record in the Text table which contains detailed information about the error.

Each time an ETL application reads data from some other system and writes it into the warehouse, it will insert a new row into this table.

The other dimension and fact tables in the warehouse will include a column called "ROW\_CREATED\_BY\_KEY" and a column called "ROW\_LAST\_UPDATED\_BY\_KEY." As the ETL process writes data into these other tables, it will populate the ROW\_CREATED\_BY\_KEY column with the key of the row it created in this table. Thus, each dimension/fact table row can be traced back to the ETL batch process which created it. When an ETL process modifies an existing row in a dimension or fact table, the ETL process should write its ETL\_BATCH\_KEY into the ROW\_LAST\_UPDATED\_BY\_KEY column.

The ETL\_BATCH\_SOURCE and ETL\_BATCH\_SOURCE\_IDENTIFIER columns are free-form text fields that can be populated as needed by the ETL application to describe the origin of the source data. For example:

- For data read from a web service, this could be the URI of the web service.
- For data read from a corporate database, this could be a descriptive identifier for the database.
- For data read from an extract file, this could be the filename of the extract file.
- For data read from an external source like a spreadsheet, this could be the filename of the spreadsheet and/or a unique GUID stored within it.

A given ETL application should establish its own naming conventions for the data it writes into these columns. However, if the string might contain any personally identifiable information (such as the name of an individual or their initials), that information must be hashed or scrambled to protect data privacy.

The ETL\_BATCH\_SOURCE column should usually contain a thorough, human-readable description of the source that was used for this particular batch load. In contrast, the ETL\_BATCH\_SOURCE\_IDENTIFIER column should include a programmatic descriptor for this data source that is unique and is not time-varying. As an example, a batch process might populate a particular row with an ETL\_BATCH\_SOURCE value of "C:\tpidw\data\_extracts\primavera\primavera\_task\_status\_extract\_2013-04-13\_12-01-36.xml" and an ETL\_BATCH\_SOURCE\_IDENTIFIER value of "primavera\_task\_status." This way, a query on the ETL\_BATCH\_SOURCE\_IDENTIFIER column can find all of the loads that have ever originated from a particular data source; while a query on the ETL\_BATCH\_SOURCE column can help an administrator identify the file that was used to perform a particular batch load.

When the ETL process finishes its work, it will come back and populate the ETL\_BATCH\_FINISH\_TIME and ETL\_BATCH\_ELAPSED\_SECONDS columns in this table. These columns can be used by warehouse administrators to monitor the performance of various ETL processes.

## Dimension: Organization

The Organization dimension is used to record information about various distinct organizational units, who are all contributing data to a single warehouse database.

Column	Type	Description
<b>ORGANIZATION_KEY</b>	Integer	A uniquely assigned surrogate key for this organization
<b>ORGANIZATION_NAME</b>	String	Descriptive name for an organization
<b>ORGANIZATION_PARENT_KEY</b>	Integer	The unique key of this organization's parent org
<b>ROW_CREATED_BY_KEY</b>	Integer	Key for the ETL batch run that created this row
<b>ROW_LAST_UPDATED_BY_KEY</b>	Integer	Key for the ETL batch run that last updated this row

In future iterations, the Organization dimension may expand to contain additional columns representing meaningful attributes identified by stakeholders. But for now, the Organization dimension will remain extremely simple, in keeping with the “minimal data” goal of the initial iteration.

The Organization dimension will be managed as a Type 1 SCD. This decision reflects an assumption that organizational structure will be useful for configuring privileges (such as constraining the set of data that a particular user can view), and that a record of historical changes to the org chart is not critical for this task.

Since organizations can form a “ragged” hierarchy of arbitrary depth, a second bridge table will be provided to simplify the act of querying data from all of the organizations that fall under a particular branch of the org chart.

Every warehouse will include a single global organization which acts as the “root” of the org structure.

## Dimension: Team

The Team dimension is used to record information about various teams of individuals who are working together on a project.

Column	Type	Description
<b>TEAM_KEY</b>	Integer	A uniquely assigned surrogate key for this team
<b>TEAM_NAME</b>	String	Descriptive name for a team
<b>ROW_CREATED_BY_KEY</b>	Integer	Key for the ETL batch run that created this row
<b>ROW_LAST_UPDATED_BY_KEY</b>	Integer	Key for the ETL batch run that last updated this row

In future iterations, the Team dimension could expand to contain additional columns for subteam, discipline (e.g. Dev vs QA), and other key attributes. But for now, the Team dimension will remain extremely simple, in keeping with the “minimal data” goal of the initial iteration.

For now, the Team dimension will be managed as a Type 1 SCD. This decision may be revisited in future iterations as additional attributes are introduced.

## Dimension: Person

The Person dimension is used to record information about individuals who have participated on a team project, and whose data has been loaded into the warehouse.

Column	Type	Description
<b>PERSON_KEY</b>	Integer	A uniquely assigned surrogate key for this person
<b>PERSON_NAME</b>	String	The name of the individual, typically in encrypted form to protect data privacy
<b>ROW_CREATED_BY_KEY</b>	Integer	Key for the ETL batch run that created this row
<b>ROW_LAST_UPDATED_BY_KEY</b>	Integer	Key for the ETL batch run that last updated this row

Personal data privacy is a crucial concern of the data warehouse, so in most cases the name of the individual will be encrypted. This encryption could potentially be disabled in very small team/personal databases where privacy is not a concern.

Regardless of encryption, the PERSON\_KEY can be used as an opaque key for an individual. This allows queries to count the number of distinct individuals who participated on a project, peer review, etc.

## Dimension: Project

The Project dimension keeps track of a defined project effort performed by a team.

Column	Type	Description
<b>PROJECT_KEY</b>	Integer	A uniquely assigned surrogate key for this project
<b>PROJECT_NAME</b>	String	Descriptive name for the project
<b>ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above

In future iterations, the Project dimension will expand to contain additional columns for subproject, iteration, start date, and other key attributes. But for now, the Project dimension will remain extremely simple, in keeping with the “minimal data” goal of the initial iteration.

For now, the Project dimension will be managed as a Type 1 SCD. This decision may be revisited in future iterations as additional attributes are introduced.

## Dimension: EV Schedule

Earned value is a central concept for TPI planning and tracking. On a given project, each individual will have a personal earned value schedule. In addition, the team will have at least one earned value rollup that they use to monitor team progress. The EV Schedule dimension is used to record these entities.

Column	Type	Description
<b>EV_SCHEDULE_KEY</b>	Integer	A uniquely assigned surrogate key for this schedule
<b>EV_SCHEDULE_NAME</b>	String	A short, human-readable name for this schedule
<b>EV_SCHEDULE_IDENTIFIER</b>	String	An identifier, possibly assigned by the source system, for this schedule
<b>EV_SCHEDULE_ROLLUP_FLAG</b>	Boolean	True if this schedule represents a team rollup, false if it represents a personal schedule

Although it is common for there to be a one-to-one mapping between projects and earned value schedules, some tools do not require this. When multiple EV schedules/rollups can exist for a given project, the entries in this dimension become very important. Tools that enforce a 1-to-1 mapping will generally create an automatic entry in this table for each individual, and an entry for the team rollup.

