Information Builders

iWay

Omni-HealthData™ HealthViews User's Guide

Version 3.11

November 18, 2019

Contents

Preface

This documentation provides prerequisites and instructions to configure Omni-HealthData HealthViews.

How This Manual Is Organized

This manual includes the following chapters:

	Chapter/Appendix	Contents
1	Configuring Omni-HealthData HealthViews	Provides an overview for Omni-HealthData HealthViews.
2	Installing and Setting Up HealthViews for Microsoft SQL Server	Describes how to install and set up the HealthViews database for Microsoft SQL Server.
3	Installing and Setting Up HealthViews for PostgreSQL	Describes how to install and set up the HealthViews database for PostgreSQL.
4	HealthViews Scripts	Describes the HealthView scripts and the tables and views they create.

Documentation Conventions

The following table lists and describes the documentation conventions that are used in this manual.

Convention	Description

`THIS TYPEFACE`
or
`this typeface`
	Denotes syntax that you must type exactly as shown.

this typeface	Represents a placeholder (or variable), a cross-reference, or an important term. It may also indicate a button, menu item, or dialog box option that you can click or select.
underscore	Indicates a default setting.
Key + Key	Indicates keys that you must press simultaneously.

Convention	Description
{}	Indicates two or three choices. Type one of them, not the braces.
|	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis (...).
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

Visit our Technical Documentation Library at *http://documentation.informationbuilders.com*. You can also contact the Publications Order Department at (800) 969-4636.

Customer Support

Do you have any questions about this product?

Join the Focal Point community. Focal Point is our online developer center and more than a message board. It is an interactive network of more than 3,000 developers from almost every profession and industry, collaborating on solutions and sharing tips and techniques. Access Focal Point at *http://forums.informationbuilders.com/eve/forums*.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our website, *http://www.informationbuilders.com*. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of *http://www.informationbuilders.com* also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

Call Information Builders Customer Support Services (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (*xxxx.xx*) when you call.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Help Us to Serve You Better

To help our consultants answer your questions effectively, be prepared to provide specifications and sample files and to answer questions about errors and problems.

The following tables list the environment information our consultants require.

Platform	
Operating System	
OS Version	
JVM Vendor	
JVM Version	

The following table lists additional questions to help us serve you better.

Request/Question	Error/Problem Details or Information
Did the problem arise through a service or event?	
Provide usage scenarios or summarize the application that produces the problem.	
When did the problem start?	
Can you reproduce this problem consistently?	
Describe the problem.	

Request/Question	Error/Problem Details or Information
Describe the steps to reproduce the problem.	
Specify the error message(s).	
Any change in the application environment: software configuration, EIS/database configuration, application, and so forth?	
Under what circumstance does the problem *not* occur?	

The following is a list of error/problem files that might be applicable.

- ❑ Input documents (XML instance, XML schema, non-XML documents)
- ❑ Transformation files
- ❑ Error screen shots
- ❑ Error output files
- ❑ Trace files
- ❑ Custom functions and agents in use
- ❑ Diagnostic Zip
- ❑ Transaction log

User Feedback

In an effort to produce effective documentation, the Technical Content Management staff welcomes your opinions regarding this document. Please use the Reader Comments form at the end of this document to communicate your feedback to us or to suggest changes that will support improvements to our documentation. You can also contact us through our website, *http://documentation.informationbuilders.com/connections.asp*.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our website (*http://education.informationbuilders.com*) or call (800) 969-INFO to speak to an Education Representative.

Chapter 1

Configuring Omni-HealthData HealthViews

Omni-HealthData HealthViews is a set of scripts that implements denormalized tables and views.

Omni-HealthData HealthViews enables organizations to leverage existing clinical data, generating valuable actionable insights, which lead to tangible business results. Leveraging Omni-HealthData HealthViews to compile and relate content across the entire organizational spectrum, business users can be empowered to communicate, visualize, and analyze data effectively.

