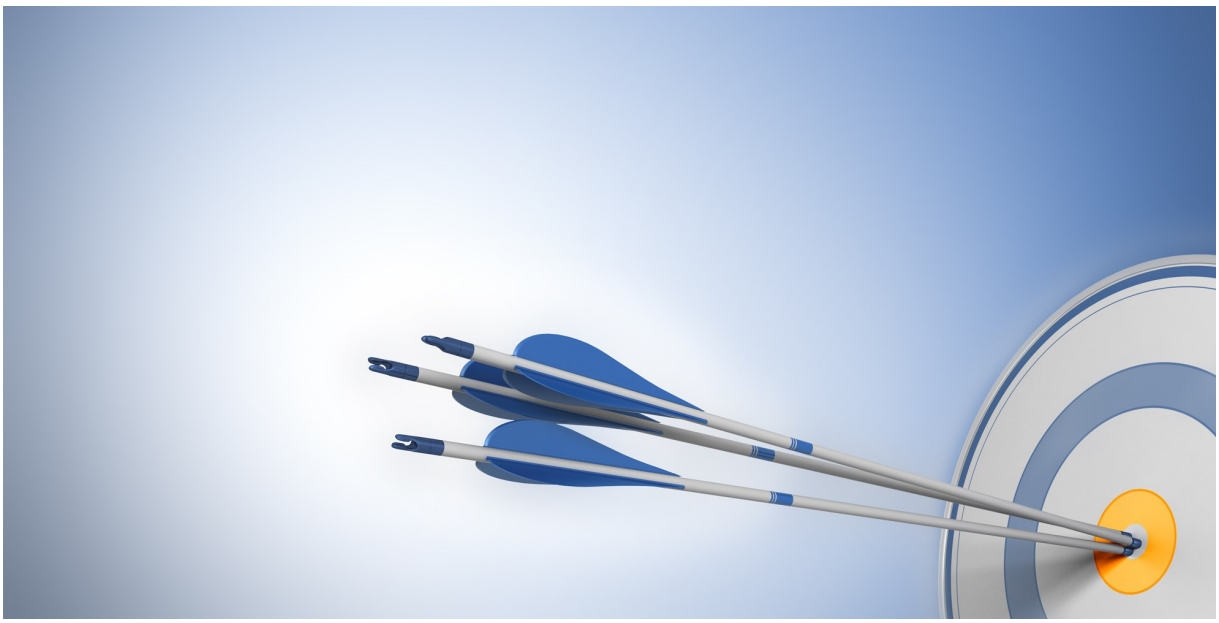

ConSol Software GmbH

ConSol CM DWH Manual

Version 6.10.5 - 6.10.7

Contents

A - Introduction

ConSol CM is a customer-centric business process management application. The data which is produced during work with the system do not only serve as a basis for the business process but they also represent an important source of information for reporting purposes.

A default ConSol CM system provides the functionalities for the installation of a Data Warehouse (DWH). This DWH is created and filled by the ConSol CM Reporting Framework (*CMRF*). CMRF is a Java EE application which can be run on the same server as ConSol CM or on a separate system.

A.1 This Book's Structure

In order to give you a quick introduction to the subject Data Warehouse, this documentation starts with the section Short Introduction to Data Warehouse Concepts.

The following section provides an overview of the ConSol CM DWH, see section Introduction to the ConSol CM Data Warehouse (DWH).

Details about the DWH tables are provided in section DWH Table Structure.

A.2 List of Manuals

ConSol CM provides documentation for several groups of users. The following documents are available:

- **Administrator Manual**
  A detailed manual for CM administrators about the ConSol CM configuration using the Admin Tool.
- **Process Designer Manual**
  A guideline for workflow developers about the graphical user interface of the Process Designer and how to program workflow scripts.
- **Operations Manual**
  A description of the ConSol CM infrastructure, the server integration into IT environments and the operation of the CM system, for IT administrators and operators.
- **Setup Manual**
  A technical description for ConSol CM setup in different IT environments. For expert CM administrators.
- **User Manual**
  An introduction to the ConSol CM Web Client for end users.
- **System Requirements**
  List of all requirements that have to be met to install ConSol CM, for IT administrators and CM administrators. Published for each ConSol CM version.
- **Technical Release Notes**
  Technical information about the new ConSol CM features. For CM administrators and key users. Published for each ConSol CM version.

A.3 Legal Notice

Since we would like to provide a manual for you which helps you manage your CM system, but which also provides additional information about connected topics, we have inserted external links into the manual. In this way, you can get some background information about a topic if you like. This can help you better understand the required CM configuration. Despite careful review, we assume no liability for the content of those external links. The operators of sites linked to are exclusively responsible for their content.

A.4 Gender Disclaimer

As far as possible, ConSol CM manuals are written gender-neutral and often address the user with "you". When the phrasing "The user .... he ..." is used, this is always to be considered to refer to both, the feminine as well as the masculine form.

A.5 Copyright

B - Short Introduction to Data Warehouse Concepts

B.1 What is a Data Warehouse?

A Data Warehouse (DWH) is a system used for reporting and data analysis. It can be used to answer questions like *Were the sales figures in January this year better than the figures last year?*, *How many working hours were booked on the migration project in the IT department?*, *How many complaints did we have this week? - From which customers? For which reasons?*. In short, a DWH helps aggregate and combine corporate data to provide a basis for the responsible persons to better understand and thus improve the business.

The data in a DWH can originate from one source or from several different sources, i.e. a DWH can represent a collection of data from different "spheres" to provide a single point of access to integrated reports. The data from operational systems (i.e. data sources) are replicated asynchronously to the DWH. Hence the DWH usually does not have the most current live data. However, the synchronization interval can vary between longer periods (e.g. every night, every week) and on-the-fly synchronization leading to DWH data more or less similar to the current production/live (transaction) data.

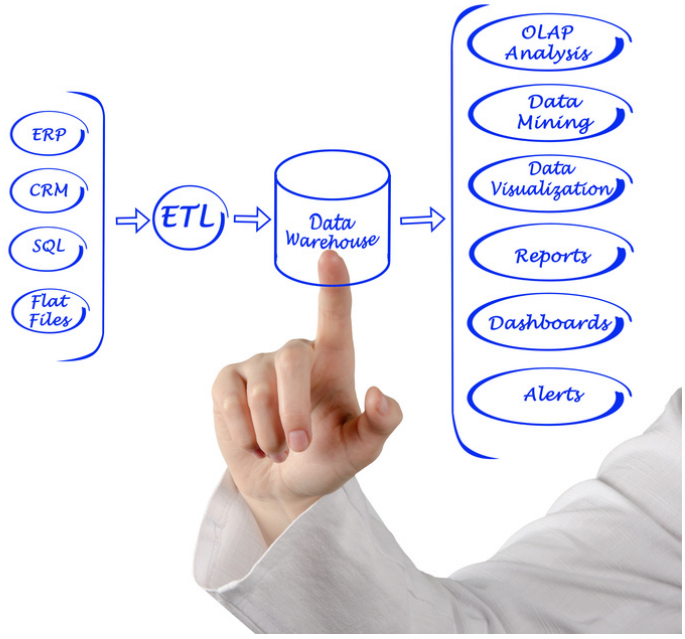

Figure 1: *DWH principle*

A DWH can have a wider or smaller scope, depending on its area of operation. A "pure" ConSol CM DWH, for example, contains ConSol CM data only and might be used by the teams who work with CM

and by managers who have to retrieve consolidated figures about the system. A corporate DWH might contain integrated data from various departments of a company and can be used by different teams each of them having different access rights and/or different reports to work with.

In a DWH, the data is usually stored in a form which is more content-based, in contrast to the operational databases where the data is stored operation-based. In ConSol CM, for example, the data of tickets is "distributed" over several tables to optimize access. In the DWH, the main ticket data can be found in a few tables named *fact_ticket_<object>*, e.g. *fact_ticket_engineer*.

B.2 Why Use a Data Warehouse ?

The use of a DWH offers some advantages:

1. The SQL statements which are used to retrieve reporting data are not executed on the production database, which would decrease the performance in the live environment, but on a separate database. Thus even highly complex queries processing large datasets will not have any influence on the production environment.
2. In a data warehouse, the data is stored in a form which is optimized for reporting, i.e. the data model is different from data models which are optimized for the daily operation of an application. In the latter, the table structure and Entity Relationship (ER) model has to be designed in a way that the application can retrieve, write, update and delete data with a very high performance. In a DWH, on the contrary, the data has to be stored in a way that is optimized for queries against large sets of usually historical data. This always requires an application which retrieves the data from the live database(s), transforms it and writes it into the DWH. This is usually an *ETL* (*Extract - Transform - Load*) operation. However in future DWH environments it might become more and more an *ELT* operation, which performs the transformation within the system which hosts the DWH.
3. A DWH can integrate data from different origins thus allowing easier access to combined data compared to operations which have to use several database connections and complex SQL statements to combine data sources in one report on-the-fly.
4. A DWH can store historical data, even if the data is no longer available on operational systems.

B.3 What Can I do With a Data Warehouse ?

Once a DWH has been created and is up and running, being filled on a regular basis, different types of reports can be used to retrieve data, for example

- simple reports like *How many tickets were opened in the Help Desk queue during the last week?*
- complex reports where the user can enter some parameters before the reporting process is started, like *How many tickets were opened in the Help Desk queue during the week that was selected by the user?*
- adhoc reporting based on data cubes (OLAP cubes) where the user can select all parameters which are part of the cube
- dashboards which provide an integrated view of several reports

The form and graphical representation of the reported data depends on the BI application which is used to build the reports. Tables and graphics like bars or pie charts are used in most cases.

A DWH provides a basis for several ways of generating information from data, e.g. data mining, working with *Big Data*.

A well-designed DWH provides data retrieval on several levels, from highly aggregated data/reports to drill-down analyses where a user can retrieve more and more details for data sets of a higher level.

C - ConSol CM DWH

The subsequent sections provide information about the ConSol CM Data Warehouse (DWH).

This chapter discusses the following:

C.1 Introduction to the ConSol CM Data Warehouse (DWH)

C.1.1 General Description

This documentation describes the ConSol CM Data Warehouse (*DWH*). The document provides an introduction to the data model and gives an overview of the DWH tables. The DWH tables can be stored in the productive database or alternatively in a separate database. A separate (i.e. DWH-specific) database is recommended to avoid the additional load generated by database requests from reports and/or data cubes.

The DWH is created and filled by the ConSol CM Reporting Framework (*CMRF*). The entire configuration is done using the ConSol CM Admin Tool. For a detailed introduction to CMRF and CMRF management, please refer to the *ConSol CM Administrator Manual*, chapter *Using CM for Reporting - Data Warehouse DWH Management*.

The following figure provides an overview of the system architecture with one ConSol CM and one CMRF server.

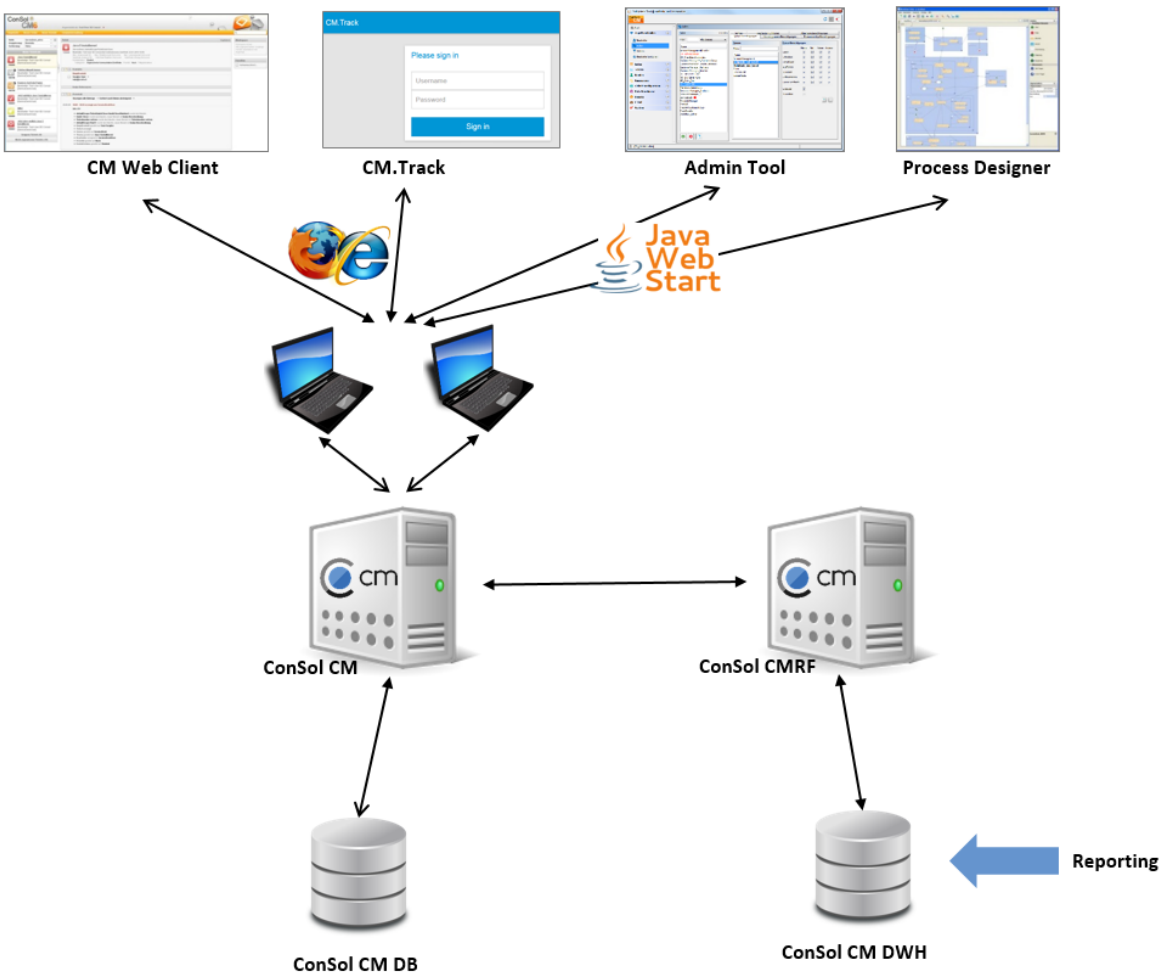

Figure 2: *ConSol CM system architecture with DWH and CMRF (two servers)*

The DWH is filled based on the following principle:

- The internal control tables are created when the CMRF is installed and started for the first time. Here, the tables for the date and time dimension are also created and filled.
- The static DWH tables (e.g. *DIM_<name>* and *FACT_<name>* tables) are created during the first data transfer from the CM database.
- The DWH tables (e.g. DIM and FACT tables) are updated during update operations based on the DWH mode (LIVE; ADMIN). For details about this topic, please read the section *DWH Management Using the Admin Tool* in the *ConSol CM Administrator Manual*, chapter *Using CM for Reporting - Data Warehouse DWH Management.* The transfer in the LIVE mode is also asynchronous, so it does not influence the work with the productive CM database. This ensures that massive DWH data updates do not slow down the CM application. But it is possible that the DWH data - even in live mode - can be a bit older than the data in the production system.
- The dynamic DWH tables for the Custom Fields are created, altered or dropped during the initial data transfer or due to changes in the ConSol CM Admin Tool.