If the EV\_SCHEDULE\_NAME for a personal schedule would reveal the identity of the associated individual, this name may need to be encrypted to protect data privacy.

A second bridge table will be provided to simplify the act of querying data from all of the personal schedules that are included in a particular rollup.

## Dimension: WBS Element

The concept of a hierarchical Work Breakdown Structure is central to virtually every TSP/TPI tool and every project plan. Within a particular plan, the WBS hierarchy is typically described in terms of Elements, which describe the decomposition of a project plan into successively smaller parts. The names of these parts are captured in the WBS Element dimension.

Column	Type	Description
<b>WBS_ELEMENT_KEY</b>	Integer	A uniquely assigned key for this WBS element name
<b>PARENT_WBS_ELEMENT_KEY</b>	Integer	The WBS_ELEMENT_KEY of this element's logical parent
<b>WBS_ELEMENT_NAME</b>	String	The full path name of this WBS element, formed by joining all path segments together with "/" as a delimiter.
<b>WBS_ELEMENT_NAME_LEN</b>	Integer	The number of name segments in this WBS element path
<b>WBS_ELEMENT_NAME_1</b>	String	First name segment in the WBS element path
<b>WBS_ELEMENT_NAME_2</b>	String	Second name segment in the WBS element path
<b>WBS_ELEMENT_NAME_N</b>	String	...etc. Unused fields have the value "-"
<b>WBS_ELEMENT_LAST_1</b>	String	Final name segment in the WBS element path
<b>WBS_ELEMENT_LAST_2</b>	String	Second-to-last name segment in the WBS element path
<b>WBS_ELEMENT_LAST_N</b>	String	...etc. Unused fields have the value "-"

For maximum flexibility when querying, WBS element names are stored in three ways:

- The full path of a WBS element will be stored in the WBS\_ELEMENT\_NAME field, in a format that is reminiscent of the path names on a UNIX file system. This full path column can be useful as a label on reports, and may also be useful in queries that use SQL pattern-matching operators.
- The hierarchical components of a path (e.g. parent / child / grandchild) will be split out and stored in the fields WBS\_ELEMENT\_NAME\_1, WBS\_ELEMENT\_NAME\_2, etc. This follows the typical pattern for a hierarchical data warehouse dimension.
- These same values will be stored in the columns WBS\_ELEMENT\_LAST\_1, WBS\_ELEMENT\_LAST\_2, etc. The primary difference is that while the WBS\_ELEMENT\_NAME\_# columns are left-aligned, the WBS\_ELEMENT\_LAST\_# columns will be right-aligned. (These columns will make it possible to find a component like "Component ABC/Subcomponent XYZ" across several projects, even if this component has been given a different parent in the various projects.)

The exact number of columns (the number "N" in the table above) will be fixed. Stakeholder input will be helpful to select a value that is high enough to capture the most complex project plan that will be encountered in practical use.

- If "N" is large enough, it should be extremely rare for a WBS element to use every column; in that case, the value "-" will be written into the higher-numbered columns.
- In the unlikely event that a particular project plan requires more detail than anticipated, the final column (WBS\_ELEMENT\_NAME\_N or WBS\_ELEMENT\_LAST\_N) will include a slash-delimited path that collects all of the remaining path segments. As a result, the data warehouse will still be able to capture the most detailed plan; but drill-down ability will be limited for WBS depths greater than "N."



When querying data, it will be extremely common to request data for a particular WBS element and all of its descendants. To support this, a second bridge table will be created to capture hierarchical relationships within the WBS.

A typical project plan will contain both WBS elements, and tasks organized underneath those elements. The WBS Element dimension only describes the portion of the hierarchy above the task level. Task details are captured separately in the Task dimension, described below.

The WBS Element dimension is not a slowly changing dimension (SCD). Entries in the WBS Element dimension are never modified or deleted, because they only represent the abstract names of WBS elements. These element names might or might not map to entries in a particular project plan; such information is recorded in the Plan Item dimension, described below.

### Dimension: Task

A TPI project plan includes a list of tasks that must be performed. The names of these tasks are captured in the Task dimension.

Column	Type	Description
<b>PROJECT_KEY</b>	Integer	A uniquely assigned surrogate key for this task name
<b>TASK_NAME</b>	String	Descriptive name for the task

Within a hierarchical plan, tasks always appear underneath a WBS element. The Task dimension will only capture the portion of the task name that appears underneath the WBS element.

Most TSP/TPI tools support the concept of a repeatable process which is used over and over again underneath multiple WBS elements. Since the Task dimension only includes the portion of the task name that appears underneath the WBS element, this TASK\_NAME field will quite regularly capture the short list of task names from a repeatable process. However, some TSP/TPI tools allow repeatable processes to have a hierarchical structure of their own. In that case, the Process Enactment dimension will be the most accurate representation of the repeatable process phases that were used to create each element in the project plan.

## Dimension: Process

Repeatable, defined processes are a fundamental building block of TSP-style planning and analysis. Thus, it will be important to capture process definitions in the warehouse, and to tag metrics with the corresponding process metadata.

Good process definitions evolve over time. In addition, process definitions are often tailored to meet the needs of a particular team or organization. Accordingly, it will be important for the schema to support these needs. The resulting schema is shown below.



First, a table will be created to contain the names of the processes that have been defined:

Column	Type	Description
PROCESS_KEY	Integer	A uniquely assigned surrogate key for a process
PROCESS_NAME	String	Name for the process definition
PROCESS_VERSION	String	Version number for the process definition
PROCESS_IDENTIFIER	String	An alphanumeric identifier for this process/version
ROW_CREATED_BY_KEY, ROW_LAST_UPDATED_BY_KEY		...same as above

Next, a table will be created to describe the phases in these processes:

Column	Type	Description
PHASE_KEY	Integer	A uniquely assigned surrogate key for a process phase
PROCESS_KEY	Integer	The key for the process to which this phase belongs
PHASE_NAME	String	Descriptive name for the phase (potentially better for reports)
PHASE_SHORT_NAME	String	Short name for the phase; may be an abbreviation
PHASE_TYPE	String	One of "Overhead," "Construction," "Appraisal" or "Failure"
PHASE_ORDINAL	Integer	A number indicating the relative order of this phase within the process. (This is important for reporting purposes, and also for calculations such as Yield.) This may be null for an older phase that has been removed from the process.
PHASE_IDENTIFIER	String	An alphanumeric identifier for this phase
ROW_CREATED_BY_KEY, ROW_LAST_UPDATED_BY_KEY		...same as above

Finally, a table will be created to document equivalency relationships between phases in two different process definitions:

Column	Type	Description
PHASE_KEY	Integer	The key for a process phase
MAPS_TO_PHASE_KEY	Integer	The key of an equivalent phase in another process

These tables will be Type 1 SCDs. The data in the tables will capture the most current known data about each process.