This section provides an overview for Omni-HealthData HealthViews and key features, in addition to describing the configuration steps that are required.

In this chapter:

Omni-HealthData HealthViews Components Configuration Overview

Omni-HealthData HealthViews offers a commercial, off the shelf, clinical and operational data model with dynamic views to empower healthcare analytics and reporting.

Omni-HealthData HealthViews provides a centralized data model, which is organized by Domain and Subject. In addition, client expansion and customization are supported.

Key features include:

- Centralized model to collect disparate healthcare data.
- Self-service healthcare business intelligence.
- Framework for developing clinical-centric analytic and data mining applications.
- Analysis for Patient Movement, Demographics, Habits, Outcomes, Volumes, Infections, and more.
- Clinical health care COTS (commercial off the shelf) data model.
- Multi-tenant data storage, allowing for custom content to be stored and integrated with the Omni-HealthData HealthViews data model.

- Data model relationships clearly defined to expedite content creation (reports, analytics).
- Optimized data model to ensure rapid answers to difficult questions.
- Capability of Omni-HealthData HealthViews to host data that generates Healthcare Performance Analytics dashboards, and balance scorecards.
- HealthViews statistics, including batch run time, records loaded per table, records added per table since the last load, and so on.

Understanding the Architecture of Omni-HealthData HealthViews

Omni-HealthData HealthViews implements a star schema model that has been denormalized for ease of use for reporting purposes. It is implemented as dynamic views over tables and provides the ability to view patient facts by date and time and also correlated clinical event data.

Key components include:

- **Healthcare data model (star schema).** Definition of common healthcare data and relationship for storage and reporting.
- **Dynamic database views of the data model.** A layer of abstraction of the data model to simplify business intelligence reporting.

Omni-HealthData HealthViews Prerequisites and Supported Platforms

Omni-HealthData HealthViews is currently supported on the following platforms:

- Microsoft Windows Server 2003 and higher. Microsoft SQL Server 2012 and higher as the database server.
- PostgreSQL Version 9.3 and higher.

HealthViews Distribution

HealthViews is distributed through a HealthViews.zip file that contains the following directories:

- Docs - contains documentation, including this manual.
- Setup_scripts - contains scripts used to set up the HealthViews environment.
- Stored_procs - contains stored procedures used by HealthViews.

The HealthViews scripts are located in the root directory of the zip file.

In addition:

- For Microsoft SQL Server, a command-line utility for executing the scripts as a batch can be found in the root directory of the zip file.
- For PostgreSQL, a bash script to run the scripts is stored in the root directory of the zip file.

Chapter 2

Installing and Setting Up HealthViews for Microsoft SQL Server

This section describes how to install and set up the HealthViews database for Microsoft SQL Server.

In this chapter:

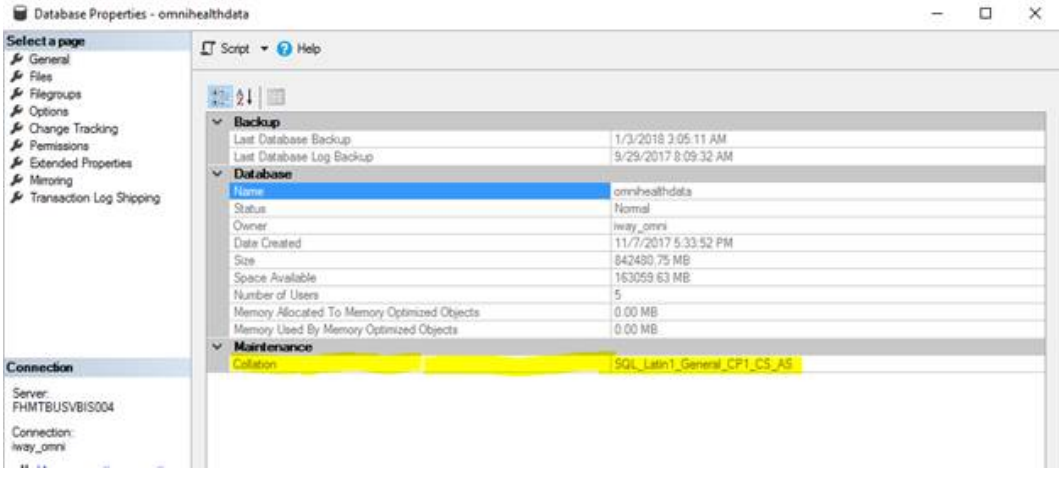
Installing HealthViews for Microsoft SQL Server