C.1.2 Extendability of the DWH Database

By default only the standard CM data is replicated to the DWH database. Custom Fields and Custom Field Groups can be added by setting the annotations *reportable* and *reportable-group* to *true* in the ConSol CM Admin Tool.

Custom Fields can be used to precompute data within CM, e.g. for storing the timestamp when a specific activity was first executed for a ticket or storing the functional closing date in addition to the technical closing date of the tickets. These Custom Fields can be annotated as invisible within the CM application if they are only relevant for reporting.

Our customers can manually extend the DWH database by adding custom tables. This allows consolidated reporting for all applications of the customer. But the data structure of the DWH tables may not be modified.

C.1.3 Database Indexes

The CMRF creates a base set of indexes for the DWH tables. As the reports are custom made, additional indexes may be necessary to ensure efficient execution.

The custom indexes on the dynamic tables can be dropped by CMRF. After changes to Custom Fields you should verify if your custom indexes still exist.

C.2 DWH Data Model

This section gives a brief overview of the types of tables used by the DWH.

The DWH table structure is explained at the end of the document (before the appendix).

C.2.1 Table Naming Conventions

All the tables are part of one single database schema. For a visualization of the naming scheme, refer to the figure below: *Table naming conventions: Example tables from the DWH database of the docu scene*.

We distinguish several kinds of tables:

- Tables for business data of CM, for which there are two different naming schemes:
  - **FACT_*** tables for the movement data. This is the primary data individual to each ticket. We distinguish between:
    - static tables (*fact_ticket_**) which exist in all CMRF databases
    - dynamically created tables (*fact_t_** and *fact_l_*) for data of the custom data model*
  - **DIM_*** tables for the master data.
    - static tables (*DIM_<name>*) for the basic CM objects like queue, action, scope, engineer, customer and resource and also for the relations between those objects. These tables exist in all CMRF databases.
    - dynamically created tables for custom data, e.g.
      - enumeration (*dim_e_**)
      - MLA (Multi Level Attribute) (*dim_m_**)
      - data object group (dim_c_*)
      - resource field group (*dim_r_**)
- **HLP_CALENDAR_***: tables containing the custom calendars
- **INT_***: internal tables of the CMRF application which are not further described in this document
- **HLP_***: additional helper tables for configuring and saving status information of the CMRF application. They are configured in the ConSol CM Admin Tool. (Exception: HLP_CALENDER)

Figure 3: *Table naming conventions: Example tables from the DWH database of the documentation scenario*

C.2.2 General Table Structure

The following columns are part of most tables:

- **<entity_name>_id**
  Tthe primary key for an entity table or a foreign key in dependent tables. This column is used to define the foreign key constraints.
- ***<entity_name>_uid***
  The corresponding *transfer_key* from the CM database. Indexes of these columns do exist, but foreign key constraints do not.
- ***_date_id, *_time_id***
  These columns always reference *dim_date* and *dim_time* entries. The *time_id* is obtained from the entry with the next higher time value. For example, 8:02:01 is set for the time entry with the value 8:03. These allow you to easily aggregate hierarchies of date and time.
- ***name*** vs **name_<language>**
  For localized objects the column *name* contains the technical internal name, which cannot be changed within the CM. For each supported localization language a column *name_<language>* with the localized name is added to the table. For static tables see the restrictions in .

> **Important Note**
>
> The IDs do not match the ID values in the CM database. The IDs are changed, when the DWH is reinitialized.
>
> IDs should only be used in join conditions. Use the *name* columns in the referenced dim_* - tables to refer to specific objects.

> **Important Note**
>
> The data model of the documentation CM scene only supports English and German.

- ***_brutto, *_netto***
  These columns contain durations in seconds either for a single activity or for the duration between specific activities. *brutto* is always just the difference between the start and end times, whereas *netto* times show the duration considering the office hours of the corresponding workflow calendar.

C.2.2.1 Historization

When ConSol CM objects change, the new status is saved in history tables. The history table names end on **_chg* or **_log*. The column of the changes' timestamps in CM is *insert_datetime*.

The activity's brutto/netto time in history tables corresponds to the previous activity. Additionally these history table entries contain the previous value of the column values.

C.2.2.2 Denormalization

There are some denormalizations used in the DWH database:

- Action and activity names are stored in the *fact_*_log/chg* tables instead of their associated IDs to avoid additional joins on *dim_action* and *dim_activity*.
- The **fact_*_log/chg** tables also contain a lot of internal names of referenced objects, e.g. engineer, contact.
- In addition to timestamps, the foreign keys to *dim_date* and *dim_time* are added to most tables to enable joins on the foreign keys instead of using date/time calculations.

C.3 Overview of the DWH Data Model

The ConSol CM DWH contains static tables and dynamic tables. Please refer to the following sections:

- ConSol CM DWH Static Tables
- ConSol CM DWH Dynamic Tables / Custom Data Model

C.3.1 ConSol CM DWH Static Tables

This section gives an overview of the tables included in all CM DWH databases. The following topics, data types and tables are explained:

C.3.1.1 Introduction

The data types are taken from an installation using a MySQL database. On Oracle and Microsoft SQL Server the data types are slightly different.

Note regarding Oracle databases:

The table and column names of the dynamic tables are all in lower case in the data dictionary. Hence in the SQLs these names must be escaped via "...", otherwise they are not found.

Example:

*select * from "dim_e_medium"* works.

*select * from dim_e_medium* leads to "ORA-00942: table or view does not exist".

This data model diagrams contains the localizations only in English.

> **Important Note**
>
> For better readability in the ER (Entity Relationship) diagram, the relations to *dim_date* and *dim_time* are omitted.

C.3.1.2 Localization columns in static tables

Since CM version 6.10, the localized values, e.g. *name* and *description,* can be retrieved for static tables in the same manner as for dynamic tables.

Note, for some static tables not all localized columns are included. E.g. in *dim_queue* only the *description* is localized, but not the *name.*

Since CM version 6.10.3, the localized values are not included (i.e. transferred to the DWH) by default as this leads to longer update times for the static tables during *(re)initialization*, *transfer* and *update*.

To include the localized values, the CM system property *cmrf.localization.enabled* has to be set to *true*. Please refer to the *ConSol CM Set-Up Manual, section DWH-related Java System Properties* for more information on how to configure these CMRF system properties.

Important note

Special treatment for MySQL DWH databases:

Due to table row restrictions in MySQL the localization for two languages at maximum can be directly stored in the DWH tables. Here per default only the columns for the default language are included.

If you want two or other languages the additional Java (not CM!) system property *cmrf.mysqlLocales* must be set. The following syntax is required: -Dcmrf.mysqlLocales=<comma-separated list of locales>

For example, start ConSol CM with the following Java system property for MySQL support of German and English localization:

-Dcmrf.mysqlLocales=en,de

Important Note

Updating from 6.10.1/2 to to 6.10.3 and higher

The localization columns are automatically removed from the static tables, if the system property *cmrf.localization.enabled* is not used.

When starting CMRF with this system property the columns are immediately added but are initially empty. A *DWH update* is necessary to fill the column.

Alternatively you can determine the localized values from the table *dim_localized_property* via join over the objects uid (aka transfer keys (tk)).

Examples

```
select s.scope_id
     , s.name as name
     , prop.value description_de
   from
      dim_scope s
         join dim_localized_property prop ON s.scope_uid = prop.object_tk
   where locale = 'de'
      and property_name = 'description' -- name of localized value
      and prop.valid_to is null -- use currently valid value
      ;
```

```
select q.queue_id
     , q.name as name
     , prop_name.value name_en
     , prop_desc.value description_de
   from dim_queue q
      left join dim_localized_property prop_name ON q.queue_uid = prop_name.object_
       tk
         and prop_name.locale = 'en'
         and prop_name.property_name = 'name'
         and prop_name.valid_to is null
      left join dim_localized_property prop_desc ON q.queue_uid = prop_desc.object_
       tk
         and prop_desc.locale = 'de'
         and prop_desc.property_name = 'description'
         and prop_desc.valid_to is null
         ;
```

C.3.1.3 Dimensions Date and Time

The dimension tables *dim_date* and *dim_time* are built for better aggregation of measures for typical date and time periods such as hour, quarter of an hour, date, week, etc. They contain the values of predefined periods of time.

The time periods and formats for the full week and month as well as for quarter values are defined in the configuration table *hlp_parameter*. These dimension tables are created during the initial CMRF start.

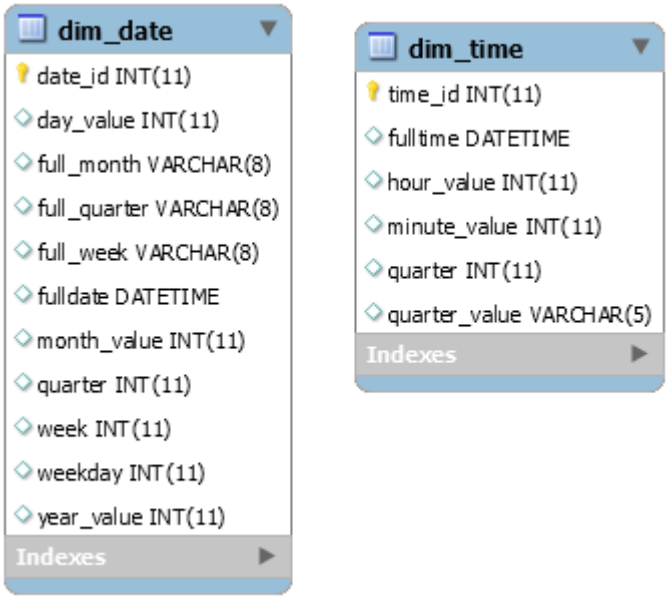

Figure 4: *Date and time tables*

> **Important Note**
>
> For each ***_date_id** / ***_time_id** corresponding columns exist in the static table.

C.3.1.4 Tickets

In this section the central tables necessary for reporting the current ticket state are described.

The central fact table is *fact_ticket* containing the references to all dimensions.

The only standard measures for tickets are their aggregated time values. These durations of ticket activities are stored in the table *fact_ticket_time*, which has to be joined using *ticket_time_uid = ticket_uid*.

Additional measures can be implemented via Custom Fields, which will result in custom/dynamic *fact_** tables.

The relation to resources, contact and engineer is generally ***n:m*** and modeled using the relation tables *dim_resource_ticket_relation*, *fact_ticket_contact* and *fact_ticket_engineer_user*.

The IDs of the dedicated first contact and the responsible engineer are also stored in *fact_ticket, as they exist for every ticket.*

The data model for resources and contacts is described in more detail in the corresponding topic below.

The tickets' current state in the workflows can be determined mapping *scope_id* and *activity_id* to *dim_scope* and *dim_activity*.

All information about attachments, e-mails and comments, but not their content itself, can be found in *fact_content_entry*. Typically for reports only the number of e-mails or the number of comments should be relevant.

Hence the large data volume of texts and file attachments is omitted in the DWH.

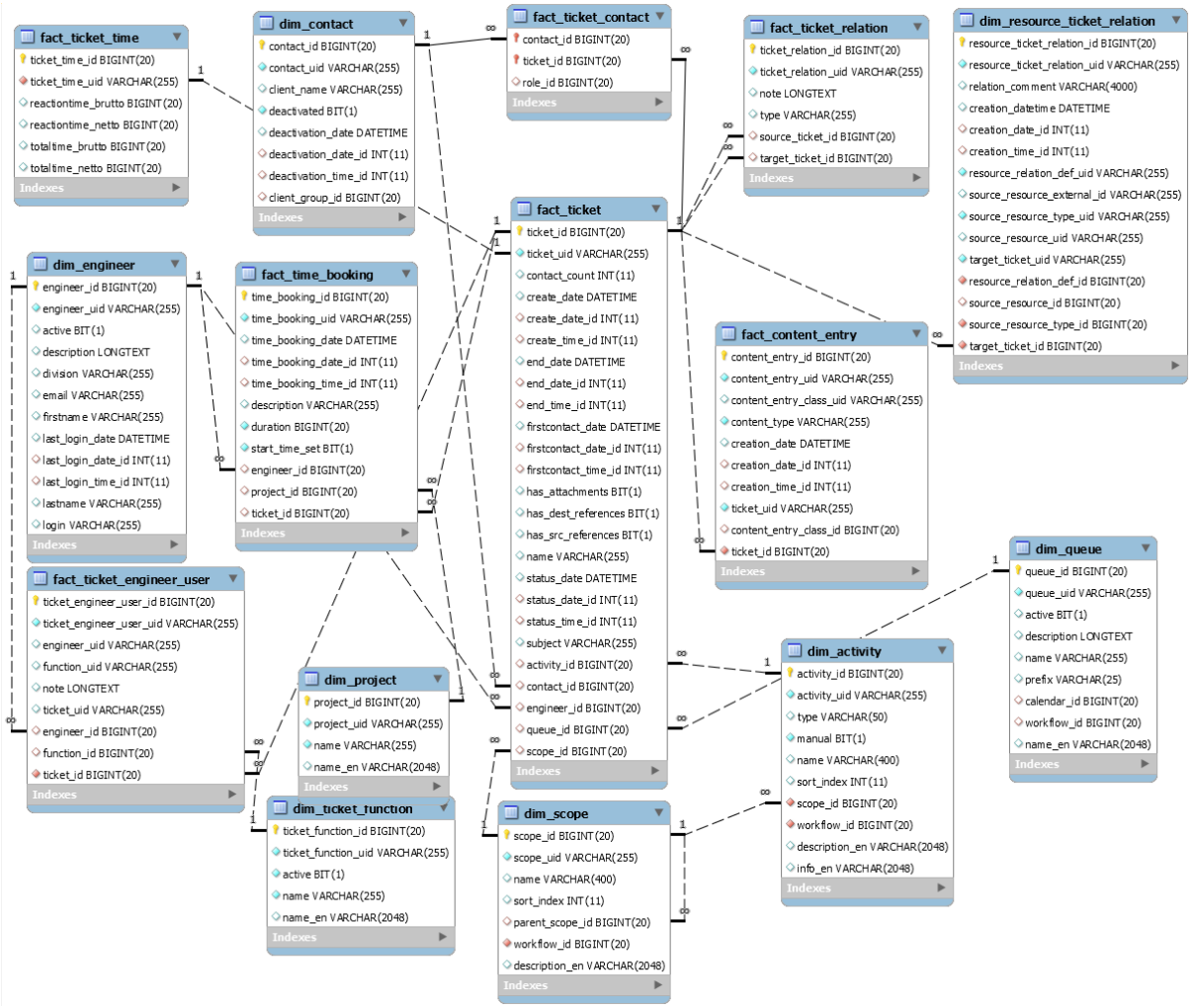

Figure 5: *Ticket related tables*

C.3.1.5 Ticket History

While only the current state of the tickets can be reported from the table *fact_ticket*, there exist several history tables through which the previous states can be reconstructed.

For each change of engineer, queue or workflow activity a log entry is created in the corresponding *fact_ticket_*_chg* table. All changes related to the ticket history in the CM can be found in *fact_ticket_log*.

In these tables, in addition to the *engineer_id*, an *executor_id* exists, that tracks which changes have been performed by which engineer (*dim_engineer.engineer_id*).

Important Note

As the table *fact_ticket_log* is very large, the **_chg* tables should be used when applicable. Especially when the *handling_times* should be aggregated in *fact_ticket_log* all the entries between two activity changes must be aggregated. In the **_chg** table the **time_brutto/netto** refers to the duration since the last change of the corresponding attribute.

