# Using Metadata to Create Your Own Service Desk Database Schema Map

***Warning: The purpose of this document is to instruct the reader to build their own database schema to assist in creating third party reporting. It should not be used to create operations that bypass the programming of the Service Desk application. Updating data directly in the database poses serious risks and is not supported by LANDesk or its partners.***

## The Challenge of Releasing the Database Schema.

### Summary:

1. Developers are cautious against releasing the database schemas.
2. Complex systems include metadata to define their schemas.
3. Knowing metadata and how it works will be the best solution to understanding Service Desk schema.

A conceptual example of databases and relationships that involves a small number of tables helps to understand database structures. However, one quickly finds that the number of tables and relationships in a production database system far exceed what a simple conceptual description.

The number of tables and relationships in a production system are too large to map in a timely manner. Trying to figure which tables relate to one another can feel like an overwhelming task. For this reason, it is common to see requests for a schema diagram that shows the tables and their relationships. This is especially true with Service Desk. There exists an understanding that tools such as Crystal Reports© will be used for enhanced reporting of Service Desk data. This implies a need for a database schema.

As common as this request may be, it is equally uncommon for developers of database oriented systems to release their schema as part of official documentation. Such documentation poses a risk from individuals that would attempt to manipulate the data via the database rather than through the program. This bypasses protections from data corruptions. Additionally, future upgrades and enhancements may require database schemas to be modified from what is published. This means that publication of this documentation requires continued maintenance. Many organizations lack resources to keep this maintained in a timely manner. Beyond this, there is a question about releasing database structures to individuals that may not fully understand them and whether such releases are helpful.

The more complex a database structure, the more likely development includes tables that define the data in the database. Data that defines the rest of the data is called metadata. In service desk metadata tables have the precursor md_ in their name. A better solution for those that truly need the schema would be to understand the metadata and use it to retrieve the necessary structure.

There remains a challenge in figuring out the relationship between the metadata tables, but is much smaller and easier to map. Service Desk makes this process easier with some well-defined naming conventions.

## Overview of Service Desk Database Naming Conventions

### Summary:

1. -Each table name has a prefix that corresponds to its module or if custom created, it has a "usr_" prefix.
2. -Each Table will have a primary key column. That column name usually includes a prefix of its module and the term "guid" as in "im_guid."
3. -Some tables inherit from the Process object and their primary key column is named "pm_id."
4. Any column that is a foreign key will be named in such a way as to assist in defining the table/column it references.
5. Using this principles and focusing on metadata ("md_" tables), we can begin to assemble queries that will define the schema.

Service Desk uses different modules to organize business objects and their use. The tables are named by giving prefixes that correspond to these modules. For example, incidents are part of the Incident Management module. Tables in this module use the prefix "im_" as their identifier. The only exception to the naming of tables based on the modules is when a new business object is created that is not part of what is known as the out-of-the-box configuration. Objects created beyond the out-of-the-box configuration are given the prefix "usr_" as their identifier. Below are a list of the commonly used modules and the table prefixes.

cm_ Change

em_ Email

hr_ Human Resources

im_ Incident Management

km_ Knowledge Management

lc_ LifeCycle

md_ Meta Data

mi_ Management Information

pm_ Process Management

pr_ Problem Management

tps_ Touchpaper Services

ui_ User Interface

usr_ Any Business Object or Attribute that has been created beyond the out of the box configuration. Usr_ indicates that it is created by a user.

Each table has a column with a primary key. The primary key column is named by using the modules prefix, followed by an underscore, followed by "guid." Therefore, tables under Incident Management will have a column named "im_guid." There is an exception to this, a number of business objects are actually inherited from a parent business object. The most significant parent is the Process business object. If you go into the Service Desk console, and open the Object Designer, and navigate into Process Management/Process, you will see that the most commonly used objects in service desk exist as children of Process. These include Incident, Problem, Change, and Request.

The objects listed underneath Process all use processes as part of their operations. They are process related objects. What matters for our needs is that this affects the naming convention mentioned above. Objects that are inherited from a parent use the parent's guid as their unique identifier. This means that the primary objects for the most commonly used modules (Incident, Problem, Change, and Request) actually do not follow the normal naming convention for their primary keys. Instead, they use the prefix of Process Management followed by "id" or in other words "pm_id."

To summarize, tables will have either a primary key named with the module prefix followed by "guid" or the parent module prefix followed by "id" depending on the object and whether or not it is inheriting from a parent. This means it should be fairly simple to look at any object and know which column is holding the primary key.

Just as a primary key follows a naming convention, the foreign keys of any table also use a naming convention. This helps identify the source of what it the foreign key references. The naming convention involves the module prefix, followed by the table or object type, followed by "guid." This helps to assemble the relationships of tables to other tables.

With this understanding of table naming and primary key naming and foreign keys naming, we can then focus our attention on metadata tables. The metadata tables follow the same naming conventions. However, the metadata tables define the relationships of all of the other tables and their columns to one another. This means we already have the essential elements to map our database schema.