The phase mapping dimension is used to capture a number of important process concepts:

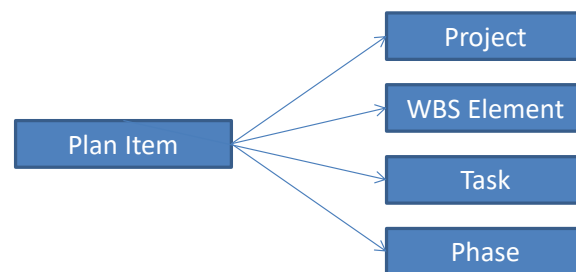
- When a team creates a tailored version of an organizational process, this would be represented by two separate entries in the Process table, and two separate lists of Phases. Rows would be created in the Phase Mapping table to describe how each phase in the tailored process maps back to a phase in the original process.
- When a team makes significant changes to their process, they can do so by “archiving” the original process definition and creating a new process. Phase mapping rows could then describe equivalency relationships between the phases in the new process and the phases in the original process. In this way, teams can capture information about process evolution, yet still include historical data in current analyses.
- Organizations may wish to aggregate data from many different teams of different disciplines, then analyze that data using a common framework. For example, it could be valuable to analyze different types of rework from every team in an organization, regardless of whether those teams write software or produce training materials. As another example, a research organization like the SEI might wish to map many different organizational processes back to a common framework like TSP. Phase Mapping rows could be created to map each team process back to the common framework.

## Dimension: Plan Item

A team project can contain many components, tasks, and milestones that are arranged hierarchically. The Plan Item dimension records these objects, and documents how they change and evolve over time.

Column	Type	Description	
PLAN_ITEM_KEY	Integer	A uniquely assigned surrogate key for this item	
PARENT_PLAN_ITEM_KEY	Integer	The unique key of the plan item which is this row's parent	
PROJECT_KEY	Integer	The unique key of a team project	
WBS_ELEMENT_KEY	Integer	The unique key of a WBS element name	
TASK_KEY	Integer	The unique key of a task name.	Null if this plan item represents a WBS element.
PHASE_KEY	Integer	The unique key of the process phase for this task.	
PLAN_ITEM_IDENTIFIER	String	A string, assigned by the source system, which uniquely identifies this item within a particular project plan	
PLAN_ITEM_ORDINAL	Number	A number indicating the relative position of this plan item within its siblings	
PLAN_ITEM_LEAF_ELEMENT_FLAG	Boolean	True if this plan item represents a WBS element with no sub-elements	
PLAN_ITEM_LEAF_TASK_FLAG	Boolean	True if this plan item represents a task with no subtasks	
PLAN_ITEM_DELETED_FLAG	Boolean	True if this item has been deleted from the plan	
ROW_CREATED/LAST_UPDATED_BY_KEY		...same as above	

These relationships are depicted visually below:



WBS Elements appear in the Plan Item dimension as rows with no TASK\_KEY or PHASE\_KEY. Project tasks appear as rows with an appropriate TASK\_KEY and PHASE\_KEY. Milestone objects appear in the Plan Item dimension as rows whose TASK\_KEY points to the milestone name, and whose PHASE\_KEY points to a special “Milestone” phase.

The Plan Item dimension will be managed as a Type 4 SCD. Thus, the table shown above will record the most current values for a particular item within a team plan. A second table will be created to track the evolution of these values over time.

## Plan Item Dimension Example

Then Plan Item dimension is central to the design of the data warehouse, so it is important to thoroughly understand how this dimension captures the information in a project plan. To that end, an example is helpful to illustrate the use of the Plan Item dimension and its relationships with the Project, WBS Element, Task, and Phase dimensions. So consider the hypothetical project plan depicted at right. This plan might be captured in the data warehouse in the following way:

### Representation in the Project table:

PROJECT_KEY	PROJECT_NAME
101	Project XYZ

### Abbreviated representation in the WBS Element table (excluding numbered columns):

WBS_ELEMENT_KEY	WBS_ELEMENT_NAME
200	<Root Element>
201	Component A
202	Component B
203	Component C
204	Component C/Subcomponent C1
205	Component C/Subcomponent C2

### Representation in the Task table:
















TASK_KEY	TASK_NAME
301	Work
302	Code
303	Test
304	Postmortem

### Representation in the Phase table:

PHASE_KEY	PHASE_SHORT_NAME	PHASE_NAME
401	MISC	Miscellaneous
402	CODE	Code
403	UT	Unit Test
404	PM	Postmortem

### Representation in the Plan Item table:

PLAN_ITEM_KEY	PROJECT_KEY	WBS_ELEMENT_KEY	TASK_KEY	PHASE_KEY
1	101	200	-	-
2	101	200	301	401
3	101	201	-	-
4	101	201	302	402
5	101	201	303	403
6	101	201	304	404
7	101	202	-	-
8	101	202	302	402
9	101	202	303	403
10	101	202	304	404
11	101	203	-	-
12	101	204	-	-
13	101	204	301	401
14	101	205	-	-
15	101	205	301	401

1	 Project XYZ
2	 Work
3	 Component A
4	 Code
5	 Test
6	 Postmortem
7	 Component B
8	 Code
9	 Test
10	 Postmortem
11	 Component C
12	 Subcomponent C1
13	 Work
14	 Subcomponent C2
15	 Work

Continuing with this hypothetical project example, it is also helpful to demonstrate the manner in which names are stored in the WBS Element dimension. The table below shows how the WBS elements in this project would be stored into the numbered columns, assuming N = 3:

WBS_ELEM_KEY	WBS_ELEMENT_NAME	ELEM_NAME_1	ELEM_NAME_2	ELEM_NAME_3	ELEM_LAST_3	ELEM_LAST_2	ELEM_LAST_1
200	<Root Element>	-	-	-	-	-	-
201	Component A	Component A	-	-	-	-	Component A
202	Component B	Component B	-	-	-	-	Component B
203	Component C	Component C	-	-	-	-	Component C
204	Component C/Subcomponent C1	Component C	Subcomponent C1	-	-	Component C	Subcomponent C1
205	Component C/Subcomponent C2	Component C	Subcomponent C2	-	-	Component C	Subcomponent C2

The plain WBS\_ELEMENT\_NAME column provides the full path name of a particular component, as it might be displayed in various reports.

The numbered columns are provided to support specific types of COTS query tools. To appreciate how these columns can be used, consider these SQL queries:

Example query	Interpretation
SELECT ... WHERE WBS_ELEMENT_NAME_1 = "Component C"	<i>Find data in the warehouse associated with Component C and all of its subcomponents</i>
SELECT ... WHERE WBS_ELEMENT_NAME_1 = "Component C" GROUP BY WBS_ELEMENT_NAME_2	<i>Create a pivot table that shows detail for each of the subcomponents underneath Component C</i>

These columns make it simple to perform many very common drill-down analyses without resorting to stored procedures or database-specific extensions. Relying only on "=" clauses and GROUP BY constructs allows extremely efficient analysis of massive amounts of data by leveraging prebuilt database table indexes.