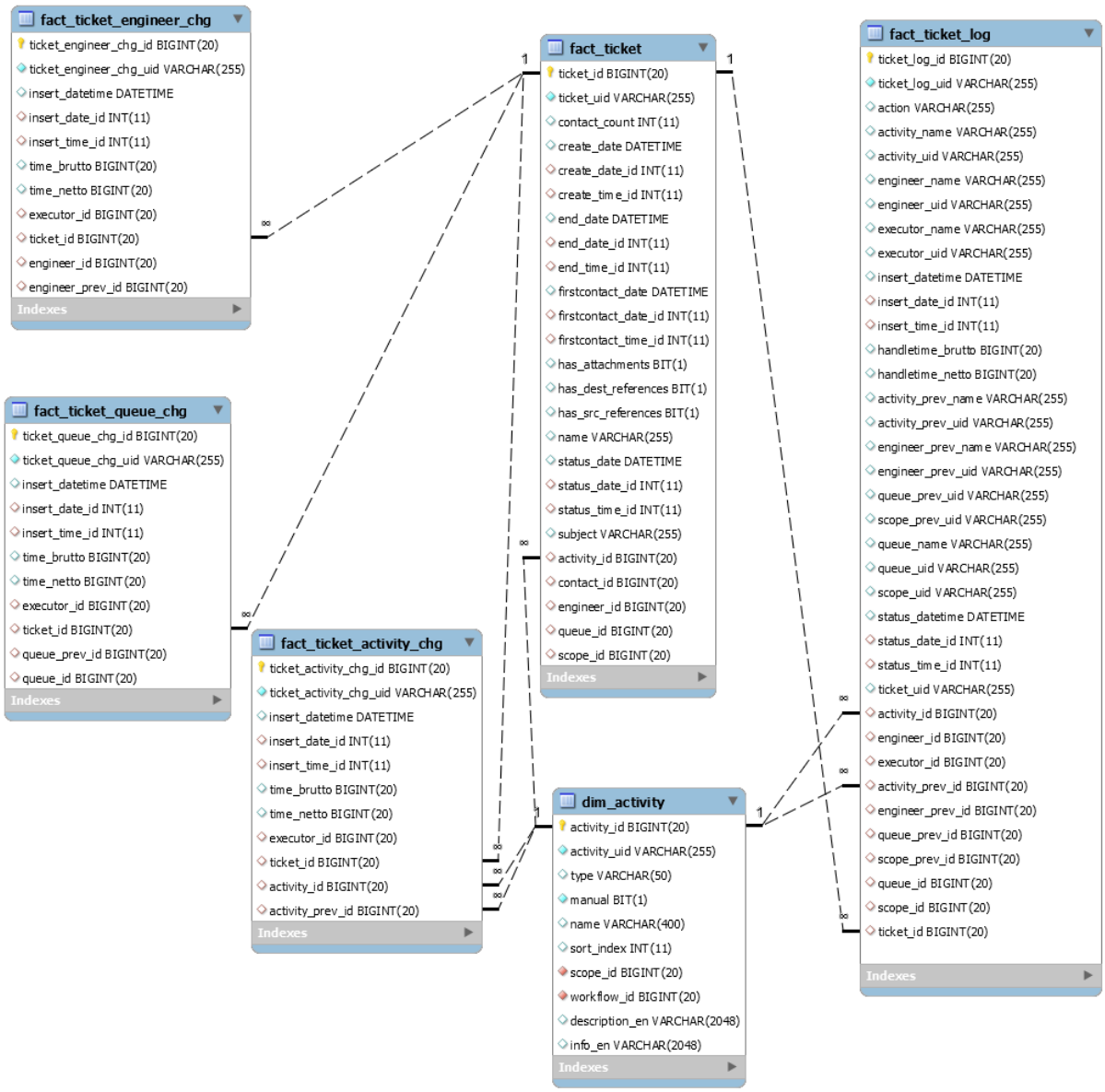

Figure 6: *Ticket history-related tables*

For better readability of the diagram the dimension tables are omitted. Analogous to *fact_ticket* foreign key relations exist in the history tables.

C.3.1.6 Contacts/Units

In the static ConSol CM data model the contact contains only an ID and information whether the contact is active or not. All other data is modeled via data object group fields and data object groups contained in dynamic tables of the DWH database.

Some contact relation tables use *unit* instead of *contact* and *unit_id* instead of *contact_id*. *Unit* is another technical name for the contact (in fact, it is the name of the corresponding Java class).

dim_contact is the base table for customers or companies in CM. The table *dim_customer_definition* defines the customer data model. Each customer in the *dim_contact* entry belongs to exactly one *dim_customer_definition* and hence one customer data model.

A contact can have a relation to other contacts. This is modeled via *dim_unit_relation*. The type of the relation is defined via *dim_unit_relation_definition*. Similarly, a specific role can be assigned to the relation to the ticket.

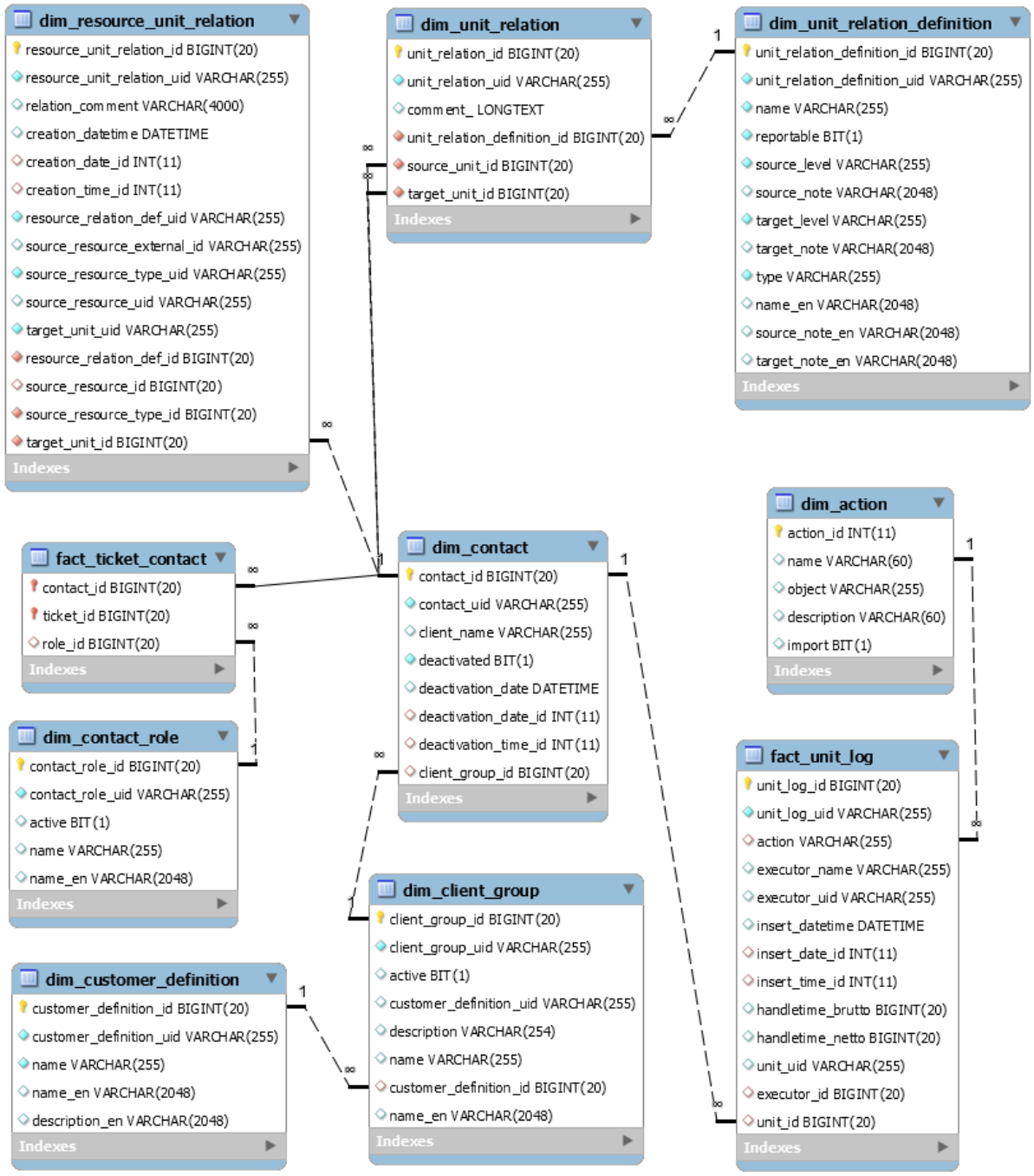

Figure 7: *Unit-related tables*

C.3.1.7 Resources

As for contacts the static CM data model for resources contains few general fields. Apart from the internal technical ID and the *resource_id* via *external_id*, a customer-specific ID or name can be stored.

The relations to ticket and contact are modeled in the tables *dim_resource_ticket_relation* and *dim_resource_ticket_unit_relation*. The relation can either be a fixed resource (typically specific hardware) or just a general resource type (e.g. a hardware model).

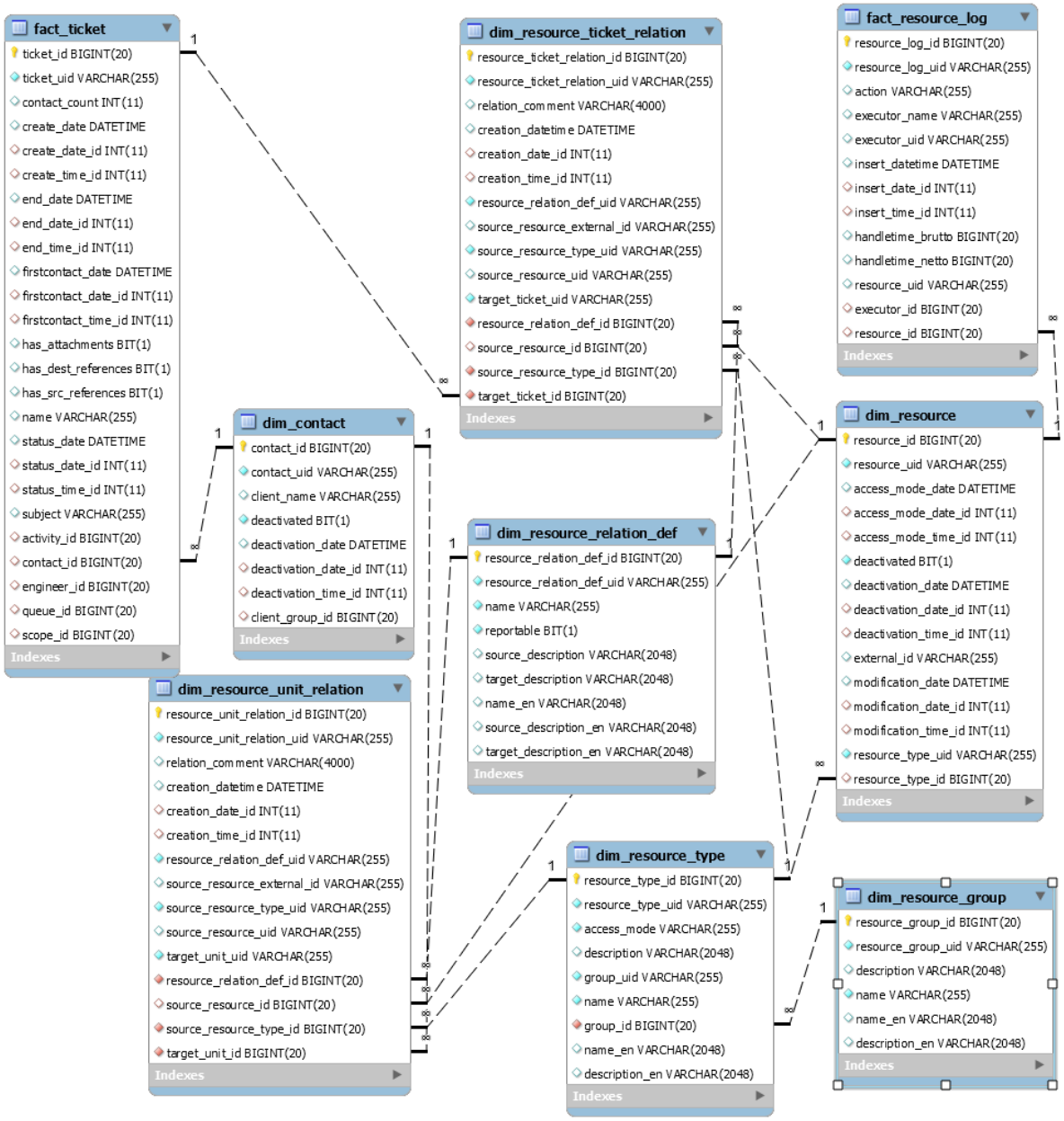

Figure 8: *Resource-related tables*

C.3.1.8 Workflow / Custom Data Definitions

The last topic on static tables contains information on the custom data model and the workflow. This includes the master data model of the CM scene. Typically this data is not used for reporting, but is essential for creating reports on dynamic tables and analyses of customer-specific process workflows.

The following tables describe the custom data model:

- ***dim_group_definition*** for the Custom Field Groups of tickets, resource groups and data object groups.
  They are distinguished by the column type: *UNIT* for customer data and *TICKET* for Custom Field Groups and *RESOURCE* for resource-specific fields.
- ***dim_field_definition*** for the Custom Fields. This includes information about associate Custom Field data types as lists, structs and enumeration dimensions, as well as the definition of basic data types via the column "*TYPE*. It also includes the names of the corresponding history tables.
- ***dim_enum_group*** and ***dim_mla*** for the mapping of enumeration names and MLA names to dimension tables.
- ***dim_queue*** lists the queue including the link to the associated calendar and workflow, if used.

  > **Important Note**
  >
  > In the CMRF database the connection between queues and Custom Field Groups is not modeled.

- ***dim_customer_definition*** lists the customer data model with some additional details.
- ***dim_client_group*** lists the related customer groups. There can be one or more customer data models per queue. This correspondence is modeled via *dim_client_group* and the relation table *hlp_queue_client*, which needs to be joined with *dim_queue*.

The tables for calendar modeling all start with *hlp_calendar**.

The basic structures of the workflow can be found in:

- ***dim_workflow*** contains the name and version of the workflow.
- ***dim_scope*** contains the scope belonging to each workflow. The nested scopes are modeled via a *parent_scope_id*, which is a foreign key to the surrounding scope.
- ***dim_activity*** contains the activities belonging to the respective scopes and workflows.
- ***dim_action*** contains the list of the basic actions, which can per performed on tickets and units. These are e.g. *create/open*, *update, subject_change*, *comment_added*.