To install HealthView for Microsoft SQL Server, expand the HealthViews .zip file on the machine that will be running the HealthViews batch process. It is recommended to set up a HealthViews directory, and within that directory create subdirectories for each environment to be supported (DEV, PROD, QA, and so on).

Setting Up the HealthViews Database

In a Microsoft SQL Server environment, HealthViews run on their own database called healthviews. Use SQL Server Management Studio to create the database. Once it is created, the collation for the database should be set to match the collation of the omnihealthdata database. To check the collation of omnihealthdata using SQL Server Management Studio, right-click the database and select *Properties*.

The Database Properties dialog box displays. The collation is in the General tab, as shown in the following image.



To change the collation of healthviews to match omnihealthdata, use the following commands:

```
Use Master;
GO
ALTER DATABASE healthviews
SET SINGLE_USER
WITH ROLLBACK IMMEDIATE;
GO
Alter database healthviews
Collate SQL_Latin1_General_CP1_CS_AS;
GO
ALTER DATABASE healthviews
SET MULTI_USER;
GO
```

Installing the Stored Procedures

Once the healthviews database has been created, four functions need to be installed:

- hash64
- split_part
- p_hvtabstatcollector
- p_count_ohd_table_rows

Scripts for all the stored procedures can be found in the \stored_procs folder in the HealthViews.zip file.

Installing the hash64 Function

HealthViews uses the hash64 function to create unique hashed values for ID fields. Using hashed values for ID fields allows for faster joins.

The hash64.sql script is provided for installing the hash64 function. Execute the script in the healthviews database. After running, dbo.hash64 should be listed as a scalar-valued function for the healthviews database.

Installing the split_part Function

The split_part.sql script is provided for installing the split_part function. The split_part function is used to divide data fields into smaller blocks and only use part of that block. Execute the script in the healthviews database. After running, dbo.split_part should be listed as a scalar-valued function in the healthviews database.

Installing the p_hvtabstatscollector Function

The fn_hvtabstatscollector function is used to compile statistics on the HealthViews data. It tracks the amount of records in each HealthViews table and also the number of records added since the last time the table was updated.

The p_hvtabstatscollector.sql script is provided for installing the p_tabstatscollector function. Execute the script in the healthviews database. After running, dbo.p_tabstatscollector should be listed as a stored procedure in the healthviews database.

Installing the p_count_ohd_table_rows Function

The p_count_ohd_table_rows function is used to determine the number of rows in each table in the omnihealthdata database, which can be useful for comparing to HealthViews.

The p_count_ohd_table_rows.sql script is provided for installing the p_count_ohd_table_rows function. Execute the script in the omnihealthdata database. After running, dbo. p_count_ohd_table_rows should be listed as a stored procedure in the omnihealthdata database.

Omnihealthdata Indexes

HealthViews uses indexes on certain Omni-HealthData tables to improve performance. The 001_create_omni_indexes.sql script, located in the \Setup_scripts directory of the HealthViews.zip file, is provided to create the indexes. This script should be run prior to the initial running of HealthViews to create the indexes and re-run if the omnihealthdata database is recreated.

Setting Up the HealthViews Batch Process

HealthViews is implemented using a batch of SQL scripts that retrieve data from the omnihealthdata database and denormalize the data. The running of the scripts is automated through a command file, build_all.cmd.

