Centric CRM 5.0

Database Persistence

Concursive

Author:	Matt Rajkowski
Creation Date:	July 11, 2003
Last Updated:	November 2, 2007
Version:	3.0

Document Control

Change Record

Date	Author	Version	Change Reference
07/14/03	Matt Rajkowski	1.0	No previous document
7/16/03	Matt Rajkowski	1.1	Added tables regarding Tasks and Communications Manager
10/11/04	Matt Rajkowski	2.0	CRM 2.9
11/02/07	Lorraine Bittner	3.0	CRM 5.0

Reviewers

Name	Position

Distribution

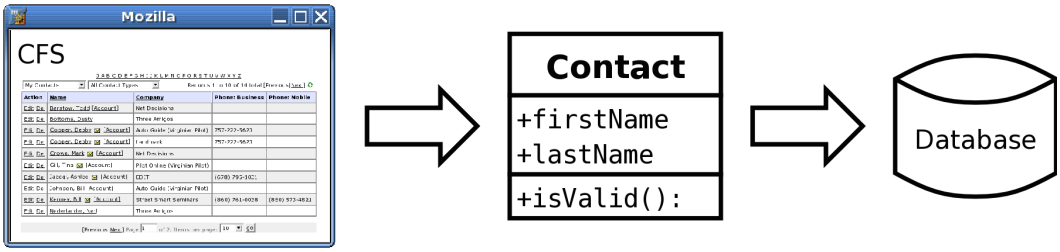
Name	Location
Centric CRM Community	http://www.centriccrm.com/Portal.do?key=community

Contents

I. Introduction

Centric CRM is made up of modules, business rules, objects and data. The modules are application code that have been programmed to manipulate objects based on business rules. The business rules are logical constraints based on specifications that cannot be expressed in the schema.

So, what that means is that modules have objects, and that objects represent real-world things, like a person, that get stored in a database.



A. Overview of Document

1. Scope

This document describes the physical data structure of Centric CRM. The modules and business rules will only be briefly described as they relate to the tables.

It is assumed that a Centric CRM database has been created and that the developer is familiar with databases and SQL. The developer should also have the Centric CRM database create scripts and a recent HTML schema for reference.

Any references to SQL scripts in this document may be abbreviated with columns removed.

2. Objectives

This document will accomplish the following objectives:

a) Identify core database tables that are used directly by the framework to support features like Login, Workflow, and Configuration
b) Identify key module database tables, tables that are used by the modules

B. Overview of Environment

1. Database server independent

Centric CRM is a platform and database independent web application licensed under the Dark Horse Ventures Open Code License.

Centric CRM works well with PostgreSQL and Microsoft SQL Server, and should work with any SQL-92 compliant database with the proper database creation scripts.

The database makes use of TABLES, INDEXES, SEQUENCES, REFERENTIAL INTEGRITY and DEFAULT VALUES. Indexes have been created for optimizing sorted queries in CFS, while sequences are used as PRIMARY KEYS for identifying unique records. References between tables ensure that data will not be lost in the database and forces the application to contain business rules for manipulating data.

The database ***does not*** use VIEWS or STORED PROCEDURES. The benefit of doing things this way is that the schema is given room to grow and change while still allowing the core application logic to be abstracted from the database logic.

2. Accessing the database

The database should always be accessed and manipulated through the Centric CRM application. All relevant pieces of information are available through the application interfaces.

The database itself does not protect against data corruption that could be caused by other applications.

3. Table conventions

Table names are lowercase and without any spacing. The table name should start with the module or function that it is associated with. For example, the tables for the project management module all begin with "project_" and lookup tables begin with "lookup_".

The field names should be consistent between tables. Each table should have a descriptive primary key, like "user_id" to simplify joins. Also, field names are lowercase and without any spacing.

Most tables are a representation of a Java class, and therefore, each record in the table represents an instance of a Java object.

Tables that reflect objects typically have the following base fields:

a) **id** – which represents a unique record

b) **entered** – which is a timestamp of when the record was created
c) **enteredby** – which identifies the user that created the record
d) **modified** – which is a timestamp of when the record was last modified
e) **modifiedby** – which identifies the user that last modified the record
f) **owner** – which identifies the user that owns the record, if applicable
g) **enabled** – specifies if this record is hidden, possibly because it is marked for archival or deletion

Most tables also use lookup tables for fields which reuse descriptions. A typical lookup table has the following fields:

h) **code** SERIAL PRIMARY KEY
i) **description** VARCHAR(50) NOT NULL,
j) **default_item** BOOLEAN DEFAULT false – in a list is this item the default choice
k) **level** INTEGER DEFAULT 0 – the order in which the item appears in a list
l) **enabled** BOOLEAN DEFAULT true – is this a valid entry or has it been deleted

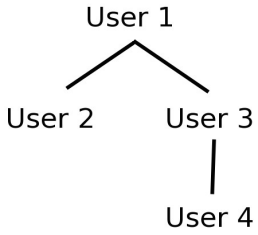
II. Core Tables

In Centric CRM, there is an underlying framework of permissions that controls how users can interact with the system. Users (1) must be a member of the system, (2) must be in a role in which permissions can are added, and (3) can be in a hierarchy which permits access to specific records.

In general, a user's role specifies which modules and which features within a module a user has permission or access to.

Role	
Sales Manager	Salesperson
User 1	User 2 User 3 User 4

A user's hierarchy is simply a reporting structure that defines who this user reports to, and which users report to this user. Records can then be controlled by hierarchy.



A. [access]

The access table lists the users of the system and their login information.

Almost every table will refer back to the **user_id** in this table for tracking purposes.

Passwords are one-way encrypted by the application and cannot be decrypted.

Each user must have a **contact_id** which maps to [contact].

Each user must have a **role_id** which maps to [role] to define their permissions.

Each user can optionally have a **manager_id.** This field allows for hierarchies to be built in an organization. Hierarchies are important throughout the modules for allowing access to records.

If the user is NOT **enabled**, then that user cannot login. Also, if **expires** is today or in the past, then that user cannot login.

```
CREATE TABLE access (
  user_id INTEGER DEFAULT nextval('access_user_id_seq') NOT NULL
PRIMARY KEY,
  username VARCHAR(80) NOT NULL,
  password VARCHAR(80),
  contact_id INT DEFAULT -1,
  role_id INT DEFAULT -1,
  manager_id INT DEFAULT -1,
  last_ip VARCHAR(15),
  last_login TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL,
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  expires date DEFAULT NULL,
  enabled boolean NOT NULL DEFAULT true
);
```

B. [permission_category]

Each feature in the application ultimately falls under a specific module. These module names are stored in [permission_category].

Category (Module)
My Home Page
General Contacts
Accounts

Each category can have a longer **description** to explain the broad **category** name.

If the category is NOT **enabled** then the module will not be available to anyone, even if the user has the permission enabled.