The WBS\_ELEMENT\_LAST\_\* columns are designed to support queries that find a particular component in multiple different project plans, even if the component had a different set of ancestors in the various plans. For example, if a team has several project iterations, and they create a new plan for each iteration, they could potentially encounter the following pattern:

- In Iteration 1, they have a WBS Element in their plan called "Component A"
- In Iteration 2, they have a WBS Element in their plan called "Iteration 1 Cleanup/Component A"
- In Iteration 3, they have a WBS Element in their plan called "Iteration 1 Rework/Component A"

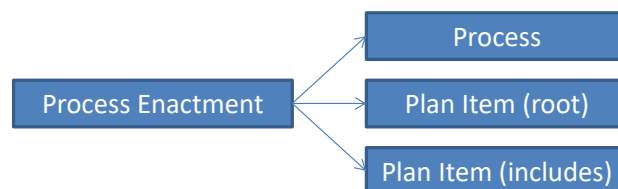
An SQL query that tests WBS\_ELEMENT\_LAST\_1 = "Component A" would allow the team to roll up Component A data from all three iterations.

## Dimension: Process Enactment

Repeatable processes form the basis for a number of common analyses in high-maturity projects. To perform these analyses, it is important to identify the various places in a project plan where a particular process was used to perform work. The Process Enactment dimension captures this information.

Column	Type	Description
<b>PROCESS_ENACTMENT_KEY</b>	Integer	A uniquely assigned surrogate key for this row
<b>PROCESS_KEY</b>	Integer	The unique key of a process that was used to perform work
<b>ROOT_PLAN_ITEM_KEY</b>	Integer	The unique key of a parent item in a project plan where this process was applied/instantiated to create subtasks
<b>INCLUDES_PLAN_ITEM_KEY</b>	Integer	The unique key of an item in the project plan that is a part of this enactment of the given process: <ul style="list-style-type: none"><li>• The project plan will include tasks corresponding to each step in the process. A row will be created in this table corresponding to each of those tasks.</li><li>• The “root” plan item is also a part of the enactment of this process, since metrics like size could be recorded there. Accordingly, a row will always be created in this table including the root.</li></ul>
<b>ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above

These relationships are depicted visually below:



Queries can search for distinct [Process / Root] pairs to find the various instances where a particular process was used to perform work. Such an instance is called an “enactment” of the process. These distinct instances could become individual data points in a regression analysis or a trend chart.

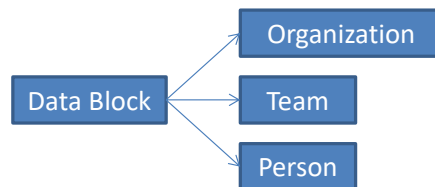
The “includes” plan item can be joined to another fact table to summarize/aggregate data for a particular instance (or collection of instances). This could be used to identify process enactments that are 100% complete, to sum up their size and time data, and to perform other analyses.

## Dimension: Data Block

Many metrics are collected during the course of a high maturity team project. For reporting and analysis purposes, it is important to track the origin of each metric. To facilitate this, the warehouse will include the Data Block dimension. A Data Block is an abstract collection of metrics and data associated with a particular individual working within a particular organization as part of a particular project team.

Column	Type	Description
<b>DATA_BLOCK_KEY</b>	Integer	A uniquely assigned surrogate key for this data block
<b>ORGANIZATION_KEY</b>	Integer	The unique key of an organization
<b>TEAM_KEY</b>	Integer	The unique key of a team
<b>PERSON_KEY</b>	Integer	The unique key of an individual
<b>ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above

These relationships are depicted visually below:



The ORGANIZATION\_KEY, TEAM\_KEY, and PERSON\_KEY fields will reference entities from the associated dimensions.



## Dimension: Size Metric

Many different metrics can be used to measure size; the Size Metric dimension records these.

Column	Type	Description
SIZE_METRIC_KEY	Integer	A uniquely assigned surrogate key for a size metric
SIZE_METRIC_NAME	String	The name of this size metric
SIZE_METRIC_SHORT_NAME	String	Short name for this metric; may be an abbreviation
ROW_CREATED/LAST_UPDATED_BY_KEY		...same as above

In future iterations, the Size Metric dimension will expand to contain additional columns representing meaningful attributes identified by stakeholders. But for now, the Size dimension will remain extremely simple, in keeping with the “minimal data” goal of the initial iteration.

## Dimension: EV Metric

Various metrics can be recorded for an EV schedule; the EV Metric dimension records these.

Column	Type	Description
EV_METRIC_KEY	Integer	A uniquely assigned surrogate key for an EV metric
EV_METRIC_NAME	String	The name of this EV metric
EV_METRIC_IDENTIFIER	String	A unique identifier for this metric; e.g. “BCWP”
EV_METRIC_UNITS	String	The unit of measure for this metric; e.g. “minutes”
EV_METRIC_AGGR_FUNCTION	String	The SQL function that might be used to aggregate metrics of this type; e.g. “sum” or “max”

## Dimension: Defect Type

Defects have a type that comes from an associated Defect Type Standard. A dimension is provided to manage these values.

Column	Type	Description
DEFECT_TYPE_KEY	Integer	A uniquely assigned surrogate key for this defect type
DEFECT_TYPE_STANDARD_NAME	String	The name of the defect type standard to which this defect type belongs
DEFECT_TYPE_NAME	String	The name of one defect type within this standard
DEFECT_TYPE_DESCRIPTION	String	A longer description of this defect type
ROW_CREATED_BY_KEY, ROW_LAST_UPDATED_BY_KEY		...same as above

## Dimension: Measurement Type

When calculating measurements, we often see examples of planned vs actual data. But these are only two possible qualifiers out of many. For example, we also see examples of replan and forecast data. The Measurement Type dimension is provided to flexibly record different types of observations that are collected or generated by various TSP/TPI tools.

Column	Type	Description
<b>MEASUREMENT_TYPE_KEY</b>	Integer	A uniquely assigned surrogate key for this type
<b>MEASUREMENT_TYPE_NAME</b>	String	The name for a measurement type

## Dimension: Dependency Type

Many tools provide the capability to describe dependencies between the various items in a plan. A dimension captures the various types of dependencies:

Column	Type	Description
<b>DEPENDENCY_TYPE_KEY</b>	Integer	A uniquely assigned surrogate key for this type
<b>DEPENDENCY_TYPE_NAME</b>	String	The name for a dependency type
<b>DEPENDENCY_TYPE_ABBR</b>	String	An abbreviation for this dependency type

This table will always contain exactly four rows, corresponding to the four standard dependency types:

- Finish-to-Start (FS)
- Finish-to-Finish (FF)
- Start-to-Start (SS)
- Start-to-Finish (SF)

is table will always c

## Dimension: Attributes

The data warehouse will be capable of aggregating data that originates from a number of different tools and source systems. These data sources will contain a variety of textual data attributes that are not directly captured in the other warehouse tables. To support the storage and analysis of those attributes, a pair of dimensional tables will be provided.