# Examining the Metadata

## Summary:

1. md_database_tables lists all tables used in Service Desk.
2. md_database_columns lista the columns used in each of the Service Desk tables.
3. md_module lists each module.
4. A foreign key exists in md_database_tables to relate a table to its module.
5. A foreign key exists in md_database_columns to relate to each column to its table.
6. md_foreign_key_column lists all relationships between tables/columns to other tables/columns.
7. Using a series of join statements, the relationships between each table/column to other tables/columns can be listed.

The metadata tables provide descriptive enough names to get started and enough information to complete our schema task. A scan through the md_ tables reveals tables to define modules (md_modules), database tables (md_database_table), database columns for those tables (md_database_column  A query joining these tables together should produce a list of each table and the columns in that table and the module associated with the table. The md_dababase_table includes a foreign key column to its related modules guid.  This matches to the primary key of each record in the md_module table. The md_database_column table has a foreign key column related to the primary key of md_database_table. This should allow us to write a query that includes join statements to connect these columns together.

```
SELECT mdl.md_title as ModuleName,
tbl.md_name as TableName,
col.md_name as ColumnName
FROM md_database_table tbl
LEFT JOIN md_module mdl on tbl.md_module_guid = mdl.md_guid
Left Join md_database_column col on tbl.md_guid = col.md_database_table_guid
Order by tbl.md_name
```

The results of this query will be a list of all tables, their related modules, and the columns found in each table.

The relationship of tables to other tables is also documented in the metadata tables. The tables md_foreign_key_column gives us the foreign key guid, plus source table/column and the target/column which provides us with sufficient information to establish a list of all relationships.  We first add the md_foreign_key_column to the md_database_column table.

```
SELECT mdl.md_title as ModuleName,
tbl.md_name as TableName,
col.md_name as ColumnName
FROM md_database_table tbl
LEFT JOIN md_module mdl on tbl.md_module_guid = mdl.md_guid
Left Join md_database_column col on tbl.md_guid = col.md_database_table_guid
LEFT JOIN md_foreign_key_column for1 on col.md_guid =
for1.md_src_database_column_guid
Order by tbl.md_name
```

Next the md_database_table and md_database_column join a second time. They join them to target foreign key columns in the md_foreign_key_column table.

```
SELECT mdl.md_title as ModuleName,
tbl.md_name as TableName,
col.md_name as ColumnName,
tgtTable.md_name RelatedTable,
tgtCol.md_name RelatedColumn
FROM md_database_table tbl
LEFT JOIN md_module mdl on tbl.md_module_guid = mdl.md_guid
LEFT JOIN md_database_column col on tbl.md_guid = col.md_database_table_guid
LEFT JOIN md_foreign_key_column for1 on col.md_guid = for1.md_src_database_column_guid
LEFT JOIN md_database_column tgtCol on for1.md_tgt_database_column_guid =
tgtCol.md_guid
LEFT JOIN md_database_table tgtTable on for1.md_tgt_database_table_guid =
tgtTable.md_guid
Order by tbl.md_name
```

In the query above, md_database_table and md_database_column are joined a second time to the query as relationships to md_foreign_key_column's target column and target table foreign keys. This establishes the relationship of tables/columns to other tables/columns. When completed, the result is a comprehensive list of every table, the module to which it belongs, the columns it contains, and the relationships it has two other tables and columns. This constitutes a basic database schema.

A slight modification to this query by including an inner join with the md_foreign_key table will produce a list of only tables that have relationships to other tables.

```
SELECT mdl.md_title as ModuleName,
tbl.md_name as TableName,
col.md_name as ColumnName,
tgtTable.md_name RelatedTable,
tgtCol.md_name RelatedColumn
FROM md_database_table tbl
LEFT JOIN md_module mdl on tbl.md_module_guid = mdl.md_guid
Left Join md_database_column col on tbl.md_guid = col.md_database_table_guid
```

LEFT JOIN md_foreign_key_column for1 on col.md_guid = for1.md_src_database_column_guid

Left JOIN md_database_column tgtCol on for1.md_tgt_database_column_guid = tgtCol.md_guid

LEFT JOIN md_database_table tgtTable on for1.md_tgt_database_table_guid = tgtTable.md_guid

**INNER JOIN md_foreign_key fkey on for1.md_guid = fkey.md_guid**

Order by tbl.md_name

The query above creates an inner join between md_foreign_key and md_foreign_key_column. The inner join assure that only the tables/columns and actually have relationships to other tables/columns display.

## Reminders

It is important to note that this query structure comes from a specific version of Service Desk. It cannot be considered authoritative for all versions Service Desk. However, this structural design is much more likely to remain intact and include any new relationships that occur in future releases. One must make sure to test this query and re-examine the tables with each new release to ensure that the structure used in these queries will continue to be valid in future releases.

It is equally important to remind the reader that the purpose of this is to assist in writing queries for reporting purposes. Using queries to update tables without going through the Service Desk program, is very dangerous as any damage caused by it cannot be supported by LANDesk technical support or the technical support of any of its channel partners.