Several fields describe additional functionality that the module might have:

1. **folders** – this module supports the "Folders" feature
2. **lookups** – this module supports the "Lookup Editor" feature
3. **viewpoints** – this module supports the "Viewpoints" feature
4. **categories** – this module supports the "Category Editor" feature
5. **scheduled_events** – this module supports the "Scheduled Events" feature

6. **object_events** – this module supports the "Object Events" feature

For example, the Trouble Ticket module has **lookups** that can be edited by a user and it supports the **categories** editor.

```
CREATE TABLE permission_category (
  category_id INTEGER DEFAULT
nextval('permission_cate_category_id_seq') NOT NULL PRIMARY KEY,
  category VARCHAR(80),
  description VARCHAR(255),
  level INT NOT NULL DEFAULT 0,
  enabled boolean NOT NULL DEFAULT true,
  active boolean NOT NULL DEFAULT true,
  folders boolean NOT NULL DEFAULT false,
  lookups boolean NOT NULL DEFAULT false,
  viewpoints BOOLEAN DEFAULT false,
  categories BOOLEAN DEFAULT false,
  scheduled_events BOOLEAN DEFAULT false,
  object_events BOOLEAN DEFAULT false
);
```

C. [permission]

The permission table lists ALL of the permissions that are available in the application, as defined by the application developer. These are general module or feature permissions and are NOT record-level permissions.

Category	*Permission*	*View*	*Add*	*Edit*	*Delete*
Account Management	Access to Account Management Module	Yes	n/a	n/a	n/a
	Account Records	Yes	Yes	Yes	Yes

The **permission** field is a unique name that corresponds to code in each module. Each permission can have the following capability:

1. **permission_view** – code in this module is restricted to those that have view access, this might restrict access to the module entirely, or the capability to view or access something in the module
2. **permission_add** – code in this module is restricted to those that have add access, typically this relates to adding records
3. **permission_edit** – code in this module is restricted to those that have edit access, typically this relates to editing records
4. **permission_delete** – code in this module is restricted to those that have delete access, typically this relates to deleting records

```
CREATE TABLE permission (
  permission_id SERIAL PRIMARY KEY,
  category_id INT NOT NULL REFERENCES permission_category,
  permission VARCHAR(80) NOT NULL,
  permission_view BOOLEAN NOT NULL DEFAULT false,
```

```
  permission_add BOOLEAN NOT NULL DEFAULT false,
  permission_edit BOOLEAN NOT NULL DEFAULT false,
  permission_delete BOOLEAN NOT NULL DEFAULT false,
  description VARCHAR(255) NOT NULL DEFAULT '',
  level INT NOT NULL DEFAULT 0,
  enabled BOOLEAN NOT NULL DEFAULT true,
  active BOOLEAN NOT NULL DEFAULT true,
  viewpoints BOOLEAN DEFAULT false
);
```

D. [role]

Each user in the system has a role.

This table defines what a **role** is called and a **description** of what it might be used for.

Related to this table is [role_permission] in which allowed permissions are stored.

```
CREATE TABLE role (
  role_id SERIAL PRIMARY KEY,
  role VARCHAR(80) NOT NULL,
  description VARCHAR(255) NOT NULL DEFAULT '',
  enteredby INT NOT NULL REFERENCES access(user_id),
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enabled boolean NOT NULL DEFAULT true
);
```

E. [role_permission]

Each role permission specifies a **permission_id** and whether this role has **role_view**, **role_add**, **role_edit**, and **role_delete** for the specified permission .

The following shows the permissions the system is capable of and the actual permissions the role has enabled.

Category	*Permission*	*Allowed Permissions*				*Sales Manager*			
		View	*Add*	*Edit*	*Delete*	*View*	*Add*	*Edit*	*Delete*
Account Management	Access to Account Management Module	Yes				Yes			
	Account Records	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Category	*Permission*	*Allowed Permissions*				*Salesperson*			
		View	*Add*	*Edit*	*Delete*	*View*	*Add*	*Edit*	*Delete*
Account Management	Access to Account Management Module	Yes				Yes			
	Account Records	Yes	Yes	Yes	Yes	Yes	**No**	Yes	**No**

```
CREATE TABLE role_permission (
  id SERIAL PRIMARY KEY,
  role_id INT NOT NULL REFERENCES role(role_id),
  permission_id INT NOT NULL REFERENCES permission(permission_id),
  role_view BOOLEAN NOT NULL DEFAULT false,
  role_add BOOLEAN NOT NULL DEFAULT false,
  role_edit BOOLEAN NOT NULL DEFAULT false,
  role_delete BOOLEAN NOT NULL DEFAULT false
);
```

III. Contact Modules (General Contacts, Employees, Account Contacts)

A. [contact]

Each user of the system must have a related contact record.

The contact table has a backwards pointer to **user_id** to simplify referencing a user. Each contact can have only one (1) login.

Each user can belong to only one (1) organization by populating the contact's **org_id**.

An **org_id** of 0 means the user belongs to the company that owns the database (an employee). If **org_id** is 0, then **department_id** further defines the contact, otherwise **department_id** is hidden at the application level.

If the user is indicated as the **primary_contact** then the corresponding organization record will be updated by the application.

If the user is not associated with an account, then **org_id** can be left NULL and a free-form text field called **company** can be populated.

The **org_name** field is maintained by the application, this is either the name of the related organization **[organization].name** OR a copy of the **company** field. This allows for more efficient access of the organization name for searches and sorting.

The **access_type** field indicates how this contact can be seen by users of the system. Access type can be set to:

1. Personal – this means that only the user that created the contact can see/manipulate this contact
2. Controlled-Hierarchy – this means that the user that created the contact, or anyone above the user in the hierarchy can see/manipulate this contact
3. Public – this means that anyone that has access to a module that shows contacts can see/manipulate this contact

All users have their contact information cached in memory by the application. This cache is reloaded by the application when (1) a user is added, updated, or deleted in the database, or (2) an administrator forces the cache to reload.

Contacts can be linked to Organizations, “Folders”, Calls, Communication Messages and Opportunities.

```
CREATE TABLE contact (
  contact_id serial PRIMARY KEY,
  user_id INT references access(user_id),
  org_id int REFERENCES organization(org_id),
  company VARCHAR(255),
  title VARCHAR(80),
  department INT references lookup_department(code),
  super INT REFERENCES contact,
  namesalutation varchar(80),
  namelast VARCHAR(80) NOT NULL,
  namefirst VARCHAR(80) NOT NULL,
  namemiddle VARCHAR(80),
  namesuffix VARCHAR(80),
  assistant INT REFERENCES contact,
  birthdate DATE,
  notes TEXT,
  site INT,
  imname VARCHAR(30),
  imservice INT,
  locale INT,
  employee_id varchar(80) UNIQUE,
  employmenttype INT,
  startofday VARCHAR(10),
  endofday VARCHAR(10),
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
  enabled BOOLEAN DEFAULT true,
  owner INT REFERENCES access(user_id),
  custom1 int default -1,
  url VARCHAR(100),
  primary_contact BOOLEAN DEFAULT false,
  employee boolean NOT NULL DEFAULT false,
  org_name VARCHAR(255),
  access_type INT REFERENCES lookup_access_types(code)
);
```

B. [contact_address] [contact_phone] [contact_emailaddress]

Each contact can have 0, 1, or unlimited addresses, phone numbers, and email addresses.