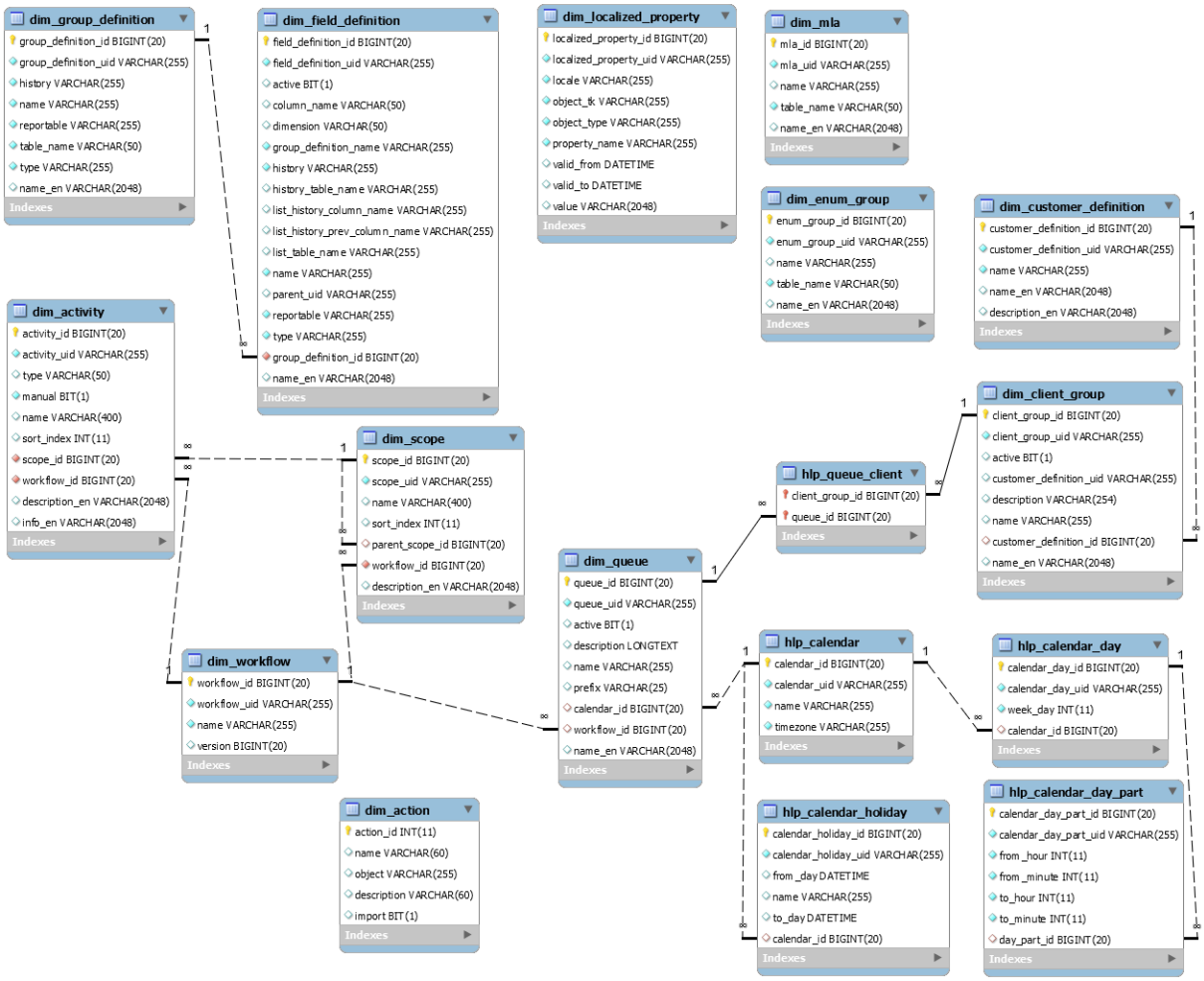

Figure 9: *Workflow-related tables*

C.3.2 ConSol CM DWH Dynamic Tables / Custom Data Model

In this section the general structures of the tables for the custom data model are presented. The following data types and tables are explained:

C.3.2.1 Custom Data Types / Dimensions

Enumerations

For each enumeration a dimension table *dim_e_** exists.

The general structure of the dimension tables for enumerations is:

Field	Type	Null
*_id	bigint(20)	NO
*_uid	varchar(255)	NO
name	varchar(255)	NO
color	varchar(255)	YES
order_index	bigint(20)	YES
enabled	bit(1)	YES
name_en	varchar(2048)	YES
name_de	varchar(2048)	YES
name_pl	varchar(2048)	YES

Only the column name of the primary key ***_id** and secondary key **_uid* are derived from the enumeration's name. The name begins either with the dimension table's name without dim_ or - if it is too long - with an abbreviation.

E.g. for *dim_e_product_gr* the columns are *product_gr_id* and *product_gr_uid*.

The display order of the definitions is represented in the *order_index* column.

The *name_** column contains the localized values for the corresponding language. The number of columns depends on the languages defined for the scenario.

MLAs

MLAs (Multi Level Attributes) contain hierarchically ordered values.

For each MLA a dimension table *dim_m_** exists.

In the Admin Tool, these MLAs can look as follows.

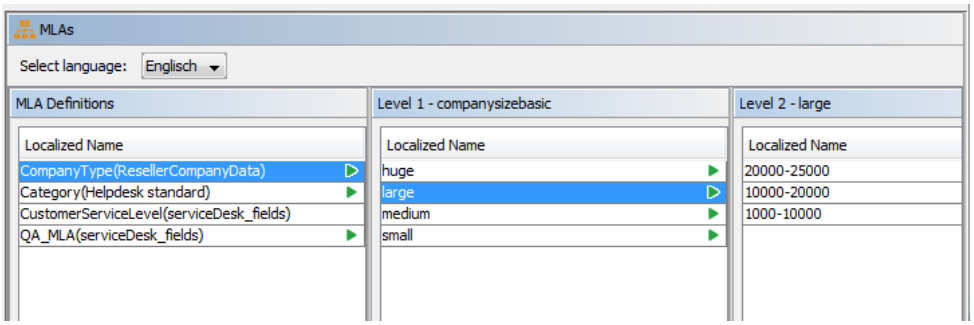

Figure 10: *Admin Tool: MLA*

The figure above shows an example of a two level hierarchy. Generally there can be more levels.

The MLA dimension tables have the general structure given below.

Field	Type	Null
*_id	bigint(20)	NO
*_uid	varchar(255)	NO
value	varchar(255)	NO
path	varchar(4000)	NO
lft	bigint(20)	NO
rght	bigint(20)	NO
lvl	bigint(20)	NO
parent_id	bigint(20)	YES
active	bit(1)	NO

Field	Type	Null
endnode	bit(1)	NO
value_en	varchar(2048)	YES
value_de	varchar(2048)	YES
value_pl	varchar(2048)	YES

The *value** columns contain the names per hierarchy level. The *parent_id* links to the parent value from the previous level. The root entry is always *root*. The *path* contains the full path through the hierarchy to the item using the names from the *value* column. *endnode* marks if the path ends (1) or not (0).

These are the entries corresponding to the Admin Tool screenshot above:

Company Type_id	value	path	lft	rght	lvl	parent_id	active	endnode
1	root	/root	1	34	0		1	0
6	large	/root/large	10	17	1	1	1	0
7	20000-25000	/root/large/20000-25000	11	12	2	6	1	1
8	10000-20000	/root/large/10000-20000	13	14	2	6	1	1
9	1000-10000	/root/large/1000-10000	15	16	2	6	1	1

Note

For better readability the *Company Type_uid* as well as the localizations where omitted. The full structure of the examples table would consist of the columns: *Company Type_id, Company Type_uid, value, path, lft, rght, lvl, parent_id, active, end-node, value_en, value_de, value_pl*

C.3.2.2 Custom Field Groups for Tickets / Fact Tables

The following section shows the general table structure for Custom Fields.

The table and column names are derived from the technical names of the Custom Fields and Custom Field Groups. At the end of this section an SQL statement is shown, which can be used to look up these names and their corresponding data type tables.

Base table for custom field groups

The base table of the Custom Field Group contains all elements which are not part of lists.

The basic structure look like this:

Field	Type	Null	Key	Comment
ticket_id	bigint (20)	NO	PRI	technical ticket ID within DWH
ticket_uid	varchar (255)	YES	MUL	ticket UID, same as in CM
cf_enum	bigint (20)	YES	MUL	example for enumeration, foreign key to dimension table
cf_number	bigint (20)	YES		example for base data type
cf_date	datetime	YES		example for *dateField*
cf_date_ date_id	bigint (20)	YES	MUL	*date_id* from *dim_date* corresponding to *cf_date* column value, for date hierarchies
cf_date_ time_id	bigint (20)	YES	MUL	*time_id* from *dim_time* corresponding to *cf_date* column value, for time of day hierarchies

These tables always contain the *ticket_id* and *ticket_uid*, as well as one column for each Custom Field which does not belong to a list and which either has the annotation *reportable=true* or whose whole group has the annotation *reportable group=true* .

For each field of the type *DateField* in CM there are additional columns containing the foreign key to *dim_date* and *dim_time*. This way date and time hierarchies can easily be included without date/time calculations.

Special case: if no reportable Custom Field without lists exists, the table is not created in the DWH, as it would only consist of the *ticket_id* and *ticket_uid*.

Lists and Structure (*struct*) Data Types

For each list within a Custom Field Group a separate *fact_l_** tables exists.

The standard table structure is:

Field	Type	Null	Key	Comment
ticket_id	bigint(20)	NO	MUL	technical ticket ID within DWH
ticket_uid	varchar(255)	YES	MUL	ticket UID, same as in CM
entry_uid	varchar(255)	YES	MUL	*entry_uid*, unique key per list element
cf_struct_elem_base	varchar(4000)	YES	YY	example for String base data type
cf_struct_elem_enum	bigint(20)	YES		example for enumeration

History Tables

For each reportable Custom Field history data exists within the DWH database, unless – starting with CM version 6.10 –

- the annotation *dwh_no_history* is set to *true* in the Custom Field Group or
- the annotation *dwh_no_history_field* is set to *true* for the Custom Field

For each Custom Field of the Custom Field Group, which is not part of a list, a separate history table exists. The general structure is as follows:

Field	Type	Null	Key	Comment
log_id	bigint(20)	NO	PRI	technical primary key
log_uid	varchar(255)	NO	UNI	UID also known in the CM database
old_value	varchar(4000)	YES		previous value, empty for first value here example for a string data type
new_value	varchar(4000)	YES		current value at *log_date* timestamp.
ticket_id	bigint(20)	NO	MUL	foreign key to the corresponding ticket
log_date_id	int(11)	YES	MUL	foreign key to **dim_date** corresponding to *log_date*
log_time_id	int(11)	YES	MUL	foreign key to **dim_time** corresponding to *log_date*
log_date	datetime	YES		timestamp of the change in CM
netto	bigint(20)	YES		
brutto	bigint(20)	YES		
executor_id	bigint(20)	YES	MUL	ID of the engineer, who did the change.

The type of the columns *old_value* and *new_value* depends on the data type of the Custom Field.

Important Note

The date of a Custom Field cannot be changed in ConSol CM.

The mapping is as follows:

- for Custom Fields of type Datetime the timestamp is stored as microseconds after 1.1.1970 00:00. Here no additional columns with *date_id* and *time_id* exist.
- for enumeration and MLA values the internal name is stored instead of the foreign key.
- the basic types are stored in the data type of the corresponding Custom Fields.

The history tables always contain the current value.

History Tables for Lists

There are separate history tables for each list within a Custom Field Group. One history table contains all Custom Fields for one list.

The general table structure is:

Field	Type	Null	Key	Comment
log_id	bigint (20)	NO	PRI	
log_uid	varchar (255)	NO	UNI	
ticket_id	bigint (20)	NO	MUL	
log_date_id	int(11)	YES	MUL	
log_time_id	int(11)	YES	MUL	
log_date	datetime	YES		
type	varchar (255)	NO		type of change: INSERT, DELETE, UPDATE
field_uid	varchar (255)	NO	MUL	UID for list entry
netto	bigint (20)	YES		
brutto	bigint (20)	YES		
executor_id	bigint (20)	YES	MUL	
cf_string	varchar (4000)	YES		example Custom Field of type String
prev_cf_string	varchar (4000)	YES		previous value of *cf_string*
cf_enum	bigint (20)	YES		example Custom Field of type enumeration. Contains the enumeration value instead of the foreign key to dimension table
prev_cf_enum	bigint (20)	YES		previous value of *cf_enum*

Field	Type	Null	Key	Comment
cf_datetime	bigint (20)	YES		example Custom Field of type Datetime. Contains micro-seconds after 1.1.1970 00:00 instead of the database datetime value.
prev_cf_datetime	bigint (20)	YES		previous value of *cf_datetime*

Two columns are provided, one for the new and one for the old value.

Meta Data of Custom Data Tables

With the following SQL statement the names of tables and columns for all Custom Fields within the DWH database can be queried:

- SQL for ConSol CM version 6.10 and higher when localizations in static tables are available:

```
select d.group_definition_name custom_field_group_tec_name
     ,d.type
     ,d.name_en custom_field_name_localized_en
     , d.name custom_field_name
     , case when d.reportable = 'UNSET' then g.reportable else d.reportable end
      reportable
     , d.column_name as cf_column_name
     , coalesce(l.list_table_name, g.table_name) cf_table_name
     , case when d.type = 'DateField' then 'BASE/dim_date/dim_time'
        else coalesce(d.dimension,'BASE type>') end dimension_table
     , case when d.history = 'UNSET' then g.reportable else d.reportable end
      history
     , coalesce(l.history_table_name, d.history_table_name) history_table_name
from dim_field_definition d
   left join dim_group_definition g on g.name = d.group_definition_name
   left join dim_field_definition s on s.field_definition_uid = d.parent_uid
   left join dim_field_definition l on l.field_definition_uid = s.parent_uid
```

Code example 1: *SQL statement when localizations in static tables are available*

- SQL for ConSol CM 6.9 version and earlier and 6.10 without using localizations in static tables:

```
select d.group_definition_name custom_field_group_tec_name
     ,d.type
     , (select min(p.value) /* only one row possible */
from dim_localized_property p
where p.object_tk = d.field_definition_uid
and object_type like '%Field%' and p.locale = 'en'
and current_timestamp between coalesce(valid_from,current_timestamp) and coalesce
 (valid_to,current_timestamp)
)
     custom_field_name_localized_en
     , d.name custom_field_name
     , case when d.reportable = 'UNSET' then g.reportable else d.reportable end
      reportable
     , d.column_name as cf_column_name
     , coalesce(l.list_table_name, g.table_name) cf_table_name
     , case when d.type = 'DateField' then 'BASE/dim_date/dim_time'
     else coalesce(d.dimension,'BASE type>') end dimension_table
     , case when d.history = 'UNSET' then g.reportable else d.reportable end
      history
     , coalesce(l.history_table_name, d.history_table_name) history_table_name
from dim_field_definition d
   left join dim_group_definition g on g.name = d.group_definition_name
   left join dim_field_definition s on s.field_definition_uid = d.parent_uid
   left join dim_field_definition l on l.field_definition_uid = s.parent_uid
```

Code example 2: *SQL statement when localizations in static tables are not available*

The columns in the query result have the following meaning:

- ***custom_field_group_tec_name***: internal name of the Custom Field.
- ***type***: CM data type of the Custom Field. The type of the Custom Field.
- ***custom_field_name_localized_en***: current name for English localization of the Custom Field. You can also add/change columns with the localized name in your preferred languages.
- ***cf_column_name***: name of the Custom Field (also within structs) in *cf_table*.
- ***reportable***: displays whether the Custom Field is available in the DWH.
- ***cf_table_name***: base database table for the Custom Field Group or corresponding list table
- ***dimension_table***: name of the **dimension_table** if the type is an enumeration or an MLA. If filled, the **ticket_column_name** only contains the reference to the primary key in the dimension table. For DateField this lists **<base>/dim_date/dim_time** to emphasize that there are three columns for the Custom Field. The *dim-column* names are derived from the Custom Fields' column names.
- ***history***: shows whether history is also available in the DWH.
- ***history_table_name***: the name of the history table for this Custom Field. For fields within lists the column names are derived from the **cf_column_name**.