Installing the HealthViews Scripts

All HealthViews scripts, as well as build_all.cmd, should be installed in the same directory. It is recommended to use a separate directory for each database instance to be supported (DEV, PROD, QA, and so on).

Configuring the HealthViews Process

The first part of the build_all.cmd file sets environmental variables that are used by the HealthViews scripts. The settings are as follows:

```
set target_server=name of the server hosting the database
set user=user name to execute the scripts
set password=password for that user
set target_db=target database, usually healthviews
set source_db=source database, usually omnihealthdata
set check_source=whether or not to check if the script needs to be run
set debug_query=turns debugging of source check on and off
```

Executing the Scripts

The build_all.cmd file gets a directory of every .sql script in the current directory and runs them in order, using the configuration information defined above. Scripts are run singly, no scripts run concurrently.

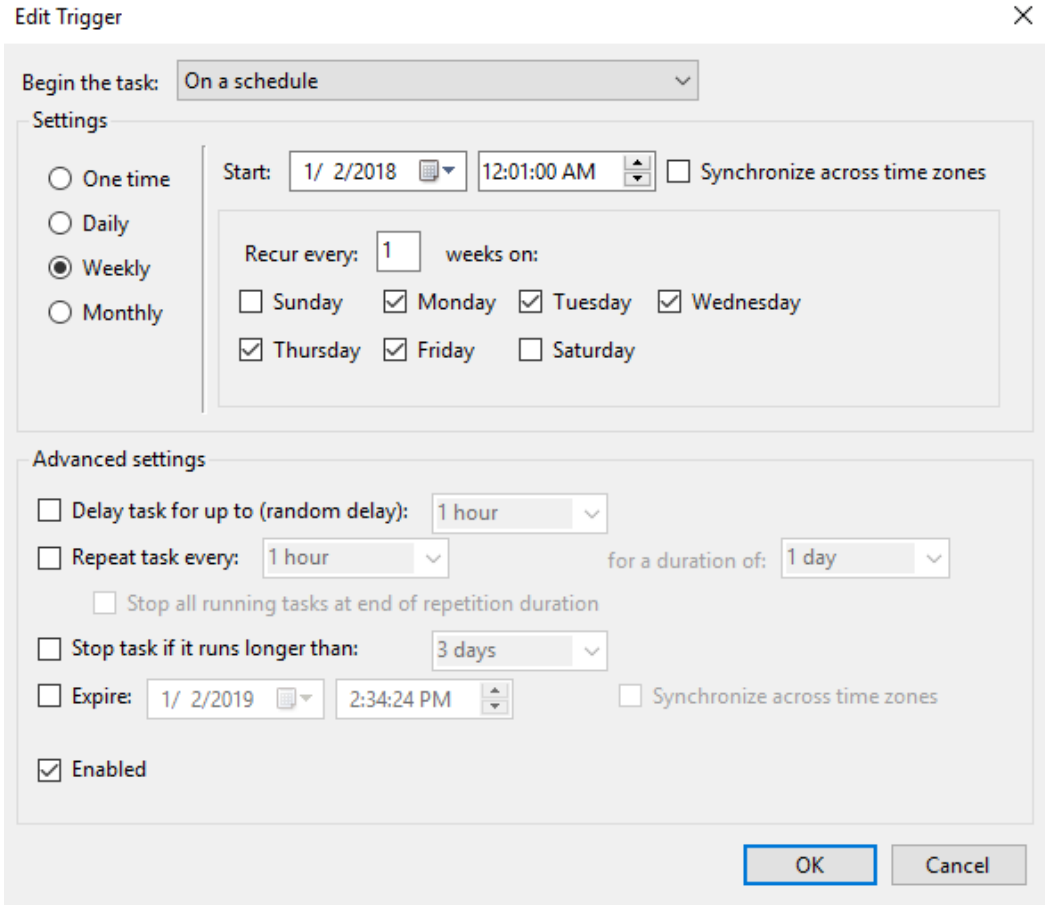
To execute the scripts, open a command prompt in the directory containing the scripts and the build_all.cmd file. At the prompt type:

```
build_all.cmd > [logfile_name].log
```