Each of these is categorized by **address_type, phone_type**, or **email_type**;

Types are typically “Business”, “Home”, etc., and can be modified using the application. The default type for listings is the first “Business” label found.

C. [contact_type_levels]

A contact can have 0, 1, or more contact types (or categories) that are defined by a user at the system level.

IV. Account Management Module

A. [organization]

An organization can either be a company or an individual.

The organization table stores some information regarding a specific organization, like:

1. **name** – the name of the account, either a company or individual
2. **account_number** – free-form id for manually inputting an account number
3. **url**
4. **revenue**, **employees**, **sic_code**, **ticker_symbol**, **taxid**, **fiscal_month**

The record with **org_id** 0 is reserved for the company that has installed the database and can be updated from record **name** "My Company" to your own company's name.

Organizations can be linked to "Folders", Contacts, Opportunities, Revenue, Tickets, and Documents.

```
CREATE TABLE organization (
  org_id INTEGER DEFAULT nextval('organization_org_id_seq') NOT NULL
PRIMARY KEY,
  name VARCHAR(80) NOT NULL,
  account_number VARCHAR(50),
  account_group INT,
  url TEXT,
  revenue FLOAT,
  employees INT,
  notes TEXT,
  sic_code VARCHAR(40),
  ticker_symbol VARCHAR(10) DEFAULT NULL,
  taxid CHAR(80),
  lead VARCHAR(40),
  sales_rep int NOT NULL DEFAULT 0,
  miner_only BOOLEAN NOT NULL DEFAULT 'f',
  defaultlocale INT,
  fiscalmonth INT,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL references access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL references access(user_id),
  enabled BOOLEAN DEFAULT true,
  industry_temp_code SMALLINT,
  owner INT references access(user_id),
  duplicate_id int default -1,
  custom1 int default -1,
  custom2 int default -1,
  contract_end date default null,
  alertdate date default null,
  alert varchar(100) default null,
  custom_data TEXT,
  namesalutation varchar(80),
  namelast varchar(80),
  namefirst varchar(80),
  namemiddle varchar(80),
```

```
  namesuffix varchar(80)
);
```

B. [organization_address] [organization_phone] [organization_emailaddress]

Each organization can have 0, 1, or unlimited addresses, phone numbers, and email addresses.

Each of these is categorized by **address_type, phone_type**, or **email_type**;

Types are typically “Primary”, “Auxiliary”, etc., and can be modified using the application. The default type for listings is the first “ Primary ” label found.

C. [account_type_levels]

An organization can have 0, 1, or more organization types (or categories) that are defined by a user at the system level.

V.Pipeline Management Module

Opportunities are made up of top-level descriptions, found in [opportunity_header], and 1 or more components, found in [opportunity_component].

Opportunities can be linked to Organizations, Contacts, Calls and Documents.

A. [opportunity_header]

The opportunity header contains information that groups opportunity components together. It is typically a **description** in which components will be attached.

An opportunity must be linked to an account, **acctlink** = [organization].org_id, and/or a contact, **contactlink** = [contact].contact_id.

```
CREATE TABLE opportunity_header (
  opp_id SERIAL PRIMARY KEY,
  description VARCHAR(80),
  acctlink INT default -1,
  contactlink INT default -1,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id)
);
```

B. [opportunity_component]

Components are specific elements that comprise opportunities, as described by the [opportunity_header].

Opportunities have several base fields:

1. **description** – the name of the component and how it might fit in with the opportunity
2. **closedate** – the expected date in which this opportunity will be realized
3. **closeprob** – the probability of confidence in which this component will be realized
4. **terms** – when an opportunity is realized, this is the number of days, months, years, etc in which the opportunity be realized
5. **units** – when an opportunity is realized, this is the declaration of whether units are in days, months, years, etc
6. **lowvalue** – the lowest expected amount that this component could generate
7. **guessvalue** – the expected amount that this component could generate
8. **highvalue** – the highest expected amount that this component could generate
9. **stage** – a component goes through 1 or more stages, as defined by the system until it is closed and either WON or LOST
10. **stagedate** – the date in which the current stage begin
11. **commission** – the expected percent the component owner would receive in commission
12. **type** – a component comes from either New or Existing business with an account

13. **closed** – the date in which this component reached a closed stage

```
CREATE TABLE opportunity_component (
  id serial PRIMARY KEY,
  opp_id int references opportunity_header(opp_id),
  owner INT NOT NULL REFERENCES access(user_id),
  description VARCHAR(80),
  closedate date not null,
  closeprob float,
  terms float,
  units char(1),
  lowvalue float,
  guessvalue float,
  highvalue float,
  stage INT references lookup_stage(code),
  stagedate date NOT NULL DEFAULT CURRENT_TIMESTAMP,
  commission float,
  type char(1),
  alertdate date,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
  closed TIMESTAMP,
  alert varchar(100) default null,
  enabled BOOLEAN NOT NULL DEFAULT true,
  notes TEXT
);
```

C. [opportunity_component_levels]

Each component can belong to one (1) or more categories, however for reporting purposes it is best to have only one (1) category per component.

VI. Activities Feature

Activities can be attached to various tables throughout multiple modules. Activities are implemented in Contacts, Accounts, and Opportunities.

A. [call_log]

The call log tracks communications, incoming and outgoing, made with a [contact], and can be in relation to an [organization] or an [opportunity_header]. Activities can be events that are complete or incomplete.

Base fields include:

1. **call_type_id** – the type of message: Phone Call, Fax, or In-Person
2. **followup_date** – a reminder when to follow-up regarding this note
3. **alertdate** – a reminder of an action to perform regarding this note
4. **result_id** – each activity has some sort of result that generally describes what the next step is with the contac
5. **parent_id** – the activity in which this activity was generated from

```
CREATE TABLE call_log (
  call_id SERIAL PRIMARY KEY,
  org_id INT REFERENCES organization(org_id),
  contact_id INT REFERENCES contact(contact_id),
  opp_id INT REFERENCES opportunity_header(opp_id),
  call_type_id INT REFERENCES lookup_call_types(code),
  length INTEGER,
  subject VARCHAR(255),
  notes TEXT,
  followup_date DATE,
  alertdate DATE,
  followup_notes TEXT,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
  alert varchar(100) default null
);
```

VII. Tasks Module/Feature

Tasks is a standalone module, as well as a feature that can be attached to various tables throughout multiple modules.

A. [task]

The task table keeps track of all tasks in the system, regardless of module.