> **Important Note**
>
> The table and column names are displayed, regardless of the reportable group and any reportable annotations. The table and column only need to exist in the database if *reportable* is *TRUE*. The base table_name for the Custom Field Group exists only if there is at least one column which is not list or struct. This is because the data for lists with the corresponding structs are kept in separate tables!

C.3.2.3 Custom Customer Data Models / dim_c_*

The modeling of customer data models is analogous to the modeling of Custom Field Groups for tickets. The only difference is that the tables for groups and fields are named *dim_c_** instead of *fact_t_** and *dim_e_**.

C.3.2.4 Custom Resource Model / dim_r_*

The modeling of custom resource models is analogous to the modeling of Custom Field Groups for tickets. The only difference is that the tables for groups are named *dim_r_** instead of *fact_t_** and *dim_e_**.

C.3.2.5 Examples

In the following examples, the retrieval of table structure information is shown displaying some Custom Field Groups of the documentation demo scene.

Example 1: Simple Custom Field Group *sales_standard*

The Custom Field Group *sales_standard* is defined as follows in the Admin Tool:

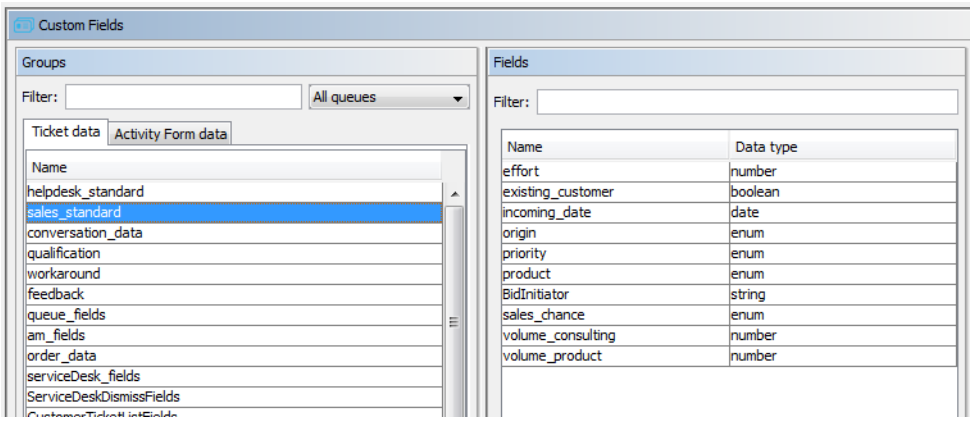

Figure 11: *Admin Tool: Custom Field Group*

With the SQL statement (MySQL) below, the DWH table structure information for the Custom Field Group *sales_standard* can be queried:

```
select d.group_definition_name custom_field_group_tec_name
     ,d.type
     ,d.name_en custom_field_name_localized_en
     , d.name custom_field_name
     , case when d.reportable = 'UNSET' then g.reportable else d.reportable end
      reportable
     , d.column_name as cf_column_name
     , coalesce(l.list_table_name, g.table_name) cf_table_name
     , case when d.type = 'DateField' then 'BASE/dim_date/dim_time' else coalesce
      (d.dimension,'BASE type>') end dimension_table
     , case when d.history = 'UNSET' then g.reportable else d.reportable end
      history
     , coalesce(l.history_table_name, d.history_table_name) history_table_name
from dim_field_definition d
   left join dim_group_definition g on g.name = d.group_definition_name
   left join dim_field_definition s on s.field_definition_uid = d.parent_uid
   left join dim_field_definition l on l.field_definition_uid = s.parent_uid where
    d.group_definition_name = 'sales_standard';
```

Code example 3: *SQL statement (MySQL) used to retrieve data of a Custom Field Group*

Exemplary results for the previous SQL query:

custom_field_ group_tec_name	type	custom_field _name_ localized_en	custom_field _name	reportable	...
sales_standard	NumberField	Effort days	effort	TRUE	...
sales_standard	BooleanField	Existing customer	existing_customer	TRUE	...
sales_standard	DateField	Incoming date	incoming_date	TRUE	...
sales_standard	EnumField	Origin	origin	TRUE	...
sales_standard	EnumField	Priority	priority	UNSET	...
sales_standard	EnumField	Product	product	TRUE	...
sales_standard	StringField	Initiator of this bid:	BidInitiator	UNSET	...
sales_standard	EnumField	Sales chance	sales_chance	TRUE	...
sales_standard	NumberField	Volume consulting	volume_consulting	TRUE	...
sales_standard	NumberField	Volume product	volume_product	TRUE	...

...	cf _column_ name	cf _table_name	dimension _table	history	history _table_name
...	effort	fact_t_sales_ standard	<base type>	UNSET	fact_t_sales_s_effort_chg
...	existing_cus- tomer	fact_t_sales_ standard	<base type>	UNSET	fact_t_sales_s_existing_ cu_chg
...	incoming_date	fact_t_sales_ standard	<base>/dim_ date/dim_time	UNSET	fact_t_sales_s_incoming_ da_chg
...	origin	fact_t_sales_ standard	dim_e_origin_gr	UNSET	fact_t_sales_s_origin_chg
...	priority	fact_t_sales_ standard	dim_e_Sales_priority	UNSET	fact_t_sales_s_priority_ chg
...	product	fact_t_sales_ standard	dim_e_product_gr	UNSET	fact_t_sales_s_product_ chg
...	BidInitiator	fact_t_sales_ standard	<base type>	UNSET	fact_t_sales_s_BidIni- tiato_chg
...	sales_chance	fact_t_sales_ standard	dim_e_sales_chance_ gr	UNSET	fact_t_sales_s_sales_ chanc_chg
...	volume_con- sulting	fact_t_sales_ standard	<base type>	UNSET	fact_t_sales_s_volume_ cons_chg
...	volume_ product	fact_t_sales_ standard	<base type>	UNSET	fact_t_sales_s_volume_ prod_chg

Important Note

Both tables are to be read as one, having the columns: *custom_field_group_tec_name, type, custom_field_name_localized_en, custom_field_name, reportable, cf_column_ name, cf_table_name, dimension_table, history, history_table_name*

The table *fact_t_sales_standard* in MySQL database:

Field	DB Type	Null
ticket_id	bigint(20)	NO
ticket_uid	varchar(255)	YES
effort	bigint(20)	YES

Field	DB Type	Null
existing_customer	bit(1)	YES
incoming_date	datetime	YES
incoming_date_date_id	int(11)	YES
incoming_date_time_id	int(11)	YES
origin	bigint(20)	YES
product	bigint(20)	YES
sales_chance	bigint(20)	YES
volume_consulting	bigint(20)	YES
volume_product	bigint(20)	YES

Important Note

For each Custom Field of type *DateField* there are three columns.

The column *incoming_date* from the data structure view linked to the fields *incoming_date_date_id* and *incoming_date_time_id* which contain the primary key to the corresponding date and minute (without day).

The *bigint(20)* columns of the example can correspond to NumberFields or are themselves foreign keys to enumeration and MLA dimension tables. E.g. referring to the result of the data structure shown above *volume_consulting* is a CM NumberField and *product* is a key value of the dimension table *dim_e_product_gr*.

The enumeration (*enum*) *product_gr* is defined in the Admin Tool as shown in the following figure.

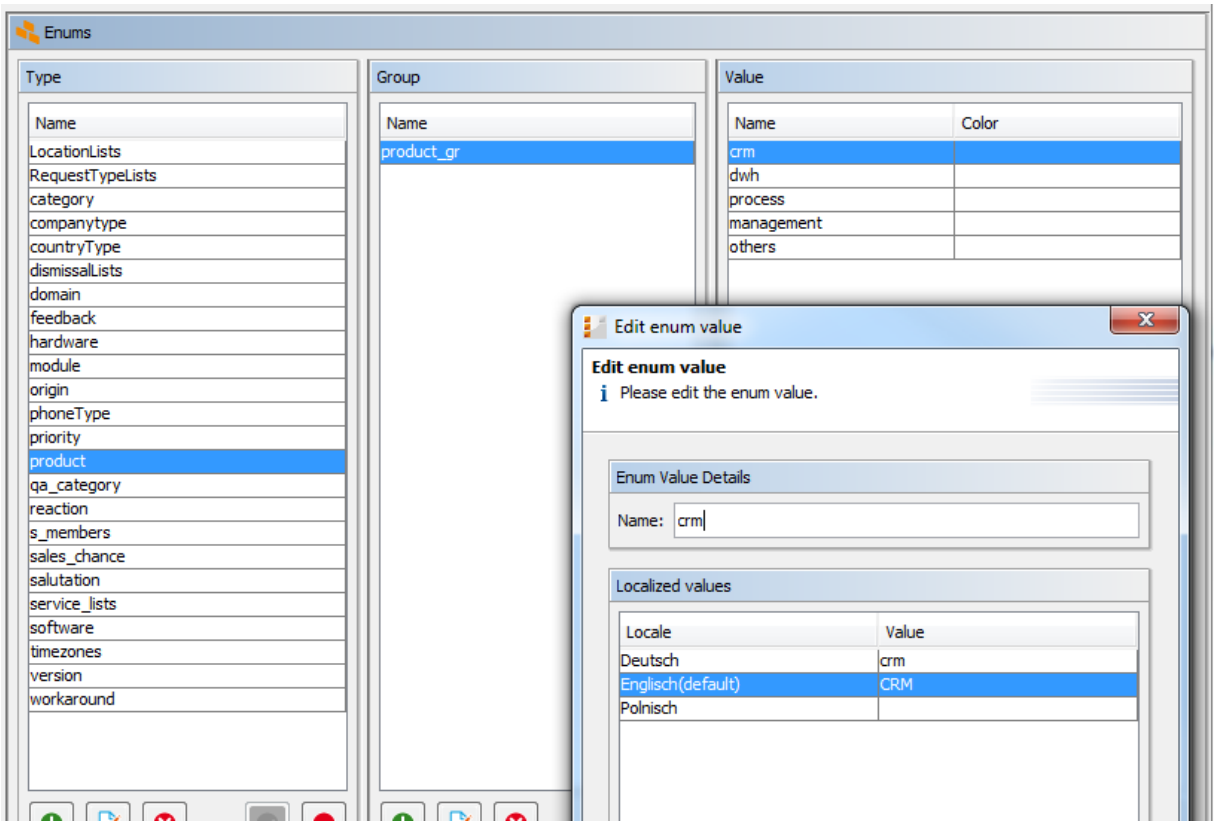

Figure 12: *Admin Tool: Enum*

The dimension table *dim_e_product_gr*

Field	Type	Null
product_gr_id	bigint(20)	NO
product_gr_uid	varchar(255)	NO
name	varchar(255)	NO
color	varchar(255)	YES
order_index	bigint(20)	YES
enabled	bit(1)	YES
name_en	varchar(2048)	YES
name_de	varchar(2048)	YES
name_pl	varchar(2048)	YES

product_gr_id is the primary key, which will be referenced in the fact table. *name* contains the technical name of the enumeration and *name_** contains the localized names per language.

The following MySQL query can be used to determine the amount of consulting volume per product.

```
select d.name_de product, sum(f.volume_consulting) sum_volume_consulting
from fact_t_sales_standard f
   join dim_e_product_gr d on f.product = d.product_gr_id
group by name_de
order by order_index
```

Code example 4: *Example SQL (MySQL) query used to retrieve the amount of consulting per product*

The result list is ordered by the display order of the enumeration values.

Example 2: Custom Field Group with List, *CustomerTicketListFields*

The Custom Field Group *CustomerTicketListFields* is defined as follows in the Admin Tool:

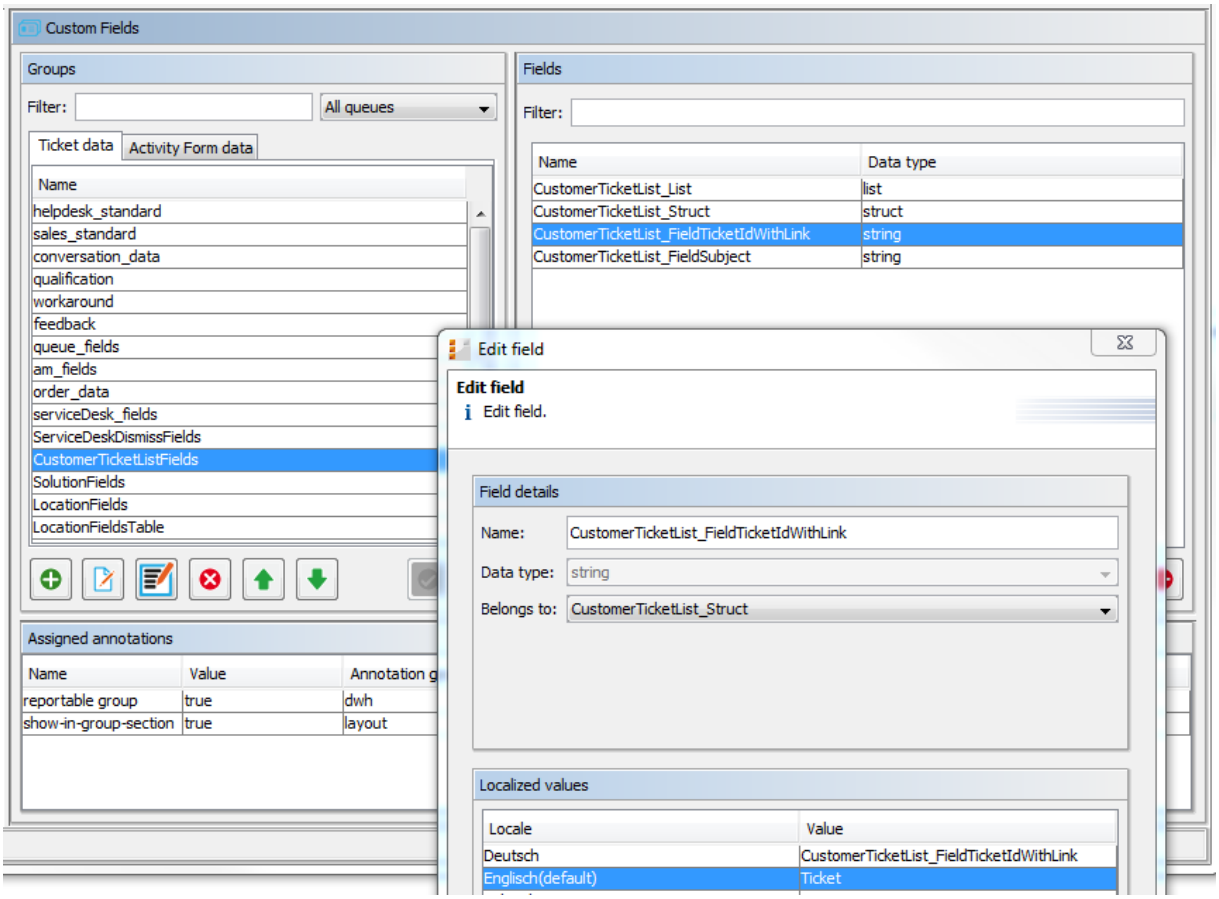

Figure 13: *Admin Tool: List of Structs*

With the SQL statement (MySQL) below, the DWH table structure information for the Custom Field Group *LocationFieldsTable* can be queried.

```
select d.group_definition_name custom_field_group_tec_name
     ,d.type
     ,d.name_en custom_field_name_localized_en
     , d.name custom_field_name
     , case when d.reportable = 'UNSET' then g.reportable else d.reportable end
      reportable
     , d.column_name as cf_column_name
     , coalesce(l.list_table_name, g.table_name) cf_table_name
     , case when d.type = 'DateField' then 'BASE/dim_date/dim_time' else coalesce
      (d.dimension,'BASE type>') end dimension_table
     , case when d.history = 'UNSET' then g.reportable else d.reportable end
      history
     , coalesce(l.history_table_name, d.history_table_name) history_table_name
from dim_field_definition d
   left join dim_group_definition g on g.name = d.group_definition_name
   left join dim_field_definition s on s.field_definition_uid = d.parent_uid
   left join dim_field_definition l on l.field_definition_uid = s.parent_uid
where d.group_definition_name = 'CustomerTicketListFields';
```

Code example 5: *SQL (MySQL) Statement used to retrive data of a Custom Field Group*

Results for the previous SQL query:

custom_field _group_tec_name	type	custom_field _name _localized_en	custom _field_name	report- able	...
Cus- tomerTicketListFields	ListField	More tickets of this customer	CustomerTicketList_ List	TRUE	...
Cus- tomerTicketListFields	StructFiel- d	Cus- tomerTicketList_ Struct	CustomerTicketList_ Struct	TRUE	...
Cus- tomerTicketListFields	StringField	Ticket	CustomerTicketList_ FieldTick- etIdWithLink	TRUE	...
Cus- tomerTicketListFields	StringField	Subject	CustomerTicketList_ FieldSubject	TRUE	...