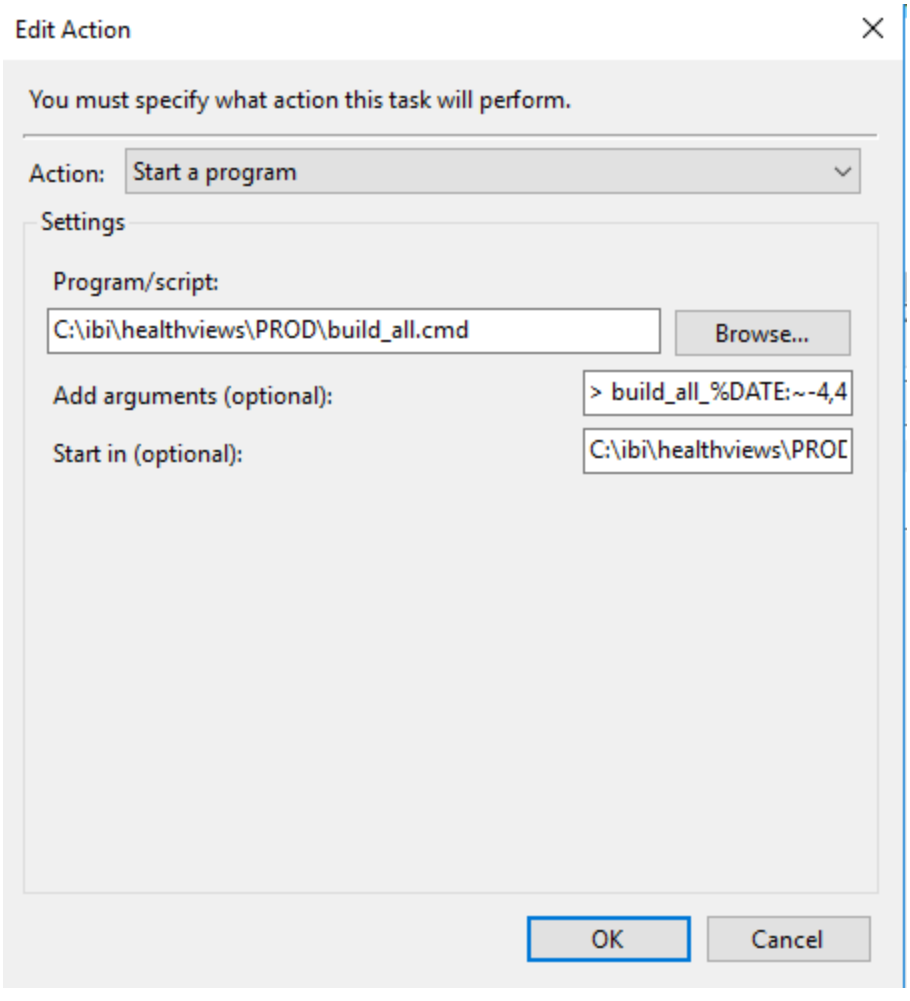
It is recommended to pipe the output of build_all.cmd to a log file so that any errors can be noted and debugged.

Automating the HealthViews Process

The Windows Task Scheduler can be used to automate the running of the HealthViews process. Through the Task Scheduler, you can configure HealthViews to run at a specific time on specific days of the week, as shown in the following image.



Output from build_all.cmd can be directed to a log file, indicating the date as part of the file name, as shown in the following image.



Chapter 3

Installing and Setting Up HealthViews for PostgreSQL

This section describes how to install and set up the HealthViews database for PostgreSQL.