Task fields:

1. **description** – each task must have a description to be valid, this is something the user is being reminded of
2. **priority** – an arbitrary priority to help in sorting tasks
3. **duedate** – the date in which the task is due
4. **reminderid** (unused)
5. **complete** – a flag indicating if the task is complete
6. **estimatedloe** – the expected amount of time it will take to complete the task
7. **estimatedloetype** – the unit of measurement (hours, days, weeks, etc) it will take, combined with the numerical amount
8. **type** – identifies which type of object this task is related to (tickets, etc) based on the Constants.java file; from the type you have to lookup the corresponding type table ([tasklink_ticket]) to get the id of the object, which enforces referential integrity
9. **owner** – the user in which this task is assigned to
10. **completedate** – the date the task was completed
11. **category_id** – the category in which this task is related, currently this is implemented in projects, but not the task module

```
CREATE TABLE task (
  task_id SERIAL PRIMARY KEY,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  priority INTEGER NOT NULL REFERENCES lookup_task_priority,
  description VARCHAR(80),
  duedate DATE,
  reminderid INT,
  notes TEXT,
  sharing INT NOT NULL,
  complete BOOLEAN DEFAULT false NOT NULL,
  enabled BOOLEAN DEFAULT false NOT NULL,
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT REFERENCES access(user_id),
  estimatedloe FLOAT,
  estimatedloetype INTEGER REFERENCES lookup_task_loe,
  type INTEGER DEFAULT 1,
  owner INTEGER REFERENCES access(user_id),
  completedate TIMESTAMP(3),
  category_id INTEGER REFERENCES lookup_task_category
);
```

B. [tasklink_contact] [tasklink_ticket] [tasklink_project]

Tasks can be linked to various objects in CFS. These tables provide the related Ids for which the link based on.

```
CREATE TABLE tasklink_contact (
  task_id INT NOT NULL REFERENCES task,
  contact_id INT NOT NULL REFERENCES contact(contact_id)
);

CREATE TABLE tasklink_ticket (
  task_id INT NOT NULL REFERENCES task,
  ticket_id INT NOT NULL REFERENCES ticket(ticketid)
);

CREATE TABLE tasklink_project (
  task_id INT NOT NULL REFERENCES task,
  project_id INT NOT NULL REFERENCES projects(project_id)
);
```

C. [taskcategory_project]

This table links a task's category to a specific project. This implementation allows each project to have a separate list of categories.

```
CREATE TABLE taskcategory_project (
  category_id INTEGER NOT NULL REFERENCES lookup_task_category,
  project_id INTEGER NOT NULL REFERENCES projects(project_id)
);
```

VIII. Folders Feature

Folders allow custom fields to be attached to other tables without dynamically modifying the database schema. They can be incorporated into a module in various ways, but typically appear as stand-alone forms within the module.

Folders are implemented in Contacts and Accounts.

Custom fields have to be defined first, then a form can be dynamically displayed against the definitions, then the populated data can be stored in related tables.

Module	*Category*	*Groups*	*Fields*
Account Management	Site Survey	Indoors	Building Location
			Building Supervisor
			# of offices
		Outdoors	Mount Location
			Closest Transmitter

A. [module_field_categorylink]

The application developer must indicate that a module or object within a module has "folders" capability by storing a record that maps back to the [permission_category] table. This is the first part in allowing an object to use folders.

Each module or object, can have any number of "folders." For example, folders could be added to a Contact, but also more narrowly on a Contact's Address.

The **category_id** here is a unique reference that the application uses.

```
CREATE TABLE module_field_categorylink (
  id INTEGER DEFAULT nextval('module_field_categorylin_id_seq') NOT
NULL PRIMARY KEY,
  module_id INTEGER NOT NULL REFERENCES
permission_category(category_id),
  category_id INT UNIQUE NOT NULL,
  level INTEGER DEFAULT 0,
  description TEXT,
  entered TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP
);
```

B. *[custom_field_category]*

When a user uses the application to define a folder for a specific [module_field_categorylink].category_id, the record are stored here.

Base fields:

1. **category_name** – the name a user sees for this category
2. **description** (unused) – a lengthy description a user sees for this category, maybe usage notes
3. **start_date** (unused) – intended for enabled a folder by date
4. **end_date** (unused) – intended for disabling a folder by date
5. **enabled –** If enabled, then this category with be visible to the user
6. **multiple_records** – specifies whether the user can add multiple records against the object that has custom fields, otherwise only one (1) record per object
7. **read_only** – specifies whether the user can add or modify records against the object that has custom fields; typically set to read_only when data is being imported or for outdated data

```
CREATE TABLE custom_field_category (
  module_id INTEGER NOT NULL REFERENCES
module_field_categorylink(category_id),
  category_id INTEGER DEFAULT
nextval('custom_field_ca_category_id_seq') NOT NULL PRIMARY KEY,
  category_name VARCHAR(255) NOT NULL,
  level INTEGER DEFAULT 0,
  description TEXT,
  start_date TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  end_date TIMESTAMP,
  default_item BOOLEAN DEFAULT false,
  entered TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  enabled BOOLEAN DEFAULT true,
  multiple_records BOOLEAN DEFAULT false,
  read_only BOOLEAN DEFAULT false
);
```

C. *[custom_field_group]*

For each [custom_field_category], before any field definitions are appended, one (1) or more groups need to be added.

A group is for storing like fields together.

Base fields:

1. **group_name** – The name the user sees above the fields
2. **level** – the position in which this group of fields is displayed under the corresponding category
3. **description** (unused) – a lengthy description to display to the user, maybe usage notes
4. **start_date** (unused)
5. **end_date** (unused)

```
CREATE TABLE custom_field_group (
  category_id INTEGER NOT NULL REFERENCES
custom_field_category(category_id),
  group_id INTEGER DEFAULT nextval('custom_field_group_group_id_seq')
NOT NULL PRIMARY KEY,
  group_name VARCHAR(255) NOT NULL,
  level INTEGER DEFAULT 0,
  description TEXT,
  start_date TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  end_date TIMESTAMP,
  entered TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  enabled BOOLEAN DEFAULT true
);
```

D. [custom_field_info]

The fields that are defined in this category, under the corresponding group.

Base fields:

1. **field_name** – The title of the field that is displayed to the user
2. **level** – The order in which the field is shown, relative to other fields in the same group
3. **field_type** – Maps back to the application to identify the field type, for example: Text Field, Multi-Line Text Field, Lookup List, Check Box, Date Field, Number Field, Percent Field, Email Field, URL Field, etc.
4. **validation_type** (unused)
5. **required** – specifies if the field must contain data before saving
6. **parameters** – special formatting requirements to be used in displaying the field
7. **start_date** (unused)
8. **end_date** (unused)
9. **enabled** – specifies whether the field is shown to the user
10. **additional_text** – Text that shows to the user to help them enter data into this field

```
CREATE TABLE custom_field_info (
  group_id INTEGER NOT NULL REFERENCES custom_field_group(group_id),
  field_id SERIAL PRIMARY KEY,
  field_name VARCHAR(255) NOT NULL,
  level INTEGER DEFAULT 0,
  field_type INTEGER NOT NULL,
  validation_type INTEGER DEFAULT 0,
  required BOOLEAN DEFAULT false,
  parameters TEXT,
  start_date TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  end_date TIMESTAMP(3) DEFAULT NULL,
  entered TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  enabled BOOLEAN DEFAULT true,
  additional_text VARCHAR(255)
);
```

E. [custom_field_lookup]

This table stores the lookup values for a custom field that is defined as type Lookup

List.

```
CREATE TABLE custom_field_lookup (
  field_id INTEGER NOT NULL REFERENCES custom_field_info(field_id),
  code SERIAL PRIMARY KEY,
  description VARCHAR(255) NOT NULL,
  default_item BOOLEAN DEFAULT false,
  level INTEGER DEFAULT 0,
  start_date TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  end_date TIMESTAMP,
  entered TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  enabled BOOLEAN DEFAULT true
);
```

F. [custom_field_record]

The following table is used for storing custom field data against the custom folder definitions. Inserted records act as a bridge between the object, its definitions, and the entered data.