...	cf_column _name	cf_table _name	dimension _table	history	history _table_name
...		fact_t_Cus- tomerTicketListField	<base type>	TRUE	fact_l_Custome_Cus- tomerTic_chg
...		fact_t_Cus- tomerTicketListField	<base type>	TRUE	

...	cf_column _name	cf_table _name	dimension _table	history	history _table_name
...	CustomerTicketList_ F	fact_l_Customer_Cus- tomerTicke	<base type>	TRUE	fact_l_Custome_Cus- tomerTic_chg
...	CustomerTicketList_ 1	fact_l_Customer_Cus- tomerTicke	<base type>	TRUE	fact_l_Custome_Cus- tomerTic_chg

Important Note

Both tables are to be read as one, having the columns: *custom_field_group_tec_name, type, custom_field_name_localized_en, custom_field_name, reportable, cf_column_ name, cf_table_name, dimension_table, history, history_table_name*

In the result table you see that there are no columns for the list and struct custom field: **NULL** in *cf_ column_name*. As the table *fact_t_CustomerTicketListField* has no data columns, the table is not created in the DWH database. The single element of the structure *CustomerTicketList_Struct* is created as a row in the list table *fact_l_Customer_CustomerTicket.*

The following query counts the number of related tickets.

```
select t.name ticket_nr, count(distinct CustomerTicketList_F) sum_tickets
from fact_ticket t join fact_l_Customer_CustomerTicke l on t.ticket_id = l.ticket_
 id
group by t.name
order by t.name
```

Code example 6: *SQL (MySQL) Statement used to retrieve the number of related tickets*

Important Note

The result only contains tickets with at least one entry in the *CustomerTicketListFields*.

ticket_nr	sum_tickets
100172	28
100179	8
100183	19
100185	21
...	

ConsolCM - DWH Table Structure

Here all static DWH tables for CM 6.10.5.3 with localizations for English and German are described.
The general structure of dynamic tables for the custom fields are described in the base document.
The tables are listed in alphabetical order. The data types of the columns are taken from the MySQL DB.

Table dim_action

Standard actions within the CM. The table dim_action is a static table with additional information for the actions used in the table fact_ticket_log, fact_resource_log and fact_unit_log.

Column	Type	Nullable	Comment
action_id	int(11)	NO	primary key
name	varchar(60)	YES	internal name
object	varchar(255)	YES	object typ affected by this action
description	varchar(60)	YES	description
import	bit(1)	YES	true(1): import this CM action from CM false(0): ignore this action - don't copy to DWH

The table dim_action is filled during initialization of the system.

Changes/Updates of the table are only necessary when additional actions are introduced (enum gains more values)

Only the actions mentioned in this table and having "import" set to true are copied from cmas__log to the DWH.

Note: it would be enough to simply leave out the unneeded actions, instead of using the "import" column. But then the admin would always have to worry whether actions have been left out by accident or on purpose. Using the column, this is easier to distinguish.

Table dim_activity

Activities from the CM workflows

Column	Type	Nullable	Comment
activity_id	bigint(20)	NO	primary key
activity_uid	varchar(255)	NO	corresponding unique key from CM
type	varchar(50)	YES	Value of enum ActivityType: START, REGULAR, END
manual	bit(1)	NO	automatic(0) or manual(1) execution
name	varchar(400)	YES	internal name
sort_index	int(11)	YES	order index for display
scope_id	bigint(20)	NO	foreign key to dim_scope
workflow_id	bigint(20)	NO	foreign key to dim_workflow

The table is filled with the data of the table cmas_process_element (discriminator element_type=activity)

Table dim_client_group

Client groups

Column	Type	Nullable	Comment
client_group_id	bigint(20)	NO	primary key
client_group_uid	varchar(255)	NO	corresponding unique key from CM
active	bit(1)	YES	still active in CM: true(1)=active, false(0): inactive
customer_definition_uid	varchar(255)	YES	CM key to dim_customer_definition
description	varchar(254)	YES	description
name	varchar(255)	YES	internal name
customer_definition_id	bigint(20)	YES	foreign key to din_customer_defitition

Table dim_contact

Base table for customer in CM, common for all single and two level customer models

Column	Type	Nullable	Comment
contact_id	bigint(20)	NO	primary key
contact_uid	varchar(255)	NO	corresponding unique key from CM
client_name	varchar(255)	YES	Name of client group, redundant information from dim_client_group
deactivated	bit(1)	NO	false(0)=active, true(1)=deactivated
deactivation_date	datetime	YES	timestamp of deactivation
deactivation_date_id	int(11)	YES	foreign key to dim_date
deactivation_time_id	int(11)	YES	foreign key to dim_time
client_group_id	bigint(20)	YES	primary key

Table dim_contact_role

Role for contacts in ticket contact relation

Column	Type	Nullable	Comment
contact_role_id	bigint(20)	NO	primary key
contact_role_uid	varchar(255)	NO	corresponding unique key from CM
active	bit(1)	YES	still active in CM: true(1)=active, false(0): inactive
name	varchar(255)	YES	internal name

Table dim_content_entry_class

Classes of Text: Classes of Content entries within CM

Column	Type	Nullable	Comment
content_entry_class_id	bigint(20)	NO	primary key
content_entry_class_uid	varchar(255)	NO	corresponding unique key from CM
customer_visible	bit(1)	NO	Visibility of entry for client in track: 1:visible, 0:invisible
name	varchar(255)	NO	internal name

Table dim_customer_definition

List of Customer models

Column	Type	Nullable	Comment
customer_definition_id	bigint(20)	NO	primary key
customer_definition_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	NO	internal name

Table dim_date

Dates table: contain one entry per day. statically filled during CMRF initialiation.

Column	Type	Nullable	Comment
date_id	int(11)	NO	primary key
day_value	int(11)	YES	day of month, e.g. 23
full_month	varchar(8)	YES	month with year, format as shown in hlp_parameter.dim_date_month_pattern, e.g. '2015 02'
full_quarter	varchar(8)	YES	quarter with year, format as shown in hlp_parameter.dim_date_quarter_pattern, e.g 'Q1 2015'
full_week	varchar(8)	YES	ISO week with year, format as shown in hlp_parameter.dim_date_week_pattern, e.g '2015 W03'
fulldate	datetime	YES	date value, time is empty
month_value	int(11)	YES	Quarter of year (1-4)
quarter	int(11)	YES	Quarter of year (1-4)
week	int(11)	YES	ISO week within the year
weekday	int(11)	YES	Day of week (1: Sunday, 2: Monday, [...], 6: Friday, 7: Saturday)
year_value	int(11)	YES	year as 4-digit number

The table is manually filled during setup of the system. The values dim_date_start and dim_date_month_count of the table hlp_parameter show which dates have been generated.

Example:

hlp_parameter.dim_date_start = 1900-01-01
hlp_parameter.dim_date_end = 2020-01-01
hlp_parameter.dim_date_month_count = 1440

Date ids have been generated for every day from January 1st, 2000 to December 31st, 2012 (12 years = 12x12 months later).

Note for date field insertion into other tables: all date fields from s, contacts, etc. are entered "as is" into their respective datetime columns, and additionally as references to dim_date / dim_time (using HOUR() and MINUTE() sql functions). If no reference to dim_date / dim_time can be made (the date is outside the configured range), those fields remain empty.

Please be aware that Java computes the week of year only conform to ISO-8601 if the correct calendar-locale is used. Please check the week-computing with description provided by: http://en.wikipedia.org/wiki/ISO_8601#Week_dates.

Table dim_engineer

Engineers/User of CM-Client

Column	Type	Nullable	Comment
engineer_id	bigint(20)	NO	primary key
engineer_uid	varchar(255)	NO	corresponding unique key from CM
active	bit(1)	YES	still active in CM: true(1)=active, false(0): inactive
description	longtext	YES	description
division	varchar(255)	YES	Division of the engineer
email	varchar(255)	YES	email address
firstname	varchar(255)	YES	first name
last_login_date	datetime	YES	timestamp of last login. Only filled in DWH mode LIVE.
last_login_date_id	int(11)	YES	foreign key to dim_date
last_login_time_id	int(11)	YES	foreign key to dim_time
lastname	varchar(255)	YES	last name
login	varchar(255)	YES	login id for CM-Client

Table dim_enum_group

Meta data for enumeration group, definied via Admintool

Column	Type	Nullable	Comment
enum_group_id	bigint(20)	NO	primary key
enum_group_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	YES	internal name
table_name	varchar(50)	NO	name of DWH table, which contains this data, automatically generated

Table dim_field_definition

Meta data for custom fields of ticket, contact/units or resources, definied via Admintool

Column	Type	Nullable	Comment
field_definition_id	bigint(20)	NO	primary key
field_definition_uid	varchar(255)	NO	corresponding unique key from CM
active	bit(1)	YES	still active in CM: true(1)=active, false(0): inactive
column_name	varchar(50)	YES	name of corresponding column within the DWH table, automatically generated
dimension	varchar(50)	YES	name of DWH table of enumeration or MLA of the datatype, empty for buildin datatypes
group_definition_name	varchar(255)	NO	name of enum group, redundant
history	varchar(255)	NO	Flag if history is kept in DWH: TRUE: history available, FALSE: not in DWH, UNSET:not explicitly set (then inherited from group)
history_table_name	varchar(255)	YES	name of DWH history table, NULL if not historized in CM
list_history_column_na ..	varchar(255)	YES	column name within the corresponding DWH history table, if custom field is part of a list structure. Otherwise name in "new_value"
list_history_prev_colum ..	varchar(255)	YES	column name of the previous value within the corresponding DWH history table, if custom field is part of a list structure. Otherwise name in "old_value"
list_table_name	varchar(255)	YES	DWH table name for custom field of typ ListType. Otherwise empty
name	varchar(255)	NO	internal name
parent_uid	varchar(255)	YES	Foreign key to field_definition_uid of parent entry, only set for nested custom fields

Column	Type	Nullable	Comment
reportable	varchar(255)	NO	Flag if field available in DWH: TRUE: replicated in DWH, FALSE: not replicated DWH, UNSET:not explicitly set (then inherited from group)
type	varchar(255)	NO	Data type of custom field
group_definition_id	bigint(20)	NO	foreign key to dim_group_definition

Table dim_group_definition

Meta data for custom fields groups of tickets, contact/units or resources, definied via Admintool

Column	Type	Nullable	Comment
group_definition_id	bigint(20)	NO	primary key
group_definition_uid	varchar(255)	NO	corresponding unique key from CM
history	varchar(255)	NO	Flag if history is kept in DWH: TRUE: history available, FALSE: not in DWH, UNSET:not explicitly set , not in DWH (will be overwritten by partial scene import)
name	varchar(255)	NO	internal name
reportable	varchar(255)	NO	Flag if field available in DWH: TRUE: replicated in DWH, FALSE: not in DWH, UNSET:not explicitly set, not replicated in DWH (will be overwritten by partial scene import)
table_name	varchar(50)	NO	name of DWH table, which contains this data, automatically generated
type	varchar(255)	NO	the type of CM objects this group by belongs to: TICKET, UNIT, RESOURCE

Table dim_localized_property

Localization of properties and object name, history is kept via time interval

Column	Type	Nullable	Comment
localized_property_id	bigint(20)	NO	primary key
localized_property_uid	varchar(255)	NO	corresponding unique key from CM
locale	varchar(255)	NO	locale/language abbreviation, e.g. en for english, de for german
object_tk	varchar(255)	NO	technical key, either CM label Id for entries with property_name=key, or uid of custom object
object_type	varchar(255)	NO	CM type of the property
property_name	varchar(255)	NO	type of property. E.g. key, name, description
valid_from	datetime	YES	vname valid from, emtpy for initial name

Column	Type	Nullable	Comment
valid_to	datetime	YES	name valid until, still valid if NULL
value	varchar(2048)	YES	localized property name

Table dim_mla

Meta data for MLAs

Column	Type	Nullable	Comment
mla_id	bigint(20)	NO	primary key
mla_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	YES	internal name
table_name	varchar(50)	NO	name of DWH table, which contains this data, automatically generated

Table dim_project

Project for time booking entries, configured via Admintool

Column	Type	Nullable	Comment
project_id	bigint(20)	NO	primary key
project_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	NO	internal name

Table dim_queue

Meta data for queues

Column	Type	Nullable	Comment
queue_id	bigint(20)	NO	primary key
queue_uid	varchar(255)	NO	corresponding unique key from CM
active	bit(1)	YES	still active in CM: true(1)=active, false(0): inactive
description	longtext	YES	description
name	varchar(255)	YES	internal name
prefix	varchar(25)	YES	
calendar_id	bigint(20)	YES	primary key
workflow_id	bigint(20)	YES	foreign key to dim_workflow

Table dim_resource

base data for resources, common attributes for all resources

Column	Type	Nullable	Comment
resource_id	bigint(20)	NO	primary key
resource_uid	varchar(255)	NO	corresponding unique key from CM
access_mode_date	datetime	YES	
access_mode_date_id	int(11)	YES	foreign key to dim_date
access_mode_time_id	int(11)	YES	foreign key to dim_time
deactivated	bit(1)	NO	false(0)=active, true(1)=deactivated
deactivation_date	datetime	YES	timestamp of deactivation
deactivation_date_id	int(11)	YES	foreign key to dim_date
deactivation_time_id	int(11)	YES	foreign key to dim_time
external_id	varchar(255)	YES	extenal id for the resource for interfaces to other applications
modification_date	datetime	YES	last modification date
modification_date_id	int(11)	YES	foreign key to dim_date
modification_time_id	int(11)	YES	foreign key to dim_time
resource_type_uid	varchar(255)	NO	CM internal unique ID to type of the resource
resource_type_id	bigint(20)	YES	foreign key to type of the resource

Table dim_resource_group

base data for resource groups, common attributes for all resources

Column	Type	Nullable	Comment
resource_group_id	bigint(20)	NO	primary key
resource_group_uid	varchar(255)	NO	corresponding unique key from CM
description	varchar(2048)	YES	description
name	varchar(255)	NO	internal name

Table dim_resource_relation_def

Names of the relation types between resources and/or tickets and units.