In this chapter:

Installing HealthViews for PostgreSQL

To install HealthViews for PostgreSQL, expand the HealthViews .zip file on the machine that will be running the HealthViews batch process. It is recommended to set up a HealthViews directory, and within that directory create subdirectories for each environment to be supported (DEV, PROD, QA, and so on).

Setting Up the HealthViews Schema

It is recommended that the HealthViews materialized views be located in their own schema, typically called healthviews. The schema should be owned by the role responsible for loading the HealthViews materialized views. Using an administrative role, type the following:

```
create schema healthviews;
alter schema healthviews owner to hv_user;
```

Installing the Stored Procedures

Once the healthviews database has been created, three functions need to be installed:

- hash64
- fn_hvtabstatcollector
- fn_hvpartstatcollector

Scripts for all the stored procedures can be found in the \stored_procs folder in the HealthViews.zip file.

Installing the hash64 Function

HealthViews uses the hash64 function to create unique hashed values for ID fields. Using hashed values for ID fields allows for faster joins.

The hash64 function uses a compiled C object and is PostgreSQL version specific. Information Builders distributes a version compiled for PostgreSQL 9.3 and PostgreSQL 9.6. The compiled object can be found in a \9.3 or \9.6 subdirectory, depending on the version. A script to register the function (register.sh) can be found in the \source directory.

Copy the appropriate ibi_functions.so and register.sh files to a common directory. Then, set the permissions to execute by typing:

```
sudo chmod 777 ibi_functions.so
sudo chmod 777 register.sh
```

The register.sh shell script contains the following:

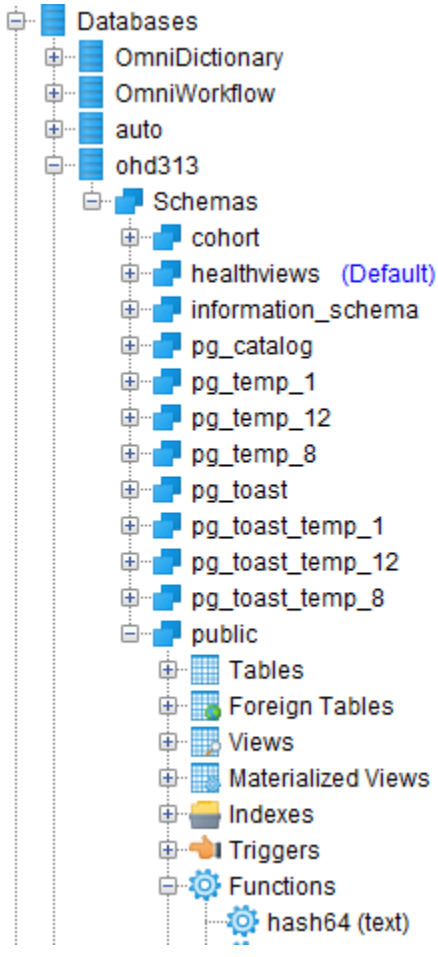
```
#!/bin/bash
PGHOME="$PWD"
UDFHOME="$PWD"
if [[ $# -ne 1 ]]; then
  echo "USAGE: register.sh <DB_NAME>"
  exit 1
fi
L_DBNAME=${1}
psql -d ${L_DBNAME} <<EOF
create or replace function hash64(text) returns int8
as '$UDFHOME/ibi_functions.so', 'hash64'
language C strict immutable;
EOF
```

It is necessary to execute register.sh as the postgres user. Ensure that there are appropriate permissions for the entire directory tree to the register.sh and ibi_functions.so files before proceeding.

Next, register the function by typing:

```
./register.sh [db_name]
```

where [*db_name*] is the name of the database being used by Omni-HealthData and HealthViews. When the script completes, there should be a hash64 function listed in the public schema for your database, as shown in the following image.