For example, a custom folder might have been defined so that a Contact can have the following additional fields: favorite color and favorite number.

When a user inserts or updates a contact, those fields would be shown, and then a corresponding record would be inserted or updated. If the [custom_field_category] only allows one (1) record, then there will only be one (1) related record here.

Base fields:

1. **link_module_id** – links back to the module
2. **link_item_id** – links back to the module record in which field data is being attached
3. **category_id** – links back to the category that these fields belong to

```
CREATE TABLE custom_field_record (
  link_module_id INTEGER NOT NULL,
  link_item_id INTEGER NOT NULL,
  category_id INTEGER NOT NULL REFERENCES
custom_field_category(category_id),
  record_id INTEGER DEFAULT nextval('custom_field_reco_record_id_seq')
NOT NULL PRIMARY KEY,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
  enabled BOOLEAN DEFAULT true
);
```

G. [custom_field_data]

Each field that comprises a [custom_field_record] is stored in this table.

Base fields:

1. **field_id** – links back to the definition for this field
2. **selected_item_id** – specifies an item [custom_field_lookup]

3. **entered_value** – the exact value the user entered in this field
4. **entered_number** – an integer version for sorting, reports, etc.
5. **entered_float** – a float version for sorting, reports, etc.

```
CREATE TABLE custom_field_data (
  record_id INTEGER NOT NULL REFERENCES custom_field_record(record_id),
  field_id INTEGER NOT NULL REFERENCES custom_field_info(field_id),
  selected_item_id INTEGER DEFAULT 0,
  entered_value TEXT,
  entered_number INTEGER,
  entered_float FLOAT
);
```

IX. Help Desk Module/Tickets Feature

Help Desk is a stand-alone module that uses the concept of a ticket, however tickets can be included in other modules as well. For example, Project Management could have tickets associated with a project.

When a ticket is entered, there are several optional relationships, which include Contracts, Assets, Labor Categories, Documents, and Tasks.

A. [ticket]

Base fields:

1. **org_id** – the organization that requested this ticket
2. **contact_id** – the contact that requested this ticket
3. **problem** – the issue that was raised in response to creating this ticket
4. **closed** – the date/time when this ticket was closed, whether solved or not
5. **pri_code** – the assigned priority
6. **level_code** (unused)
7. **department_code** – the department to which this ticket is assigned
8. **source_code** – the method in which this ticket was submitted: email, phone call, etc.
9. **cat_code** – the top level category that this ticket maps to
10. **subcat_code1** – the category narrowed another level
11. **subcat_code2** – the category narrowed another level
12. **subcat_code3** – the category narrowed another level
13. **assigned_to** – the user that is assigned this ticket
14. **comment** (unused)
15. **solution** – the solution that was used to solve the issue
16. **scode** – The severity in which the issue is disruptive of service
17. **critical** (unused)

```
CREATE TABLE ticket (
  ticketid SERIAL PRIMARY KEY,
  org_id INT REFERENCES organization,
  contact_id INT REFERENCES contact,
  problem TEXT NOT NULL,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
  closed TIMESTAMP,
  pri_code INT REFERENCES ticket_priority(code),
  level_code INT REFERENCES ticket_level(code),
  department_code INT REFERENCES lookup_department,
  source_code INT REFERENCES lookup_ticketsource(code),
  cat_code INT,
  subcat_code1 INT,
  subcat_code2 INT,
  subcat_code3 INT,
  assigned_to INT REFERENCES access,
  comment TEXT,
  solution TEXT,
```

```
  scode INT REFERENCES ticket_severity(code),
  critical TIMESTAMP,
  notified TIMESTAMP,
  custom_data TEXT
);
```

B. [ticketlog]

When a ticket is updated, the data is analyzed and all changes are appended to a log, including comments.

For example, the ticket workflow might include the following steps, at each step a log record is created:

1. The ticket issue is entered into the database
2. The ticket is assigned to a user
3. The user reviews the ticket and adds a comment
4. The user closes the ticket

These fields resemble a ticket and map back to the user that made the changes.

```
CREATE TABLE ticketlog (
  id serial PRIMARY KEY
  ,ticketid INT REFERENCES ticket(ticketid)
  ,assigned_to INT REFERENCES access(user_id)
  ,comment TEXT
  ,closed BOOLEAN
  ,pri_code INT REFERENCES ticket_priority(code)
  ,level_code INT
  ,department_code INT REFERENCES lookup_department(code)
  ,cat_code INT
  ,scode INT REFERENCES ticket_severity(code)
  ,entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP
  ,enteredby INT NOT NULL REFERENCES access(user_id)
  ,modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP
  ,modifiedby INT NOT NULL REFERENCES access(user_id)
);
```

X.Communications Manager

Communications are essentially a group of contacts in which a message is to be sent to. There's more to it than that because you have to build a group, build the message, define when the communication should occur, and how it should occur.

A. [campaign]

Each campaign record defines some basic information about a communication. Some of the information occurs while the user is constructing a communication, and some of the information is stored for historical purposes after the communication is sent.

Fields:

1. **name** – the title of the communication that shows in lists
2. **description** – an internal description about the communication
3. **list_id**
4. **message_id** – the linked message that WILL BE sent to the campaign groups
5. **reply_addr** – the reply address that WAS sent to the campaign groups
6. **message** – the message that WAS sent to the campaign recipients
7. **status_id** – an application status code describing the state of the campaign (under construction, in the queue, sent)
8. **status** – a text description of the status_id
9. **active/active_date** – indicates if the campaign is active (in the queue or sent)
10. **send_method_id** – indicates if the campaign should be emailed, faxed, or a combination of various methods
11. **inactive_date**
12. **approval_date/approvedby**
13. **type**

```
CREATE TABLE campaign (
  campaign_id serial PRIMARY KEY,
  name VARCHAR(80) NOT NULL,
  description VARCHAR(255),
  list_id int,
  message_id int DEFAULT -1,
  reply_addr VARCHAR(255) DEFAULT NULL,
  subject VARCHAR(255) DEFAULT NULL,
  message TEXT DEFAULT NULL,
  status_id INT DEFAULT 0,
  status VARCHAR(255),
  active BOOLEAN DEFAULT false,
  active_date DATE DEFAULT NULL,
  send_method_id INT DEFAULT -1 NOT NULL,
  inactive_date DATE DEFAULT NULL,
  approval_date TIMESTAMP(3) DEFAULT NULL,
  approvedby INT REFERENCES access(user_id),
  enabled BOOLEAN NOT NULL DEFAULT true,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
```

```
  type INT DEFAULT 1
);
```

B. [campaign_run]

When a campaign has been activated, a background process begins sending emails and faxes. A record will be created in [campaign_run] with information regarding the process.

```
CREATE TABLE campaign_run (
  id serial PRIMARY KEY,
  campaign_id INTEGER NOT NULL REFERENCES campaign(campaign_id),
  status INTEGER NOT NULL DEFAULT 0,
  run_date TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  total_contacts INTEGER DEFAULT 0,
  total_sent INTEGER DEFAULT 0,
  total_replied INTEGER DEFAULT 0,
  total_bounced INTEGER DEFAULT 0
);
```

C. [scheduled_recipient]

When a campaign has been activated, a query will generate a list of contacts that will receive a message and store them in [scheduled_recipient].

