Common Denominator Developer And System Integration Guide

Below are the specifications for querying and modifying the Common Denominator database. System integrators should only perform inserts, updates or deletes through stored procedures. Queries should only be performed through views or the functions listed in the 'Views, Data Queries and Reports' section.

Common Denominator stores personally identifiable information (PII). Medical and financial information is not stored. Ensure that all integrators follow an equally restrictive privacy policy, and that secure coding techniques are practiced. We should consider all of our member's demographics information as confidential.

Database Setup Guide

There are 5 included SQL files. Only 2 are necessary to create the full database. The remainder are for testing purposes.

CommonDenominator-01-DB.sql
Creates the commondenom database.

CommonDenominator-02-Schema.sql
Creates the tables, constraints, views, functions, triggers and procedures. Note that running this script will drop all of the above mentioned items, effectively removing all data from the system in the process. The 'Down' portion of this script is intended for sandbox/test environments only, with the exception of first time deployment to production.

CommonDenominator-03-DataLoad.sql
Loads sample data into all tables. To ensure referential integrity, this script will manually set the primary key values. This script is intended for sandbox/test environments only.

CommonDenominator-04-DataVerification.sql
This test script will output a small amount of data from all tables and query some functions/views. All of these operations are being done as read only, so existing table data will not be modified. Note that some of the tests expect certain values from the DataLoad script in order to pass. This script is intended for sandbox/test environments only, but could be safely run in production with potential false positives.

CommonDenominator-05-DataTesting.sql
This test script will perform insert, update or delete operations to test the programatic functions. This will modify the existing table data, or insert new records. Note that some of the tests expect certain values from the DataLoad script in order to pass. This script is intended for sandbox/test environments only.

Views, Data Queries and Reports

The following represent how a developer can query the system for data or reports. These will be a combination of views (denoted as v_*) or functions (denoted as f_*). In this document, we are grouping these items into logical categories.

Member Information

v_member_list_all_info
The member list views does not include the password column. Even though we are storing it in an encrypted format, there's no good reason to expose the encrypted phrase.

v_member_list_contact_info
This view only has contact information for promotional campaigns. It is intended to convey very little personal demographics information.

v_members_demographics_only
The demographics only view is intended to remove personally identifiable information. This allows us to perform or contract out data analytics without revealing the members. It should be noted that for many people, you could still potentially tie this data back to a specific individual with a high level of confidence, so we must still be careful in who has access to this data and who can see it.

Class Information

v_classes_all
All classes, past and future

v_classes_upcoming
Classes scheduled

v_classes_prior
Classes previously held

v_classes_happening_now
Classes currently in session

f_classes_by_leader_id
@leaderid Integer, the members.member_id key of the class leader
See all the classes taught by a particular leader.

f_classes_by_leader_email
@leaderemail Varchar (255), the members.member_email of the class leader
See all the classes taught by a particular leader.

Class Enrollment

v_class_enrollment
Class enrollment report for all classes

f_class_attendance_by_class_id
@classid Integer, the classes.class_id for a particular class
All the members enrolled in a specific class.

f_class_attendance_by_leader_id
@leaderid Integer, the members.member_id key of the class leader
All the members enrolled for all classes conducted by a particular leader

f_class_attendance_by_leader_email
@leaderemail Varchar (255), the members.member_email of the class leader

All the members enrolled for all classes conducted by a particular leader

f_class_attendance_by_member_id
@memberid Integer, the member.member_id of an enrolled member
All classes a member enrolled in.

f_class_attendance_by_member_email
@memberid Varchar (255), the member.member_email of an enrolled member
All classes a member enrolled in.

f_wordcloud_by_class_id
@classid Integer, the classes.class_id for a particular class
Produces a word cloud compatible listing of all the demographics for a particular class

f_gcd_by_class_id
@classid Integer, the classes.class_id for a particular class
@topcount Integer, top number of entries to display
Displays the most common demographics

f_lcd_by_class_id
@classid Integer, the classes.class_id for a particular class
@bottomcount Integer, bottom number of entries to display
Displays the least common demographics

Invoices/Revenue

v_invoices_all
All Invoices

f_invoices_by_class_id
@classid Integer, the classes.class_id of a particular class
All invoices related to one class

f_invoices_by_member_id
@memberid Integer, the member.member_id of a member
All invoices for a particular member

f_invoices_by_member_email
@memberid Varchar (255), the member.member_email of a member
All invoices for a particular member

f_revenue_by_class_id
@classid Integer, the classes.class_id of a particular class
All invoices associated to a particular class. Determines how much revenue this class produced.

f_revenue_by_leader_id
@leaderid Integer, the members.member_id key of the class leader
All invoices across all classes lead by a particular leader. Determines how much revenue this leader generated.

f_revenue_by_leader_timeframe
@start_time Datetime, Any class that begins on or after class.class_date_time_start
@end_time Datetime, Any class that end on or before class.class_date_time_start
Displays the revenue generated within the timeframe, grouped by leader id.