The first table will capture the names and unique identifiers for the different types of attributes:

Column	Type	Description
<b>ATTRIBUTE_KEY</b>	Integer	A uniquely assigned surrogate key for this attribute
<b>ATTRIBUTE_IDENTIFIER</b>	String	A unique identifier for the attribute
<b>ATTRIBUTE_NAME</b>	String	A human-readable description for this attribute

Textual attribute values will be stored in a separate table:

Column	Type	Description
<b>ATTRIBUTE_VALUE_KEY</b>	Integer	A uniquely assigned surrogate key for this attribute value
<b>ATTRIBUTE_KEY</b>	Integer	A key into the previous table, describing which type of attribute this value represents
<b>ATTRIBUTE_VALUE_TEXT</b>	String	A textual value stored for this attribute

In the future, this model could possibly be extended to allow for the storage of attribute values which are numbers, timestamps, or other primitive types. But in keeping with the simple goals of the initial iteration, attribute values will only be stored as text.

Finally, fact tables (described in the next section) will be used to capture the association of attribute values to various entities in the warehouse.

## Dimension: Text

Many facts have associated comments or descriptions. The Text dimension provides a common, reusable mechanism for storing these free-text values.

Column	Type	Description
<b>TEXT_KEY</b>	Integer	A uniquely assigned surrogate key for this text item
<b>TEXT_VALUE</b>	String	An arbitrary block of free text, which could be of significant length

## Dimension: Date

Date values appear often in the metrics that are collected and analyzed by teams. The Date dimension provides an efficient means of representing and rapidly analyzing these values.

Column	Type	Description
<b>DATE_KEY</b>	Integer	An integer key representing a particular date
<b>FULL_DATE</b>	Date	The full representation of this date stored as a native database Date object
<b>DATE_NAME</b>	String	A human readable description of this date in the form YYYY-MM-DD
<b>DATE_NAME_US</b>	String	A human-readable description of this date in the form M/D/YY
<b>DATE_NAME_EU</b>	String	A human-readable description of this date in the form D/M/YY
<b>DAY_OF_WEEK</b>	Integer	1 for Sunday, 2 for Monday, 7 for Saturday
<b>DAY_NAME_OF_WEEK</b>	String	A day name like "Sunday," "Monday," etc.
<b>WEEKDAY_WEEKEND</b>	String	The text "Weekday" if this is a day from Monday to Friday, "Weekend" otherwise
<b>DAY_OF_MONTH</b>	Integer	The calendar day of the month, from 1 to 31
<b>LAST_DAY_OF_MONTH_FLAG</b>	Boolean	True if this date is the last day of the month
<b>MONTH_OF_YEAR</b>	Integer	1 for January, 12 for December
<b>MONTH_NAME</b>	String	"January", "February", etc.
<b>CALENDAR_YEAR</b>	Integer	The year, for example 2013
<b>DATE_SEQ</b>	Integer	The number of whole days elapsed since Nov 17, 1858 (i.e., Modified Julian Date). This value enables fast date arithmetic via addition/subtraction

Even though dates can be stored natively by the underlying database engine, it is a data warehousing best practice to augment those timestamps with a dimension such as this one. This provides several benefits:

- "Group by" queries can make use of database indexes to efficiently slice and dice data in other tables by day/week/month/year without the overhead of row-at-a-time date arithmetic
- The Date dimension can include extra rows to hold special values like "Never" or "Unknown."

Within this dimension, 8-digit integer keys of the form YYYYMMDD will be assigned to individual dates. In addition, the following "special" date values (always > 99999000) will be added to the table:

Key	Date Name	Interpretation
<b>99999830</b>	Not Yet	A particular event has not yet occurred
<b>99999860</b>	Never	A particular event is never projected to happen
<b>99999880</b>	Cannot Calculate	A date calculation cannot be performed due to insufficient data
<b>99999930</b>	Unknown	A date was missing from the source system or otherwise not provided
<b>99999960</b>	Invalid	An invalid or corrupt date was read from the source system

## Dimension: Baseline

Baselines are a common concept in project planning. Baselines may be saved independently for various projects, and many tools allow the creation of multiple historical baselines. The baseline dimension provides a place to capture a list of the various baselines that exist:

Column	Type	Description
<b>BASELINE_KEY</b>	Integer	A uniquely assigned surrogate key for this baseline
<b>BASELINE_NAME</b>	String	A user-displayable name for this baseline
<b>BASELINE_IDENTIFIER</b>	String	A string, assigned by the source system, which uniquely identifies this baseline
<b>BASELINE_DESCRIPTION_TEXT_KEY</b>	Integer	A key into the Text dimension, pointing to a description for this baseline. (Optional)
<b>BASELINE_EFFECTIVE_DATE</b>	Timestamp	The date/time of the moment when the baseline was created/saved in the source system
<b>BASELINE_EFFECTIVE_DATE_KEY</b>	Integer	The baseline effective date as an index into the Date dimension
<b>BASELINE_TARGET_TYPE</b>	String	A string describing the type of entity this baseline is for. If the baseline was saved for an entity that has a representation in the warehouse (such as a Project, Team, or EvSchedule), this should be the simple name of the entity as seen in the Java-based data model. If the baseline was saved for an external entity that does not map exactly to a warehouse entity, this can be a custom value.
<b>BASELINE_TARGET_KEY</b>	Integer	If the baseline was saved for an entity that has a representation in the warehouse, this should be a key pointing to a row in the associated dimension. For example, if the target type is "Project", this would be the key of an item in the Project dimension. If the baseline was saved for an external entity, this may be null.
<b>BASELINE_ACTIVE_FLAG</b>	Boolean	True if this is the "active" baseline for the specified target. A given target can have only one active baseline.
<b>ROW_CREATED_BY_KEY, ROW_LAST_UPDATED_BY_KEY</b>		...same as above

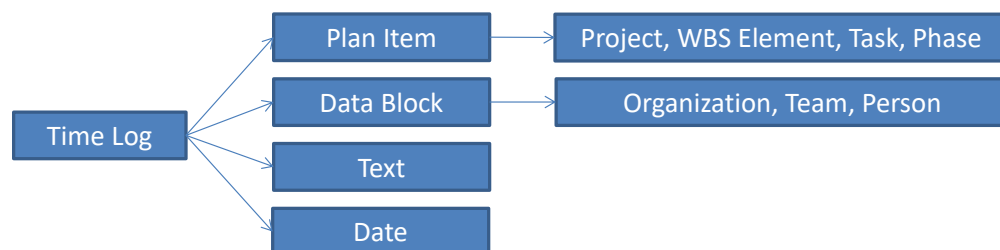
As stated above, the baseline dimension captures the list of the baselines that exist. The data actually associated with a baseline will be stored in one or fact tables (described below), with a **BASELINE\_KEY** pointing back to a row in this table.

## Fact: Time Log

The Time Log fact table records time log entries collected by individuals. The grain of the table is a single time log entry recorded by a particular individual.