A separate process will go through each scheduled recipient and send them the campaign message. The status of the message is stored here as to whether the message was successfully sent or not.

```
CREATE TABLE scheduled_recipient (
  id SERIAL PRIMARY KEY,
  campaign_id INT NOT NULL REFERENCES campaign(campaign_id),
  contact_id INT NOT NULL REFERENCES contact(contact_id),
  run_id INT DEFAULT -1,
  status_id INT DEFAULT 0,
  status VARCHAR(255),
  status_date TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  scheduled_date TIMESTAMP(3) DEFAULT CURRENT_TIMESTAMP,
  sent_date TIMESTAMP(3) DEFAULT NULL,
  reply_date TIMESTAMP(3) DEFAULT NULL,
  bounce_date TIMESTAMP(3) DEFAULT NULL
);
```

D. [excluded_recipient]

When a campaign is processing, contacts that appear in [excluded_recipient] will be skipped.

```
CREATE TABLE excluded_recipient (
  id SERIAL PRIMARY KEY,
  campaign_id INT NOT NULL REFERENCES campaign(campaign_id),
  contact_id INT NOT NULL REFERENCES contact(contact_id)
);
```

E. [campaign_list_groups]

A communication is sent to groups of contacts. A user will add groups to the campaign by choosing 1 or more through the interface. Criteria for groups are defined in other tables.

```
CREATE TABLE campaign_list_groups (
  campaign_id INT NOT NULL REFERENCES campaign(campaign_id),
  group_id INT NOT NULL REFERENCES saved_criterialist(id)
);
```

F. [saved_criterialist]

A campaign group is made up of criteria. For example, maybe a group of contacts should be "all account contacts in zip code 23456 AND all account contacts of type active." This table stores the **name** for the contact group and the user who owns the criteria for this list. The **contact_source** field is no longer used at this level, it is used in the [saved_criteriaelement] table.

```
CREATE TABLE saved_criterialist (
  id SERIAL PRIMARY KEY,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INTEGER NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INTEGER NOT NULL REFERENCES access(user_id),
  owner INTEGER NOT NULL REFERENCES access(user_id),
  name VARCHAR(80) NOT NULL,
  contact_source INTEGER DEFAULT -1,
  enabled BOOLEAN NOT NULL DEFAULT true
);
```

G. [saved_criteriaelement]

Each criteria list is made up of elements that can be used to generate a contact list later. The elements will be used to dynamically generate a query, constructed in the ContactList.java class.

Fields:

1. **field** – the query will be filtered by this field specified
2. **operator** – this field specifies how the filter should be applied and will be used in the query; "contains", "is", "is not", etc.
3. **operatorid** – this field relates to a full description of the operator
4. **value** – the data entered by the user that the filter will use in the query
5. **source** – this field indicates if the contacts are to be generated from "My General Contacts", "All General Contacts", "Account Contacts", or "Employees."

```
CREATE TABLE saved_criteriaelement (
  id INTEGER NOT NULL REFERENCES saved_criterialist(id),
  field INTEGER NOT NULL references search_fields(id),
  operator VARCHAR(50) NOT NULL,
  operatorid INTEGER NOT NULL references field_types(id),
  value VARCHAR(80) NOT NULL,
```

```
  source INT NOT NULL DEFAULT -1
);
```

H.[search_fields]

This table contains a list of contact fields in which criteria can be specified when generating a list. These items are added by a developer and correspond to the ContactList.java class.

When a user uses "Build Groups" in the web application, a list of fields that can be filtered on is shown from this table.

```
CREATE TABLE search_fields (
  id SERIAL PRIMARY KEY,
  field varchar(80),
  description VARCHAR(255),
  searchable BOOLEAN NOT NULL DEFAULT true,
  field_typeid int NOT NULL DEFAULT -1,
  table_name varchar(80),
  object_class varchar(80),
  enabled BOOLEAN DEFAULT true
);
```

I. [field_types]

This table contains the operators that can be used on each contact field from [search_fields] based on its **field_typeid**.

So if the search field "company" is selected, then the operators include:

1. Company field "is"
2. Company field "is not"
3. Company field "is empty"
4. Company field "is not empty"
5. Company field "contains"
6. Company field "does not contain"

J. [message]

Users can create messages that will be associated with [campaign] records. This table holds message records that resemble e-mail messages that will either be e-mailed, faxed, or printed.

Fields:

1. **name** – an internal name that will be displayed to users
2. **description** – an internal description of the message to be displayed to users
3. **template_id** (unused)
4. **subject** – the message subject that a recipient will see
5. **body –** the body of the message that a recipient will see
6. **reply_addr** – the return e-mail address to be added to the message
7. **url** (unused)

8. **img** (unused)
9. **access_type** – this field indicates whether the message is Public and can be seen by any user of the system, or Controlled in which only the user's hierarchy can see the message

```
CREATE TABLE message (
  id SERIAL PRIMARY KEY,
  name VARCHAR(80) NOT NULL,
  description VARCHAR(255),
  template_id INT,
  subject VARCHAR(255) DEFAULT NULL,
  body TEXT,
  reply_addr VARCHAR(100),
  url VARCHAR(255),
  img VARCHAR(80),
  enabled BOOLEAN NOT NULL DEFAULT true,
  entered TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  enteredby INT NOT NULL REFERENCES access(user_id),
  modified TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
  modifiedby INT NOT NULL REFERENCES access(user_id),
  access_type INT REFERENCES lookup_access_types(code)
);
```

K. [message_template] (unused)

L. [survey_*]

Tables used in the creation of surveys that will be attached to a [campaign]. Users can edit these surveys and will not affect an activated campaign.

M.[active_survey_*]

Tables used for historical purposes when a campaign has been sent. These surveys are not editable but are displayed once a campaign has been activated.

XI. Project Management

Projects are at the core of the Project Management module. Once a project has been created, there are other objects that can be related, including News Articles, Documents, Discussions, Lists, Tickets, Team Members, Outlines, and Permissions.

A. [projects]

Fields:

1. **title, shortdescription** – describe the project_id
2. **portal, allow_guests** (not used)
3. ***_enabled** – determines if the capability is enabled for this projec
4. ***_label** – renames the tabs in the applicatio

B. [project_requirements]

Describe a project plan.

C. [project_assignments_folder]

A component of a project plan, a folder, in which assignments can be added.

D. [project_assignments]

Project plan, an assignment, representing work that needs to be done.

E. [project_assignments_status]

Not implemented.

F. [project_requirements_map]

Tracks the order in which assignments and assignments folders are added to the requirement outline.