Column	Type	Nullable	Comment
resource_relation_def_id	bigint(20)	NO	primary key
resource_relation_def_u ..	varchar(255)	NO	corresponding unique key from CM

Column	Type	Nullable	Comment
name	varchar(255)	NO	internal name
reportable	bit(1)	NO	Flag if Relation should be transfered to the DWH or not
source_description	varchar(2048)	YES	Description of relation source
target_description	varchar(2048)	YES	Description of relation target

Table dim_resource_resource_relation

Relations between resources or resource types.

Column	Type	Nullable	Comment
resource_resource_relat..	bigint(20)	NO	primary key
resource_resource_relat..	varchar(255)	NO	corresponding unique key from CM
relation_comment	varchar(4000)	YES	comment
creation_datetime	datetime	YES	timestamp when entry was created in CM
creation_date_id	int(11)	YES	foreign key to dim_date
creation_time_id	int(11)	YES	foreign key to dim_time
resource_relation_def_u..	varchar(255)	NO	UID to Relation type from dim_resource_relation_def
source_resource_extern..	varchar(255)	YES	extern Id from source ressource. Redundante from dim_resource
source_resource_type_ ..	varchar(255)	NO	UID from dim_resource_type
source_resource_uid	varchar(255)	YES	UID from source ressource. Empty, if only relation with recource type.
target_resource_uid	varchar(255)	NO	UID from target ressource
resource_relation_def_id	bigint(20)	NO	ID of relation type from dim_resource_relation_def
source_resource_id	bigint(20)	YES	ID from source ressource. Empty, if only relation with recource type.
source_resource_type_id	bigint(20)	NO	ID from dim_resource_type
target_resource_id	bigint(20)	NO	ID from target ressource

Only available if relation type from dim_resource_relation_def is set reportable=yes

Table dim_resource_ticket_relation

Relations between resources and tickets.

Column	Type	Nullable	Comment
resource_ticket_relation ..	bigint(20)	NO	primary key
resource_ticket_relation ..	varchar(255)	NO	corresponding unique key from CM
relation_comment	varchar(4000)	YES	comment

Column	Type	Nullable	Comment
creation_datetime	datetime	YES	timestamp when entry was created in CM
creation_date_id	int(11)	YES	foreign key to dim_date
creation_time_id	int(11)	YES	foreign key to dim_time
resource_relation_def_u ..	varchar(255)	NO	UID to Relation type from dim_resource_relation_def
source_resource_extern ..	varchar(255)	YES	extern Id from source ressource. Redundante from dim_resource
source_resource_type_ ..	varchar(255)	NO	UID from dim_resource_type
source_resource_uid	varchar(255)	YES	UID from source ressource. Empty, if only relation with recource type.
target_ticket_uid	varchar(255)	NO	UID from target ticket
resource_relation_def_id	bigint(20)	NO	ID of relation type from dim_resource_relation_def
source_resource_id	bigint(20)	YES	ID from source ressource. Empty, if only relation with recource type.
source_resource_type_id	bigint(20)	NO	ID from dim_resource_type
target_ticket_id	bigint(20)	NO	ID from target ticket

Only available if relation type from dim_resource_relation_def is set reportable=yes

Table dim_resource_type

base data for resource types, common attributes for all resources

Column	Type	Nullable	Comment
resource_type_id	bigint(20)	NO	primary key
resource_type_uid	varchar(255)	NO	corresponding unique key from CM
access_mode	varchar(255)	NO	
description	varchar(2048)	YES	description
group_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	NO	internal name
group_id	bigint(20)	NO	primary key

Table dim_resource_unit_relation

Relations between resources and units/contacts.

Column	Type	Nullable	Comment
resource_unit_relation_id	bigint(20)	NO	primary key
resource_unit_relation_ ..	varchar(255)	NO	corresponding unique key from CM
relation_comment	varchar(4000)	YES	comment

Column	Type	Nullable	Comment
creation_datetime	datetime	YES	timestamp when entry was created in CM
creation_date_id	int(11)	YES	foreign key to dim_date
creation_time_id	int(11)	YES	foreign key to dim_time
resource_relation_def_u ..	varchar(255)	NO	UID to Relation type from dim_resource_relation_def
source_resource_extern ..	varchar(255)	YES	extern Id from source ressource. Redundante from dim_resource
source_resource_type_ ..	varchar(255)	NO	UID from dim_resource_type
source_resource_uid	varchar(255)	YES	UID from source ressource. Empty, if only relation with recource type.
target_unit_uid	varchar(255)	NO	UID from target unit
resource_relation_def_id	bigint(20)	NO	ID of relation type from dim_resource_relation_def
source_resource_id	bigint(20)	YES	ID from source ressource. Empty, if only relation with recource type.
source_resource_type_id	bigint(20)	NO	ID from dim_resource_type
target_unit_id	bigint(20)	NO	ID from target unit

Only available if relation type from dim_resource_relation_def is set reportable=yes

Table dim_scope

List of workflow scopes

Column	Type	Nullable	Comment
scope_id	bigint(20)	NO	primary key
scope_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(400)	YES	internal name
sort_index	int(11)	YES	order index for display
parent_scope_id	bigint(20)	YES	foreign key to scope_id of surrounding scope
workflow_id	bigint(20)	NO	foreign key to dim_workflow

The table is filled with the data of the table cmas_process_element (discriminator element_type=scope)

Table dim_supported_locale

List of available locales for localization

Column	Type	Nullable	Comment
supported_locale_id	bigint(20)	NO	primary key
supported_locale_uid	varchar(255)	NO	corresponding unique key from CM
locale	varchar(255)	YES	locate/language abbreviation

Table dim_ticket_function

Engineer Functions: Roles for Engineer in Tickets

Column	Type	Nullable	Comment
ticket_function_id	bigint(20)	NO	primary key
ticket_function_uid	varchar(255)	NO	corresponding unique key from CM
active	bit(1)	NO	still active in CM: true(1)=active, false(0): inactive
name	varchar(255)	NO	internal name

Table dim_time

Contains one entry per minute of a day. Statically filled during CMRF initialiation.

Column	Type	Nullable	Comment
time_id	int(11)	NO	primary key
fulltime	datetime	YES	full datetime value, day set to 1.1.1970
hour_value	int(11)	YES	hour of the day
minute_value	int(11)	YES	minute of hour
quarter	int(11)	YES	Quarter: mapping of Minutes to quarter number 0-14: 1st quarter 15-29: 2nd quarter 30-44: 3rd quarter 45-59: 4th quarter
quarter_value	varchar(5)	YES	"0-14", "15-29", "30-44" or "45-59"

Table dim_unit_relation

Relations between units/contacts

Column	Type	Nullable	Comment
unit_relation_id	bigint(20)	NO	primary key
unit_relation_uid	varchar(255)	NO	corresponding unique key from CM
comment_	longtext	YES	Description
unit_relation_definition_id	bigint(20)	NO	ID for relation defintion
source_unit_id	bigint(20)	NO	Id for source unit
target_unit_id	bigint(20)	NO	Id for target unit

Only available if relation type from dim_unit_relation_definition is set reportable=yes

Table dim_unit_relation_definition

Data object relations between units/contacts

Column	Type	Nullable	Comment
unit_relation_definition_id	bigint(20)	NO	primary key
unit_relation_definition_ ..	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	NO	internal name
reportable	bit(1)	NO	Flag if Relation should be transfered to the DWH or not
source_level	varchar(255)	NO	data model level of source: CUSTOMER or COMPANY
source_note	varchar(2048)	YES	Description for relation source
target_level	varchar(255)	NO	data model level of target: CUSTOMER or COMPANY
target_note	varchar(2048)	YES	Description for relation target
type	varchar(255)	NO	type of relation: E.g. DIRECTIONAL or REFERENCE

Table dim_workflow

List of workflows

Column	Type	Nullable	Comment
workflow_id	bigint(20)	NO	primary key
workflow_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	NO	internal name
version	bigint(20)	YES	current version of the workflow

Table fact_content_entry

The base information of history entries

Column	Type	Nullable	Comment
content_entry_id	bigint(20)	NO	primary key
content_entry_uid	varchar(255)	NO	corresponding unique key from CM
content_entry_class_uid	varchar(255)	YES	UID of content entry class
content_type	varchar(255)	NO	name of content entry class
creation_date	datetime	YES	timestamp when entry was created in CM
creation_date_id	int(11)	YES	foreign key to dim_date
creation_time_id	int(11)	YES	foreign key to dim_time
ticket_uid	varchar(255)	NO	corresponding unique key from CM

Column	Type	Nullable	Comment
content_entry_class_id	bigint(20)	YES	UID of content entry class
ticket_id	bigint(20)	NO	foreign key to fact_ticket

Table fact_resource_log

Resource history

Column	Type	Nullable	Comment
resource_log_id	bigint(20)	NO	primary key
resource_log_uid	varchar(255)	NO	corresponding unique key from CM
action	varchar(255)	YES	Name of action from dim_action
executor_name	varchar(255)	YES	Name of the executor (Engineer) or technical user/component
executor_uid	varchar(255)	YES	Reference to dim_engineer.engineer_uid. Empty if automatic activity.
insert_datetime	datetime	YES	timestamp when entry was created in CM
insert_date_id	int(11)	YES	foreign key to dim_date
insert_time_id	int(11)	YES	foreign key to dim_time
handletime_brutto	bigint(20)	YES	Difference to previous log entry (insert_datetime) - w/o workflow calendar [seconds]
handletime_netto	bigint(20)	YES	Difference to previous log entry (insert_datetime) - with workflow calendar [seconds]
resource_uid	varchar(255)	YES	transfer key of dim_resource
executor_id	bigint(20)	YES	engineer who performed the action. foreign key to dim_engineer. Empty if automatic activity.
resource_id	bigint(20)	YES	foreign key to dim_resource

Table fact_ticket

The base table for tickets.

Column	Type	Nullable	Comment
ticket_id	bigint(20)	NO	primary key
ticket_uid	varchar(255)	NO	corresponding unique key from CM
contact_count	int(11)	YES	Number of contacts this ticket has
create_date	datetime	YES	timestamp when object was created in CM
create_date_id	int(11)	YES	foreign key to dim_date
create_time_id	int(11)	YES	foreign key to dim_time

Column	Type	Nullable	Comment
end_date	datetime	YES	Timestamp when ticket was technically closed
end_date_id	int(11)	YES	foreign key to dim_date
end_time_id	int(11)	YES	foreign key to dim_time
firstcontact_date	datetime	YES	Timestamp of first response on ticket, must be manually set in workflow-action.
firstcontact_date_id	int(11)	YES	foreign key to dim_date
firstcontact_time_id	int(11)	YES	foreign key to dim_time
has_attachments	bit(1)	YES	Shows if ticket has attachments.
has_dest_references	bit(1)	YES	Shows if ticket is referenced by other tickets.
has_src_references	bit(1)	YES	Shows if ticket is referencing other tickets.
name	varchar(255)	YES	internal name
status_date	datetime	YES	Timestamp until when ticket in waiting status
status_date_id	int(11)	YES	foreign key to dim_date
status_time_id	int(11)	YES	foreign key to dim_time
subject	varchar(255)	YES	Subject of the ticket
activity_id	bigint(20)	YES	foreign key to dim_activity
contact_id	bigint(20)	YES	foreign key to dim_contact
engineer_id	bigint(20)	YES	foreign key to dim_engineer
queue_id	bigint(20)	YES	foreign key to dim_queue
scope_id	bigint(20)	YES	foreign key to dim_scope

Table fact_ticket_activity_chg

This table contains the time intervals between activity changes.

Column	Type	Nullable	Comment
ticket_activity_chg_id	bigint(20)	NO	primary key
ticket_activity_chg_uid	varchar(255)	NO	corresponding unique key from CM
insert_datetime	datetime	YES	timestamp when entry was created in CM
insert_date_id	int(11)	YES	foreign key to dim_date
insert_time_id	int(11)	YES	foreign key to dim_time
time_brutto	bigint(20)	YES	time interval between this log and the last log (of the same type) - w/o workflow calendar [seconds]
time_netto	bigint(20)	YES	time interval between this log and the last log (of the same type) - with workflow calendar [seconds]

Column	Type	Nullable	Comment
executor_id	bigint(20)	YES	engineer who performed the action. foreign key to dim_engineer. Empty if automatic activity.
ticket_id	bigint(20)	YES	foreign key to fact_ticket
activity_id	bigint(20)	YES	foreign key to dim_activity
activity_prev_id	bigint(20)	YES	value from previous log entry

Initially calculated from cmas_ticket and cmas_ticket_log for TicketLogType.ACTIVITY_CHANGE.

Please note that this fact table should contains only records for real changes, there should not be situation that activity_tk and activity_tk_prev are the same. But such change should be insert in fact_ticket_log.

Table fact_ticket_contact

This table contains the contacts for each ticket, and their respective roles (within this ticket)..

Column	Type	Nullable	Comment
contact_id	bigint(20)	NO	foreign key to dim_contact
ticket_id	bigint(20)	NO	foreign key to fact_ticket
role_id	bigint(20)	YES	foreign key to dim_contact_role

Table fact_ticket_engineer_chg

This table contains the time intervals between engineer changes.

Column	Type	Nullable	Comment
ticket_engineer_chg_id	bigint(20)	NO	primary key
ticket_engineer_chg_uid	varchar(255)	NO	corresponding unique key from CM
insert_datetime	datetime	YES	timestamp when entry was created in CM
insert_date_id	int(11)	YES	foreign key to dim_date
insert_time_id	int(11)	YES	foreign key to dim_time
time_brutto	bigint(20)	YES	time interval between this log and the last log (of the same type) - w/o workflow calendar [seconds]
time_netto	bigint(20)	YES	time interval between this log and the last log (of the same type) - with workflow calendar [seconds]
executor_id	bigint(20)	YES	engineer who performed the action. foreign key to dim_engineer. Empty if automatic activity.
ticket_id	bigint(20)	YES	foreign key to fact_ticket
engineer_id	bigint(20)	YES	foreign key to dim_engineer
engineer_prev_id	bigint(20)	YES	value from previous log entry

Initially calculated from cmas_ticket and cmas_ticket_log for TicketLogType.ENGINEER_CHANGE

Please note that this fact table should contains only records for real changes, there should not be sitiation that enginner_tk and engineer_tk_prev are the same. But such change should be insert in fact_ticket_log.

Table fact_ticket_engineer_user

Relation of ticket to engineers

Column	Type	Nullable	Comment
ticket_engineer_user_id	bigint(20)	NO	primary key
ticket_engineer_user_uid	varchar(255)	NO	corresponding unique key from CM
engineer_uid	varchar(255)	YES	transfer key of dim_engineer
function_uid	varchar(255)	YES	uid of dim_ticket_function
note	longtext	YES	additional text
ticket_uid	varchar(255)	YES	corresponding unique key from CM
engineer_id	bigint(20)	YES	foreign key to dim_engineer
function_id	bigint(20)	YES	foreign key to dim_ticket_function
ticket_id	bigint(20)	NO	foreign key to fact_ticket

Note: Contains only the engineers which have been asigned a specific role/function is the ticket.