Column	Type	Description
<b>TIME_LOG_FACT_KEY</b>	Integer	A unique ID for this table row
<b>PLAN_ITEM_KEY</b>	Integer	Key for the plan item this time log entry is attached to; conveys the project, WBS element, task, and process phase for this time log entry
<b>DATA_BLOCK_KEY</b>	Integer	Key for the data block this time log entry came from; conveys the organization, team, and individual this time log entry is associated with
<b>TIME_LOG_START_DATE</b>	Timestamp	The start date/time of this time log entry
<b>TIME_LOG_START_DATE_KEY</b>	Integer	The start date as an index into the Date dimension
<b>TIME_LOG_END_DATE</b>	Timestamp	The end date/time of this time log entry
<b>TIME_LOG_DELTA_MINUTES</b>	Number	The number of work minutes for this time log entry, calculated as total elapsed time minus interrupt time
<b>TIME_LOG_INTERRUPT_MINUTES</b>	Number	The number of minutes of interrupt time
<b>TIME_LOG_COMMENT_TEXT_KEY</b>	Integer	The key of a row in the Text dimension, holding the comment for this time log entry
<b>ROW_EFF_START_DATE</b>	Timestamp	The date/time this row became effective
<b>ROW_EFF_END_DATE</b>	Timestamp	The date and time when this row was replaced / deleted
<b>ROW_CURRENT_FLAG</b>	Boolean	True if this row represents current information, False if it has been replaced or deleted
<b>ROW_CREATED_BY_KEY, ROW_LAST_UPDATED_BY_KEY</b>		...same as above

These relationships are depicted visually below. (The ETL Audit dimension is omitted from the diagram for brevity.)



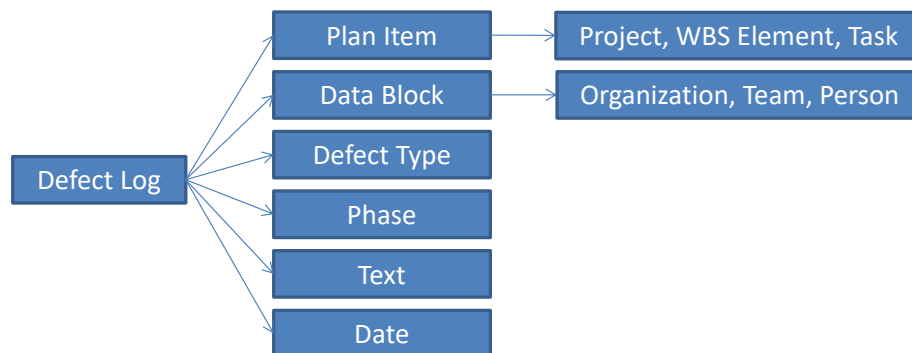
During the course of a project, an individual might edit or delete existing time log entries. (Generally, these changes represent data corrections.) The ROW\_EFF\_\* columns allow the fact table to capture the history of these changes. If a report wishes to analyze time log data as it appeared at some historical point in time, it can use a BETWEEN clause on these columns. To query or summarize current time log data, the ROW\_CURRENT\_FLAG column can be used to select the rows that are currently in effect.

## Fact: Defect Log

The Defect Log fact table records defect log entries collected by individuals. The grain of the table is a single defect recorded by a particular individual.

Column	Type	Description
<b>DEFECT_LOG_FACT_KEY</b>	Integer	A unique ID for this table row
<b>PLAN_ITEM_KEY</b>	Integer	Key of the plan item this defect is attached to
<b>DATA_BLOCK_KEY</b>	Integer	Key of the data block this defect came from
<b>DEFECT_IDENTIFIER</b>	String	ID for this defect, possibly from the source system
<b>DEFECT_FOUND_DATE</b>	Timestamp	The date this defect was found
<b>DEFECT_FOUND_DATE_KEY</b>	Integer	The date this defect was found, as an index into the Date dimension
<b>DEFECT_TYPE_KEY</b>	Integer	The unique key of a defect type from the Defect Type Standard dimension
<b>DEFECT_INJECTED_PHASE_KEY</b>	Integer	The key of a process phase where this defect was injected
<b>DEFECT_REMOVED_PHASE_KEY</b>	Integer	The key of a process phase where this defect was found or removed
<b>DEFECT_FIX_PENDING_FLAG</b>	Boolean	True if this defect has not been fixed yet
<b>DEFECT_FIX_TIME_MINUTES</b>	Number	The number of minutes of fix time
<b>DEFECT_FIX_COUNT</b>	Integer	The number of distinct defects represented by this entry (usually 1)
<b>DEFECT_FIX_DEFECT_IDENTIFIER</b>	String	The identifier of another defect that was being fixed when this defect was injected
<b>DEFECT_DESCRIPTION_TEXT_KEY</b>	Integer	Key of a description stored in the "Text" dimension
<b>ROW_EFF_START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED_BY_KEY, ROW_LAST_UPDATED_BY_KEY</b>		...same as above

These relationships are depicted visually below. (The ETL Audit dimension is omitted from the diagram for brevity.)



As with the time log, ROW\_EFF\_\* columns are provided to track the history of edits to a particular defect over time. These rows can be tied together with the DEFECT\_IDENTIFIER column.

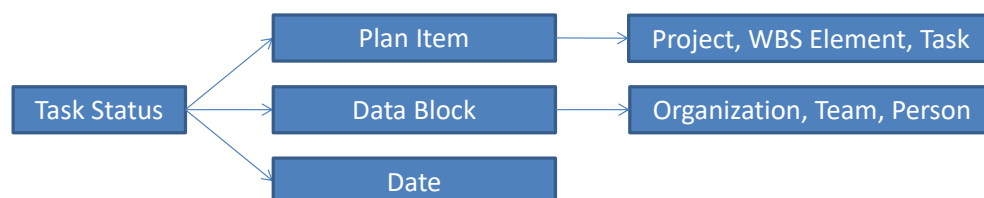
## Fact: Task Status

The Task Status table brings together several commonly used pieces of information about tasks. The granularity of this table is a row per leaf task per assigned individual. So each time an individual is assigned to a task, a row is created in this table. If multiple individuals are assigned to a task, a separate row is created for each person. Rows are only created for “leaf” tasks – they are not created for hierarchical parents like WBS elements. This ensures that the rows in this table represent non-overlapping measurements that can be grouped and summed in arbitrary ways.