G. [project_issues_categories]

Represents the discussion forum.

H. [project_issues]

Represents topics that have been added to the forum.

I. [project_issue_replies]

Represents messages that have been added to a topic.

J. [project_team]

The members of the team, always users, and their role within the project. The role will decided the permissions of the member.

K. [project_news]

Represents project news articles.

L. [project_permissions]

Each project can determine how a role acesses the project.

XII. Reports Module

The Reports Module stores data required for generating user requested reports.
The report table defines the reports for the application, while the remaining tables hold specific details for a single user report.

A. [report]

All the reports defined in the application are represented by a row in this table.

Fields:

1. report_id – an internal id for this report
2. category_id – defines which category this report will be listed beneath
3. permission_id – the lowest role that can view this report
4. filename – the name of the xml configuration file for this report
5. type - unused
6. title – a title that will be displayed to the user
7. description – a description that will be displayed to the user
8. custom - ?

B. [report_criteria]

??
Fields:

1. criteria_id
2. report_id
3. owner
4. subject

C. [report_queue]

When a user requests a report to be generated a row representing that request is recorded in this table. When the report is completed it's status is updated here.
Fields:

1. queue_id – an internal id for this queue
2. report_id – the report as listed in the [report] table
3. processed – the timestamp of the time this report was processed
4. status – the status of the report
5. filename – the filename the report is saved as on the server
6. filesize – the size of the completed report
7. output_type – the file type of the report as defined in [lookup_report_type]
8. email – boolean that specifies whether the report will be sent to the user when it is complete

D. [report_queue_criteria]

These criteria are associated with a given report already in the [report_queue] table. Any criteria present will be loaded as parameters dynamically in the report.

Fields:

1. criteria_id – an internal id for this criteria
2. queue_id – the report this criteria is assigned to
3. parameter – the name of the parameter, this should match the name in the xml configuration file for this report
4. value – the value of this parameter, this will be the value dynamically loaded in the report

XIII. Actions Module

The Actions Module...

A. [action_item]

Fields:

1. item_id -
2. action_id -
3. link_item_id -
4. completedate -
5. enabled -

B. [action_item_log]

Fields:

1. log_id -
2. item_id -
3. link_item_id -
4. type -

C. [action_item_work]

Fields:

1. item_work_id -
2. phase_work_id -
3. action_step_id -
4. status_id -
5. owner -
6. start_date -
7. end_date -
8. link_module_id -
9. link_item_id -
10. level -

D. [action_item_work_notes]

Fields:

1. note_id -
2. item_work_id -
3. description -

E. [action_item_work_selection]

Fields:

1. selection_id -
2. item_work_id -
3. selection -

F. [action_list]

Fields:

1. action_id -
2. description -
3. owner -
4. completedate -
5. link_module_id -

G. [action_phase]

Fields:

1. phase_id -
2. parent_id -
3. plan_id -
4. phase_name -
5. description -
6. random -
7. global -

H. [action_phase_work]

Fields:

1. phase_work_id -
2. plan_work_id -
3. action_phase_id -
4. status_id -
5. start_date -
6. end_date -
7. level -

I. [action_plan]

Fields:

1. plan_id -
2. plan_name -

3. description -
4. enabled -
5. approved -
6. archive_date -
7. cat_code -
8. subcat_code1 -
9. subcat_code2 -
10. subcat_code3 -
11. link_object_id -
12. site_id -

J. [action_plan_category]

Fields:

1. id -
2. cat_level -
3. parent_cat_code -
4. description -
5. full_description -
6. default_item -
7. level -
8. site_id -

K. [action_plan_category_draft]

Fields:

1. id -
2. link_id -
3. cat_level -
4. parent_cat_code -
5. description -
6. full_description -
7. default_item -
8. level -
9. site_id -

L. [action_plan_constants]

Fields:

1. map_id -
2. constant_id -
3. description -

M. [action_plan_work]

Fields:

1. plan_work_id -
2. action_plan_id -
3. manager -
4. assignedto -
5. link_module_id -
6. link_item_id -
7. current_phase -

N. [action_plan_work_notes]

Fields:

1. note_id -
2. plan_work_id -
3. description -

O. [action_step]

Fields:

1. step_id -
2. parent_id -
3. phase_id -
4. description -
5. duration_type_id
6. estimated_duration -
7. category_id -
8. field_id -
9. permission_type -
10. role_id -
11. department_id -
12. allow_skip_to_here -
13. label -
14. action_required -
15. group_id -
16. target_relationship -
17. action_id -
18. allow_update -
19. campaign_id -
20. allow_duplicate_recipient -
21. display_in_plan_list -
22. plan_list_label -
23. quick_complete -

XIV. Assets Module

The Assets Module...

A. [asset]

Fields:

1. asset_id -
2. account_id -
3. contract_id -
4. date_listed -
5. asset_tag -
6. status -
7. location -
8. level1 -
9. level2 -
10. level3 -
11. serial_number -
12. model_version -
13. description -
14. expiration_date -
15. inclusions -
16. exclusions -
17. purchase_date -
18. po_number -
19. purchased_from -
20. contact_id -
21. notes -
22. response_time -
23. telephone_service_model -
24. onsite_service_model -
25. email_service_model
26. purchase_cost -
27. date_listed_timezone -
28. expiration_date_timezone -
29. purchase_date_timezone -
30. parent_id -
31. vendor_code -
32. manufacturer_code -

B. [asset_category]

Fields:

1. id -

2. cat_level -
3. parent_cat_code -
4. description -
5. full_descripiton -
6. default_item -
7. level -
8. site_id -

C. [asset_category_draft]

D. [asset_materials_map]

Fields:

1. map_id -
2. asset_id -
3. code -
4. quantity -

XV. Documents Module

The Documents Module...

The project_files table stores information on the files themselves.

A. [document_accounts]

Fields:

1. id -
2. document_store_id -
3. org_id -

B. [document_store]

Fields:

1. document_store_id -
2. template_id -
3. title -
4. shortdescription -
5. requestedby -
6. requesteddept -
7. requestdate -
8. requestdate_timezone -
9. approvaldate -
10. approvalby -
11. closedate -
12. public_store -

C. [document_store_department_member]

Fields:

1. document_store_id -
2. item_id -
3. userlevel -
4. status -
5. last_accessed -
6. site_id -
7. role_type -

D. [document_store_permissions]

Fields:

1. id -
2. document_store_id -
3. permission_id -
4. userlevel -

E. [document_store_role_member]

Fields:

1. document_store_id
2. item_id -
3. userlevel -
4. status -
5. last_accessed -
6. site_id -
7. role_type -

F. [document_store_user_member]

Fields:

1. document_store_id -
2. item_id -
3. userlevel -
4. status -
5. last_accessed -
6. site_id -
7. role_type -

G. [project_files]

Fields:

1. item_id -
2. link_module_idl -
3. link_item_id -
4. folder_id -
5. client_filename -
6. filename -
7. subject -
8. size -
9. version -
10. enabled -
11. downloads -
12. default_file -
13. allow_portal_access -

XVI. Product Module

The Product Module...