This must not include the current assigned engineer.

Table fact_ticket_log

Entries of ticket history taken from cmas_ticket_log. Deprecated. fact_ticket_..._chg should be used instead.

Column	Type	Nullable	Comment
ticket_log_id	bigint(20)	NO	primary key
ticket_log_uid	varchar(255)	NO	corresponding unique key from CM
action	varchar(255)	YES	Name of action from dim_action
activity_name	varchar(255)	YES	Name of the activity
activity_uid	varchar(255)	YES	corresponding unique key from CM
engineer_name	varchar(255)	YES	login name of the engineer
engineer_uid	varchar(255)	YES	transfer key of dim_engineer
executor_name	varchar(255)	YES	Name of the executor (Engineer) or technical user/component
executor_uid	varchar(255)	YES	Reference to dim_engineer.engineer_uid. Empty if automatic activity.
insert_datetime	datetime	YES	timestamp when entry was created in CM
insert_date_id	int(11)	YES	foreign key to dim_date
insert_time_id	int(11)	YES	foreign key to dim_time

Column	Type	Nullable	Comment
handletime_brutto	bigint(20)	YES	Difference to previous log entry (insert_datetime) - w/o workflow calendar [seconds]
handletime_netto	bigint(20)	YES	Difference to previous log entry (insert_datetime) - with workflow calendar [seconds]
activity_prev_name	varchar(255)	YES	value from previous log entry
activity_prev_uid	varchar(255)	YES	value from previous log entry
engineer_prev_name	varchar(255)	YES	value from previous log entry
engineer_prev_uid	varchar(255)	YES	value from previous log entry
queue_prev_uid	varchar(255)	YES	value from previous log entry
scope_prev_uid	varchar(255)	YES	value from previous log entry
queue_name	varchar(255)	YES	name of queue
queue_uid	varchar(255)	YES	corresponding unique key from CM
scope_uid	varchar(255)	YES	corresponding unique key from CM
status_datetime	datetime	YES	timestamp when actitivity was executed in CM
status_date_id	int(11)	YES	foreign key to dim_date
status_time_id	int(11)	YES	foreign key to dim_time
ticket_uid	varchar(255)	YES	corresponding unique key from CM
activity_id	bigint(20)	YES	foreign key to dim_activity
engineer_id	bigint(20)	YES	foreign key to dim_engineer
executor_id	bigint(20)	YES	engineer who performed the action. foreign key to dim_engineer. Empty if automatic activity.
activity_prev_id	bigint(20)	YES	value from previous log entry
engineer_prev_id	bigint(20)	YES	value from previous log entry
queue_prev_id	bigint(20)	YES	value from previous log entry
scope_prev_id	bigint(20)	YES	value from previous log entry
queue_id	bigint(20)	YES	foreign key to dim_queue
scope_id	bigint(20)	YES	foreign key to dim_scope
ticket_id	bigint(20)	YES	foreign key to fact_ticket

It contain the activity change information. The "..._prev_.. columns" contains the values of the previous ticket state.

This table is very large as it is denormalized. It also contains the name of referenced objects.

Table fact_ticket_queue_chg

This table contains the time intervals between queue changes.

Column	Type	Nullable	Comment

Column	Type	Nullable	Comment
ticket_queue_chg_id	bigint(20)	NO	primary key
ticket_queue_chg_uid	varchar(255)	NO	corresponding unique key from CM
insert_datetime	datetime	YES	timestamp when entry was created in CM
insert_date_id	int(11)	YES	foreign key to dim_date
insert_time_id	int(11)	YES	foreign key to dim_time
time_brutto	bigint(20)	YES	time interval between this log and the last log (of the same type) - w/o workflow calendar [seconds]
time_netto	bigint(20)	YES	time interval between this log and the last log (of the same type) - with workflow calendar [seconds]
executor_id	bigint(20)	YES	engineer who performed the action. foreign key to dim_engineer. Empty if automatic activity.
ticket_id	bigint(20)	YES	foreign key to fact_ticket
queue_prev_id	bigint(20)	YES	value from previous log entry
queue_id	bigint(20)	YES	foreign key to dim_queue

Initially calculated from cmas_ticket and cmas_ticket_log for TicketLogType.QUEUE_CHANGE.

Please note that this fact table should contains only records for real changes, there should not be situation that queue_tk and queue_tk_prev are the same. But such change should be insert in fact_ticket_log.

Table fact_ticket_relation

This table contains the time intervals between queue changes.

Column	Type	Nullable	Comment
ticket_relation_id	bigint(20)	NO	primary key
ticket_relation_uid	varchar(255)	NO	corresponding unique key from CM
note	longtext	YES	additional text
type	varchar(255)	YES	type of relation: MASTER_SLAVE, REFERENCE, PARENT_CHILD
source_ticket_id	bigint(20)	YES	primary key for source
target_ticket_id	bigint(20)	YES	primary key for target

Table fact_ticket_time

Sums of all relevant ticket activity times.

Column	Type	Nullable	Comment
ticket_time_id	bigint(20)	NO	foreign key to dim_time
ticket_time_uid	varchar(255)	NO	corresponding unique key from CM

Column	Type	Nullable	Comment
reactiontime_brutto	bigint(20)	YES	difference between open_date and firstcontact_date - w/o workflow calendar [seconds]
reactiontime_netto	bigint(20)	YES	difference between open_date and firstcontact_date - with workflow calendar [seconds]
totaltime_brutto	bigint(20)	YES	difference between open_date and closing of ticket [seconds]
totaltime_netto	bigint(20)	YES	difference between open_date and closing of ticket, with workflow calendar [seconds]

All times are calculated as follows: when a ticket time is updated, the current transaction entry is taken and the ticket time added to the existing time.

Brutto time: simply subtract the timestamps

Netto time: only count the time within working hours (using CM workflow calendar functions and the current workflow of the ticket)

Table fact_time_booking

This table contains the facts about time bookings.

Column	Type	Nullable	Comment
time_booking_id	bigint(20)	NO	primary key
time_booking_uid	varchar(255)	NO	corresponding unique key from CM
time_booking_date	datetime	YES	timestamp when the time booking starts
time_booking_date_id	int(11)	YES	foreign key to dim_date
time_booking_time_id	int(11)	YES	foreign key to dim_time
description	varchar(255)	YES	description
duration	bigint(20)	NO	duration of time booking [milliseconds]
start_time_set	bit(1)	NO	indicates whether the start time was set for this booking.
engineer_id	bigint(20)	YES	foreign key to dim_engineer
project_id	bigint(20)	YES	ID of the project (dim_project.project_id)
ticket_id	bigint(20)	YES	foreign key to fact_ticket

Table fact_unit_log

Contact/Unit history

Column	Type	Nullable	Comment
unit_log_id	bigint(20)	NO	primary key
unit_log_uid	varchar(255)	NO	corresponding unique key from CM
action	varchar(255)	YES	Name of action from dim_action

Column	Type	Nullable	Comment
executor_name	varchar(255)	YES	Name of the executor (Engineer) or technical user/component
executor_uid	varchar(255)	YES	Reference to dim_engineer.engineer_uid. Empty if automatic activity.
insert_datetime	datetime	YES	timestamp when entry was created in CM
insert_date_id	int(11)	YES	foreign key to dim_date
insert_time_id	int(11)	YES	foreign key to dim_time
handletime_brutto	bigint(20)	YES	Difference to previous log entry (insert_datetime) - w/o workflow calendar [seconds]
handletime_netto	bigint(20)	YES	Difference to previous log entry (insert_datetime) - with workflow calendar [seconds]
unit_uid	varchar(255)	YES	tranfer key of dim_contact
executor_id	bigint(20)	YES	engineer who performed the action. foreign key to dim_engineer. Empty if automatic activity.
unit_id	bigint(20)	YES	foreign key to dim_contact

Table hlp_calendar

List of available custom calenders.

Column	Type	Nullable	Comment
calendar_id	bigint(20)	NO	primary key
calendar_uid	varchar(255)	NO	corresponding unique key from CM
name	varchar(255)	NO	internal name
timezone	varchar(255)	NO	Timezone of the calender

Table hlp_calendar_day

Week days which are workdays in the corresponding calender

Column	Type	Nullable	Comment
calendar_day_id	bigint(20)	NO	primary key
calendar_day_uid	varchar(255)	NO	corresponding unique key from CM
week_day	int(11)	NO	Id of week day (1: Sunday, 2: Monday, [...], 6: Friday, 7: Saturday)
calendar_id	bigint(20)	YES	Id of the calendar (hlp_calendar.id)

Table hlp_calendar_day_part

Working hours for the workdays of the calenders

Column	Type	Nullable	Comment
calendar_day_part_id	bigint(20)	NO	primary key
calendar_day_part_uid	varchar(255)	NO	corresponding unique key from CM
from_hour	int(11)	NO	start hour
from_minute	int(11)	NO	start minute
to_hour	int(11)	NO	end hour
to_minute	int(11)	NO	end minute
day_part_id	bigint(20)	YES	Foreign key to hlp_calendar_day

Table hlp_calendar_holiday

Holiday in the corresponding calenders

Column	Type	Nullable	Comment
calendar_holiday_id	bigint(20)	NO	primary key
calendar_holiday_uid	varchar(255)	NO	corresponding unique key from CM
from_day	datetime	YES	Start day
name	varchar(255)	YES	internal name
to_day	datetime	YES	End day. Empty for single day holidays.
calendar_id	bigint(20)	YES	Id of the calendar (hlp_calendar.id)

Table hlp_notification

Log of notifications sent by email.

Column	Type	Nullable	Comment
notification_id	int(11)	NO	primary key
description	varchar(2000)	NO	description
mail_from	varchar(255)	NO	outgoing email-messages from address
host	varchar(255)	NO	outgoing mail-server host
notification_type	int(11)	NO	Enumerated value
password	varchar(255)	YES	outgoing mail-server password
port	int(11)	NO	outgoing mail-server port
protocol	varchar(255)	NO	

Column	Type	Nullable	Comment
subject	varchar(255)	NO	outgoing email-messages - subject
mail_to	varchar(255)	NO	outgoing email-messages to address
username	varchar(255)	YES	outgoing mail-server username

There are three types of notifications:

ERROR

FINISHED_SUCCESSFULLY

FINISHED_UNSUCCESSFULLY

In case something is going wrong during work of cmrf, email-notifications with error description are sent to the emails configured in the admin tool.

Table hlp_parameter

These parameters show configuration for filling static dimension tables.

Column	Type	Nullable	Comment
parameter_id	int(11)	NO	primary key
parameter_description	varchar(2000)	YES	More information for the parameter
parameter_name	varchar(50)	YES	internal name
parameter_value	varchar(255)	YES	Value of parameter as string

The Parameters contains

- start and end dates for table dim_date

- format strings for filling string fields in dim_date (see dime_date)

- status of DWH. For description of the status see Section "CMDB / CMRF/ Data Warehouse Synchronization Process" in ConSol CM Operations Manual.

- last date transfer.

Table hlp_queue_client

The table contains relation between client groups and queues.

Column	Type	Nullable	Comment
client_group_id	bigint(20)	NO	primary key
queue_id	bigint(20)	NO	foreign key to dim_queue

Table hlp_transfer_error

For storing any error messages of exceptions occuring during a data transfer or update.

Column	Type	Nullable	Comment
transfer_error_id	int(11)	NO	primary key

Column	Type	Nullable	Comment
error_msg	varchar(255)	YES	(Optional) error message
error_time	datetime	NO	Timestamp when the error occurred
type	varchar(255)	NO	Object type
object_uid	varchar(255)	YES	uid of object for which transfer failed

Entries are never deleted.

D - Appendix

This section contains several appendices:

- Trademarks
- Glossary

D.1 Glossary

B

BI

Business Intelligence - methods, technologies, and architectures to transform data into useful information for business purposes.

C

CMDB

ConSol CM database - the working database of the CM system.

CMRF

ConSol CM Reporting Framework - a JEE application which synchronizes data between the ConSol CM database and the DWH.

company

The company is the upper hierarchical level of a two-level customer model. A company can have several contacts.

contact

The contact is the lower hierarchical level of a two-level customer model. A contact can only belong to one company.

CRM

Customer Relationship Management. Approach to manage a company's customers, e.g., to collect data from different sources and integrate the data to generate information which allows, e.g., to optimize the services for the customers.

Custom Field

A field where ticket data can be stored.

Custom Field Group

A group of Custom Fields where ticket data can be stored.

customer

The customer represents the external side of a ticket. It designates the person or object that gave the reason for creating a ticket. A customer can either be a company or a contact.

D

Dashboard

A type of report which integrates data from different sources providing an overall perspective of a certain topic. Often times graphical representation is used.

DWH

Data Warehouse - A database used for reporting and data analysis. In a standard ConSol CM distribution, a DWH is included and only has to be installed and configured.

E

engineer

Engineers are the users who work on the tickets in the Web Client

ERP system

Enterprise Resource Planning - often used for this type of enterprise management software.

ETL

Extract Transform Load - extracts data from one source (a database or other source), transforms it, and loads it into a database, e.g., a data warehouse.

K

KPI

Key Performance Indicator - parameter used for performance measurement for companies, projects, etc.

M

main customer

The main customer is the customer who gave the reason for creating the ticket. The main customer is mandatory for a ticket.

P

Pentaho

PentahoTM is a business intelligence (BI) suite which is available in open source and as enterprise editions.

R

RDBMS

Relational Database Management System - e.g. Oracle ® , MS SQL Server ® , MySQL.

T

ticket

The ticket is the request of the customer which the engineer works on. It is the object which runs through the business process defined by the workflow.

U

Unit

Java class which represents a customer object. i.e. a contact is an object of class Unit and a company is also an object of class Unit.

W

workflow

The workflow is the implementation of the business process managed in ConSol CM. It contains a series of steps which are carried out by the engineers.

D.2 Trademarks

- The Apache Commons Codec ™ library is a trademark of the Apache Software Foundation. See Apache Commons Codec web page.
- Apache OpenOffice™ – Apache and the Apache feather logos are trademarks of The Apache Software Foundation. OpenOffice.org and the seagull logo are registered trademarks of The Apache Software Foundation. See Apache OpenOffice Trademarks web page.
- Google Maps™ – Google Maps is a trademark of Google Inc. See Google trademark web page for details.
- Microsoft® – Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page.
- Microsoft® Active Directory® – Microsoft and Microsoft Active Directory are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page.
- Microsoft® Exchange Server – Microsoft and Microsoft Exchange Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page.
- Microsoft® Office – Microsoft and Microsoft Office are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page.
- Windows® operating system – Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page.
- Microsoft® SQL Server® – Microsoft and Microsoft SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page.
- Microsoft® Word® – Microsoft and Microsoft Word are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. See Microsoft trademark web page.
- MuleSoft™ and Mule ESB™ are among the trademarks of MuleSoft, Inc. See Mule Soft web page.
- Oracle® – Oracle is a registered trademark of Oracle Corporation and/or its affiliates. See Oracle trademarks web page.
- Oracle® WebLogic – Oracle is a registered trademark of Oracle Corporation and/or its affiliates. See Oracle trademarks web page.
- Pentaho® – Pentaho and the Pentaho logo are registered trademarks of Pentaho Inc. See Pentaho trademark web page.