f_revenue_by_timeframe
@start_time Datetime, Any class that begins on or after class.class_date_time_start
@end_time Datetime, Any class that end on or before class.class_date_time_start
Displays the total revenue generated within the timeframe

External Program Logic

p_password_validate
@password Varchar (255), the plain text passphrase
@memberid Integer, the members.member_id of a member
Verify if a password is correct. We don't know the 'correct' password value, just if it matches after encrypting. Must supply the guessed password and member_id value. Returns 1 if passwords match, returns 0 if passwords do not match.

p_password_change
@oldpassword Varchar (255), the prior plain text passphrase
@newpassword Varchar (255), the new plain text passphrase
@memberid (Integer), the members.member_id of a member
This procedure will change a member's password. The correct old password must be supplied. If the old password is incorrect, an exception will be thrown.

p_member_create
@member_id Integer
@member_email Varchar (255)
@member_firstname Varchar (80)
@member_lastname Varchar (80)
@member_religion Varchar (80)
@member_birthday Date
@member_phone Varchar (50)
@member_language_1 Varchar (100)
@member_language_2 Varchar (100)
@member_language_3 Varchar (100)
@member_language_4 Varchar (100)
@member_language_5 Varchar (100)
@member_origin_country Varchar (255)
@member_race Varchar (255)
@member_gender Varchar (255)
@member_pronoun Varchar (255)
@member_political_affiliation Varchar (255)
@member_password Varchar (2000)
Create a new member. If member_id is present, be sure to do a 'SET identity_insert dbo.members ON' first, otherwise the insert will fail. If you are allowing us to automatically determine the member_id (default behavior), set @member_id to 0.

p_member_update

@member_id Integer
@member_email Varchar (255)
@member_firstname Varchar (80)
@member_lastname Varchar (80)
@member_religion Varchar(80)
@member_birthday Date
@member_phone Varchar (50)
@member_language_1 Varchar (100)
@member_language_2 Varchar (100)
@member_language_3 Varchar (100)
@member_language_4 Varchar (100)
@member_language_5 Varchar (100)
@member_origin_country Varchar (255)
@member_race Varchar (255)
@member_gender Varchar (255)
@member_pronoun Varchar (255)
@member_political_affiliation Varchar (255)
Update a member's information based on the passed in @member_id. We do not do differential updates, so you must supply all the fields. Note that this procedure will not let you change the member's password. You must use p_password_change for that.

p_class_create
@class_id Integer
@class_name Varchar (255)
@class_leader_id Integer
@class_date_time_start Datetime
@class_date_time_end Datetime
@class_description Varchar (8000),
@class_price Decimal (10,2),
@class_max_size Decimal (4,0)
Create a new class. If class_id is present, be sure to do a 'SET identity_insert dbo.classes ON' first, otherwise the insert will fail. If you are allowing us to automatically determine the class_id (default behavior), set @class_id to 0.

p_class_update
@class_id Integer
@class_name Varchar (255)
@class_leader_id Integer
@class_date_time_start Datetime
@class_date_time_end Datetime
@class_description Varchar (8000)
@class_price Decimal (10,2),
@class_max_size Decimal (4,0)
Update class information based on the passed in @class_id. We do not do differential updates, so you must supply all the fields. Note that if you reduce the class_max_size, we do not unenroll any member. That class will effectively be oversubscribed, but no additional members will be able to enroll.

p_class_enroll_member
@class_id Integer, the classes.class_id of a particular class

@member_id Integer, the members.member_id of a particular member
Enroll a member into a class

p_class_unenroll_member
@class_id Integer, the classes.class_id of a particular class
@member_id Integer, the members.member_id of a particular member
Unenroll a member from a class. Note that this procedure will not remove any associated invoices. We need to wait for the business to determine how refunds/cancellations work before we will implement that logic. This delete action cannot be undone.

Internal Program Logic

f_generate_password
@password Varchar (255), the plain text passphrase
@salt Varchar (36), the encryption salt value
Generate a hashed, salted password.

f_validate_password
@password Varchar (255), the plain text passphrase
@salt Varchar (36), the encryption salt value
@encryptedpassword Varchar (2000), the encrypted password
Returns 1 if passwords match, returns 0 if passwords do not match.

Triggers

t_max_class_size (Instead Of Insert)
This trigger will check if a class is full before allowing a member to register. First, it counts the current number of members registered for a class from the class_members table. Then, it finds out the max class size from the classes table. If the number of enrolled members is at or above capacity, disallow the insert. We are not allowing API access (via stored procedures) for a member to transfer classes. They will only be inserted or deleted, never updated.

t_encryptpassword (Instead Of Insert)
This trigger ensures the member password is always encrypted with a unique salt. Supports putting in with a member_id present or not. If member_id is present, be sure to do a 'SET identity_insert dbo.members ON' first, otherwise it will fail.

t_invoice_enrolled_member (After Insert)
This trigger is designed to automatically invoice a member upon registration. It should only fire if the class is not free. The business decided not to invoice free classes.