A. [product_catalog]

Fields:

1. product_id -
2. parent_id -
3. product_name -
4. abbreviation -
5. short_description -
6. long_description -
7. type_id -
8. special_notes -
9. sku -
10. in_stock -
11. format_id -
12. shipping_id -
13. estimated_ship_time -
14. thumbnail_image_id -
15. small_image_id -
16. large_image_id -
17. list_order -
18. start_date -
19. expiration_date -
20. manufacturer_id -
21. comments -
22. import_id -
23. status_id -

B. [product_catalog_category_map]

Fields:

1. price_id -
2. product_id -
3. tax_id -
4. msrp_currency -
5. msrp_amount -
6. price_currency -
7. price_amount -
8. recurring_currency -
9. recurring_amount -
10. recurring_type -

11. start_date -
12. expiration_date -
13. cost_currency -
14. cost_amount -

C. [product_category]

Fields:

1. category_id -
2. parent_id -
3. category_name -
4. abbreviation -
5. short_description -
6. long_description -
7. type_id -
8. thumbnail_image_id -
9. small_image_id -
10. large_image_id -
11. list_order -
12. start_date -
13. expiration_date -
14. import_id -
15. status_id -

D. [product_category_map]

Fields:

1. id -
2. category1_id -
3. category2_id -

E. [product_keyword_map]

Fields:

1. product_id -
2. keyword_id -

F. [product_option]

Fields:

1. option_id -
2. configurator_id -
3. parent_id -

4. short_description -
5. long_description -
6. allow_customer_configure -
7. allow_user_configure -
8. required -
9. start_date -
10. end_date -
11. option_name -
12. has_range -
13. has_multiplier -

---

*G. product_option_**

....

XVII. Order Module

The Order Module...

A. [order_product]

Fields:

1. item_id -
2. order_id -
3. product_id -
4. quantity -
5. msrp_currency -
6. msrp_amount -
7. price_currency -
8. price_amount -
9. recurring_currency -
10. recurring_amount -
11. recurring_type -
12. extended_price -
13. total_price -
14. status_id -
15. status_date -

B. [order_product_option_*]

...

C. [order_product_status]

Fields:

1. order_product_status_id -
2. order_id -
3. item_id -
4. status_id -

D. [order_address]

Fields:

1. address_id -
2. order_id -
3. address_type -
4. addrline1 -

5. addrline2 -
6. addrline3 -
7. addrline4 -
8. city -
9. state -
10. country -
11. postalcode -

E. [order_entry]

Fields:

1. order_id -
2. parent_id -
3. org_id -
4. quote_id -
5. sales_id -
6. orderedby -
7. billing_contact_id -
8. source_id -
9. grand_total -
10. status_id -
11. status_date -
12. contract_date -
13. expiration_date -
14. order_terms_id -
15. order_type_id -
16. description -
17. notes -
18. submitted -
19. approx_ship_date -
20. approx_delivery_date -

F. [order_payment]

Fields:

1. payment_id -
2. order_id -
3. payment_method_id -
4. payment_amount -
5. authorization_ref_number -
6. authorization_code -
7. authorization_date -
8. order_item_id -
9. history_id -
10. date_to_process -

11. creditcard_id -
12. bank_id -
13. status_id -

G. [order_payment_status]

Fields:

1. payment_status_id -
2. payment_id -
3. status_id -

XVIII. Quotes Module

The Quotes Module...

A. [quote_product]

Fields:

1. item_id -
2. quote_id -
3. product_id -
4. quantity -
5. price_currency -
6. price_amount -
7. recurring_currency -
8. recurring_amount -
9. recurring_type -
10. extended_price -
11. total_price -
12. estimated_delivery_date -
13. status_id -
14. status_date -
15. estimated_delivery -
16. comment -

B. [quote_condition]

Fields:

1. map_id -
2. quote_id -
3. condition_id -

C. [quote_entry]

Fields:

1. quote_id -
2. parent_id -
3. org_id -
4. contact_id -
5. source_id -
6. grand_total -
7. status_id -
8. status_date -
9. expiration_date -

10. quote_terms_id -
11. quote_type_id -
12. issued -
13. short_description -
14. notes -
15. ticketid -
16. product_id -
17. customer_product_id -
18. opp_id -
19. version -
20. group_id -
21. delivery_id -
22. email_address -
23. phone_number -
24. address -
25. fax_number -
26. submit_action -
27. closed -
28. show_total -
29. show_subtotal -
30. logo_file_id -
31. trashed_date -

D. [quote_group]

Fields:

1. group_id -
2. unused -

E. [quote_notes]

Fields:

1. notes_id -
2. quote_id -
3. notes -

F. [quote_remark]

Fields:

1. map_id -
2. quote_id -
3. remark_id -

G. [quote_product_option_*]

...

XIX. Web Module

The Web Module...

A. [web_page]

Fields:

1. page_id -
2. page_name -
3. page_position -
4. active_page_version_id -
5. construction_page_version_id -
6. page_group_id -
7. tab_banner_id -
8. notes -
9. page_alias -
10. link_module_id -
11. link_container_id -
12. keywords -
13. change_freq -
14. page_priority -

B. [web_layout]

Fields:

1. layout_id -
2. layout_constraint -
3. layout_name -
4. jsp -
5. thumbnail -
6. custom -

C. [web_page_access_log]

Fields:

1. page_id -
2. site_log_id -

D. [web_page_group]

Fields:

1. page_group_id -

2. group_name -
3. internal_description -
4. group_position -
5. tab_id -

E. [web_page_role_map]

Fields:

1. page_role_map_id -
2. web_page_id -
3. role_id -

F. [web_page_row]

Fields:

1. page_row_id -
2. row_position -
3. page_version_id -
4. row_column_id -

G. [web_page_version]

Fields:

1. page_version_id -
2. version_number -
3. internal_description -
4. notes -
5. parent_page_version_id -
6. page_id -

H. [web_product_access_log]

Fields:

1. product_id -
2. site_log_id -

I. [web_product_email_log]

Fields:

1. product_id -
2. emails_to
3. from_name -

4. comments -
5. site_log_id -

J. [web_row_column]

Fields:

1. row_column_id -
2. column_position -
3. width -
4. page_row_id -
5. icelet_id -

K. [web_site]

Fields:

1. site_id -
2. site_name -
3. internal_description -
4. hit_count -
5. notes -
6. layout_id -
7. style_id -
8. logo_image_id -
9. scripts -
10. url -

L. [web_site_access_log]

Fields:

1. site_log_id -
2. site_id -
3. user_id -
4. ip -
5. browser -
6. referrer -

M. [web_site_log]

Fields:

1. site_log_id -
2. site_id -
3. user_id -
4. username -

5. ip -
6. browser -

N. [web_style]

Fields:

1. style_id -
2. style_constant -
3. style_name -
4. css -
5. thumbnail -
6. custom -
7. layout_id -

O. [web_tab]

Fields:

1. tab_id -
2. display_text -
3. internal_description -
4. site_id -
5. tab_position -
6. keywords -

P. [web_tab_banner]

Fields:

1. tab_banner_id -
2. tab_id -
3. image_id -