Column	Type	Description
<b>TASK_STATUS_FACT_KEY</b>	Integer	A unique ID for this table row
<b>PLAN_ITEM_KEY</b>	Integer	Key of the plan item for this task
<b>DATA_BLOCK_KEY</b>	Integer	Key of the data block for the assigned individual
<b>TASK_PLAN_TIME_MINUTES</b>	Number	The amount of time this individual plans to spend on this task, in minutes
<b>TASK_ACTUAL_TIME_MINUTES</b>	Number	The actual amount of time this individual has spent on this task, in minutes
<b>TASK_ACTUAL_START_DATE</b>	Timestamp	The date this individual began working on this task; null if the task has not started
<b>TASK_ACTUAL_START_DATE_KEY</b>	Integer	The start date as an index into the Date dimension. If the task has not yet started, this will contain the special date “Not Yet”
<b>TASK_ACTUAL_COMPLETE_DATE</b>	Timestamp	The date this task was marked complete; null if the task is still incomplete
<b>TASK_ACTUAL_COMPLETE_DATE_KEY</b>	Integer	The completion date as an index into the Date dimension; “Not Yet” if the task is incomplete
<b>BASELINE_KEY</b>	Integer	A key into the Baseline dimension, if this fact represents a portion of a saved baseline.
<b>ROW_EFF_START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above

The columns in this table were selected because they form the basis for a large number of common reporting needs. Some columns could be calculated from other data (for example, actual time and actual start date could be retrieved from the time log), but they are summarized here for convenience.

Dates are stored as both native timestamps, and as foreign keys into the Date dimension. The native timestamps columns include time of day information, which may reflect wide time zone variances for geographically distributed teams. In contrast, the DATE\_KEY values will include the date portion only, expressed relative to the time zone of the assigned individual. Null native timestamps will translate to the special DATE\_KEY “Not Yet,” allowing the use of a SQL MIN and MAX functions to quickly summarize the team start/completion date of a particular Plan Item that was assigned to multiple individuals.





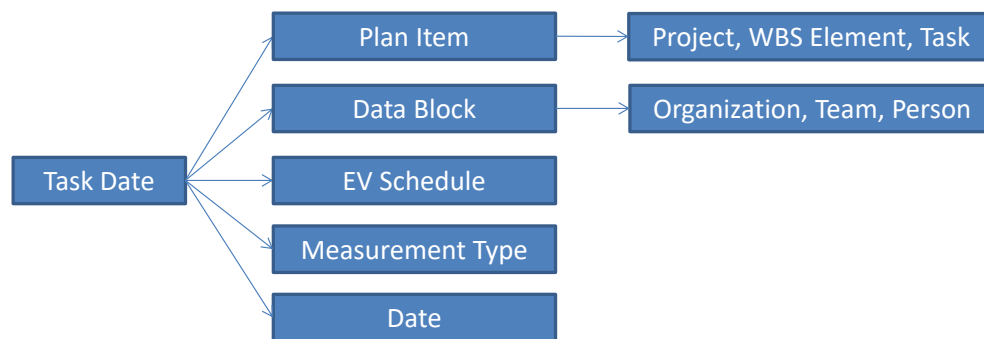
## Fact: Task Dates

The earned value planning and tracking activity calculates a number of useful dates for the tasks in a project plan. These dates are captured in the Task Dates table. The grain of this table is one row per assigned individual, per “leaf task” in a personal EV schedule, per applicable measurement type.

Column	Type	Description
<b>TASK_DATE_FACT_KEY</b>	Integer	A unique ID for this table row
<b>PLAN_ITEM_KEY</b>	Integer	Key of the plan item for this task
<b>DATA_BLOCK_KEY</b>	Integer	Key of the data block for the assigned individual
<b>EV_SCHEDULE_KEY</b>	Integer	Key of the personal EV schedule that is responsible for this date
<b>MEASUREMENT_TYPE_KEY</b>	Integer	Key indicating the type of date this row represents (e.g. plan, actual, forecast, etc.)
<b>TASK_DATE_KEY</b>	Integer	The completion date of the task as an index into the Date dimension.
<b>BASELINE_KEY</b>	Integer	A key into the Baseline dimension, if this fact represents a portion of a saved baseline.
<b>ROW_EFF_START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above

At a minimum, tools should create rows in this table for “Plan” and “Actual” completion dates. Additional rows can be created for other types of dates (for example, to represent different types of replan/forecast calculations).

If a task has not been completed yet, its “Actual” row will still be included in this table with the DATE\_KEY for “Not Yet.” If a particular calculation believes that a task is never projected to finish, the row will be created in this table with the DATE\_KEY for “Never.” This convention will allow the SQL MAX function to summarize team completion dates for various measurement types.

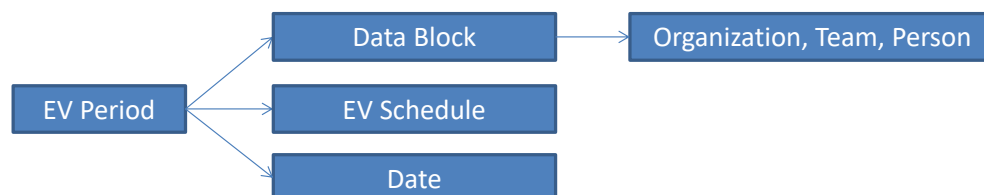


## Fact: EV Schedule Periods

When planning and tracking work with earned value, the overall calendar is broken into time periods (generally a week long). Then time and value is tracked against those time periods. In the warehouse, this information is recorded into the EV Schedule Period fact table. The grain of this table is one row per assigned individual, per EV schedule, per time period.

Column	Type	Description
<b>EV_SCHEDULE_PERIOD_FACT_KEY</b>	Integer	A unique ID for this table row
<b>EV_SCHEDULE_KEY</b>	Integer	Key of the personal EV schedule that contains this schedule period
<b>DATA_BLOCK_KEY</b>	Integer	Key of the data block for the individual who owns that schedule
<b>PERIOD_START_DATE</b>	Date	Start date/time for this calendar period
<b>PERIOD_START_DATE_KEY</b>	Integer	Start date as an index into the Date dimension
<b>PERIOD_END_DATE</b>	Date	End date/time for this calendar period
<b>PERIOD_END_DATE_KEY</b>	Integer	End date as an index into the Date dimension
<b>PLAN_TIME_MINUTES</b>	Number	The number of minutes of time this individual planned to work during this calendar period
<b>CUM_PLAN_TIME_MINUTES</b>	Number	A running total of planned time
<b>ACTUAL_TIME_MINUTES</b>	Number	The number of minutes of time this individual actually worked during this calendar period
<b>CUM_ACTUAL_TIME_MINUTES</b>	Number	A running total of actual time
<b>PLAN_VALUE_MINUTES</b>	Number	The amount of value this individual planned to earn in this calendar period, expressed in minutes
<b>CUM_PLAN_VALUE_MINUTES</b>	Number	A running total of planned value
<b>EARNED_VALUE_MINUTES</b>	Number	The amount of value this individual actually earned during this calendar period, expressed in minutes
<b>CUM_EARNED_VALUE_MINUTES</b>	Number	A running total of earned value
<b>ACTUAL_COST_MINUTES</b>	Number	The total number of minutes of actual time that were spent on tasks that were marked complete during this calendar period
<b>CUM_ACTUAL_COST_MINUTES</b>	Number	A running total of actual cost
<b>BASELINE_KEY</b>	Integer	A key into the Baseline dimension, if this fact represents a portion of a saved baseline.
<b>ROW EFF START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above

This fact table will only contain data for personal EV schedules. A secondary view will be provided that rolls up data from all of the schedules for a team. For convenience, that secondary view will also provide columns that normalize planned and actual EV to percentage points.



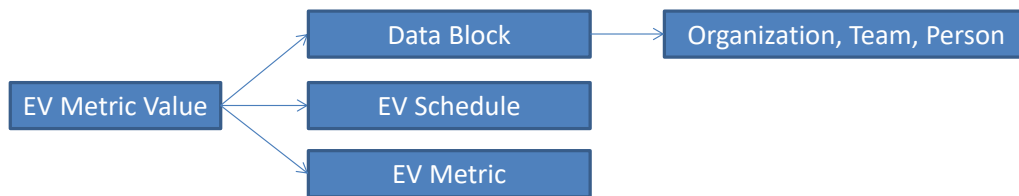
## Fact: EV Metric Values

Once data has been collected against an EV schedule, a great number of useful metrics can be calculated.

The most common set of values will be numeric. These are recorded in an “EV Metric Number” fact table. The grain of this table is one row per individual, per EV schedule, per EV Metric.

Column	Type	Description
<b>EV_METRIC_NUMBER_FACT_KEY</b>	Integer	A unique ID for this table row
<b>EV_SCHEDULE_KEY</b>	Integer	Key of a personal EV schedule
<b>DATA_BLOCK_KEY</b>	Integer	Key of the data block for the individual who owns that schedule
<b>EV_METRIC_KEY</b>	Integer	Key of an EV Metric
<b>VALUE</b>	Number	The numeric value of that EV Metric for this schedule for this individual
<b>BASELINE_KEY</b>	Integer	A key into the Baseline dimension, if this fact represents a portion of a saved baseline.
<b>ROW_EFF_START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above

Initially, this table will include rows to capture the values of the standard EV metrics BCWS, BCWP, ACWP, and BAC. Additional EV Metrics will be added in future releases.

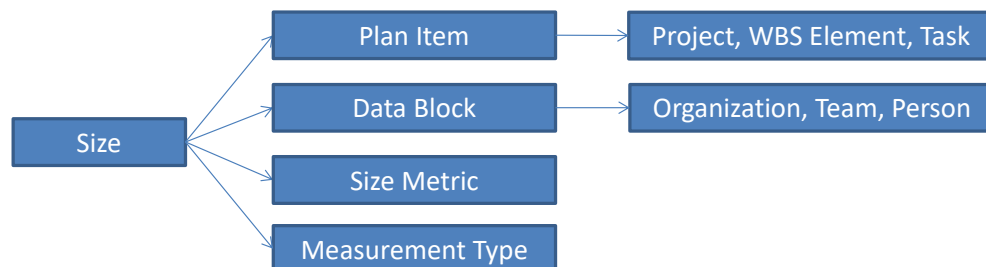


## Fact: Size

The Size fact table records information about the planned and actual sizes of work products.

Column	Type	Description
SIZE_FACT_KEY	Integer	A unique ID for this table row
PLAN_ITEM_KEY	Integer	Key for the plan item that this size measurement is associated with
DATA_BLOCK_KEY	Integer	Key for the data block for the individual who recorded this size measurement
SIZE_METRIC_KEY	Integer	Key of the size metric for this table row
MEASUREMENT_TYPE_KEY	Integer	Indicator of whether this row represents a planned or actual size
SIZE_BASE	Number	Standard size accounting metrics
SIZE_ADDED	Number	"
SIZE_DELETED	Number	"
SIZE_MODIFIED	Number	"
SIZE_REUSED	Number	"
SIZE_TOTAL	Number	"
SIZE_ADDED_AND_MODIFIED	Number	A&M size, precalculated for ease of use in queries, analyses, and reports
BASELINE_KEY	Integer	A key into the Baseline dimension, if this fact represents a portion of a saved baseline.
ROW Eff_Start/End Date, Row_Current_Flag, Row_Created/Last_Updated_By_Key		...same as above

These relationships are depicted visually below. (The ETL Audit dimension is omitted from the diagram for brevity.)



Future iterations of the warehouse may include additional data, such as the name of specific work products that were created. But for now, the Size fact table will remain extremely simple, in keeping with the “minimal data” goal of the initial iteration.

### Fact: Plan Item Attributes

Tools commonly allow plan items to be annotated with keywords, labels, system IDs, and other types of data. The Plan Item Attribute table will make it possible to capture this information about the plan.

Column	Type	Description
<b>PLAN_ITEM_ATTR_FACT_KEY</b>	Integer	A unique ID for this table row
<b>PLAN_ITEM_KEY</b>	Integer	Key for the plan item that this attribute value is associated with
<b>ATTRIBUTE_KEY</b>	Integer	Key of the entry in the Attribute dimension describing the type of this value.
<b>ATTRIBUTE_VALUE_KEY</b>	Integer	Key for an entry in the Attribute Value table
<b>ROW_EFF_START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above



### Fact: Plan Item Notes

Some tools allow free-text notes to be attached to the items in a project plan. The Plan Item Note table captures this information.

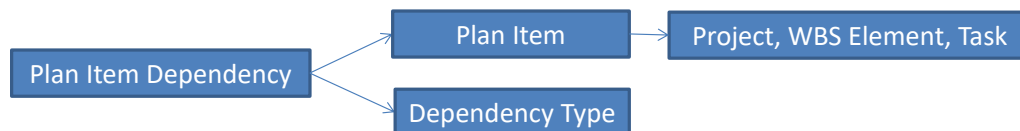
Column	Type	Description
<b>PLAN_ITEM_NOTE_FACT_KEY</b>	Integer	A unique ID for this table row
<b>PLAN_ITEM_KEY</b>	Integer	Key for the plan item that this note is associated with
<b>NOTE_TEXT_KEY</b>	Integer	Key of an entry in the Text dimension that holds the value of this note
<b>ROW_EFF_START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above



## Fact: Plan Item Dependencies

Real world plans often contain dependencies between elements, tasks, and milestones. The Plan Item Dependency table captures these dependencies.

Column	Type	Description
<b>PLAN_ITEM_DEPENDENCY_FACT_KEY</b>	Integer	A unique ID for this table row
<b>PREDECESSOR_PLAN_ITEM_KEY</b>	Integer	Key for the plan item that is the predecessor in this dependency
<b>SUCCESSOR_PLAN_ITEM_KEY</b>	Integer	Key for the plan item that is the successor in this dependency
<b>DEPENDENCY_TYPE_KEY</b>	Integer	Key for an entry in the dependency type dimension
<b>DEPENDENCY_LAG_TIME_DAYS</b>	Number	The lag time for this dependency, expressed in days. Lag times of zero are common. A negative value in this column indicates a lead time.
<b>ROW_EFF_START/END_DATE, ROW_CURRENT_FLAG, ROW_CREATED/LAST_UPDATED_BY_KEY</b>		...same as above



Each row in this table defines a single dependency between two plan items (which could be WBS Elements, tasks, or milestones). It is worth noting that the predecessor and successor can potentially belong to different projects.