For PostgreSQL versions other than 9.3 and 9.6, the source files used to compile ibi_functions.so can be found in the \source directory. You will also need the PostgreSQL header files installed on your system. Use your package manager to install the header files appropriate for your version of PostgreSQL. For example, Debian distributions of PostgreSQL would use package postgresql-server-dev-9.5, which can be installed by typing:

```
sudo apt-get install postgresql-server-dev-9.5
```

Installing the fn_hvtabstatscollector Function

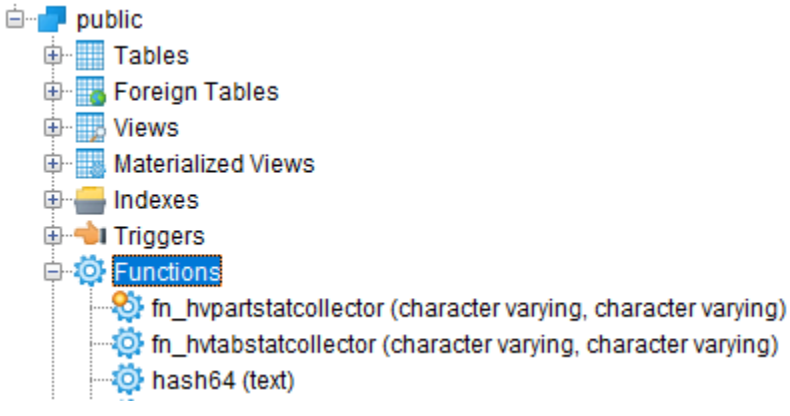
The fn_hvtabstatscollector function is used to compile statistics on the HealthViews data. It tracks the number of records in each HealthViews table and also the number of records added since the last time the table was updated.

The fn_hvtabstatscollector.sql script is provided for installing the fn_tabstatscollector function. Execute the script in the public schema. After running, fn_tabstatscollector should be listed as a stored procedure in the public schema, as shown in the following image.

Installing the fn_hvpartstatcollector Function

The fn_hvpartstatcollector function is used to collect statistics on any partitions in the healthviews database.

The fn_hvpartcollector.sql script is provided for installing the fn_hvpartstatcollector function. Execute the script in the public schema. After running, fn_hvpartstatcollector should be listed as a stored procedure in the public schema, as shown in the following image.



Other Setup Steps

A table, hv_availability, needs to be created in the public schema. The aa_hv_availability.sql script for this is provided in the setup_scripts directory. Run this script in the public schema prior to the first load of Healthviews.

Setting Up the HealthViews Batch Process

HealthViews is implemented using a batch of SQL scripts that retrieve data from the omnihealthdata database and denormalize the data. The running of the scripts is automated through a shell script, build_all.sh.

The script first configures variables:

```
L_PROGNAME=$(basename $0)
L_DB_CONFIG=db_config/postgresql.properties
L_TEMPFILE=$(mktemp /tmp/healthviews-XXXX)
L_SQL_FILES=$(\ls *.sql)
```

The script assumes that the configuration file will be located in a subdirectory named db_config. Change this value if a different directory is used. It also gets a list of all SQL scripts in the current directory. The scripts are processed in order later in the script:

```
echo "start: ${L_SQL_FILE} $(date)"
psql -h ${database_host} -U ${database_username} -d ${database_name}
-q -f ${L_TEMPFILE}
echo "end: ${L_SQL_FILE} $(date)"
```

The two echo statements are used to create log file entries. The script then calls each HealthViews script and executes it.

Installing the HealthViews Scripts

All HealthViews scripts, as well as build_all.sh should be installed in the same directory. It is recommended to use a separate directory for each database instance to be supported (DEV, PROD, QA, and so on).

Configuring the HealthViews Process

A configuration file, postgresql.properties, is used to configure the HealthViews batch process. This file should be located in a \db_config directory, under the main HealthViews directory.

The settings are as follows:

```
database.type=the type of database.  Should be set to POSTGRESQL
database.host=the location of the database server.  Can be either a
logical name or a dotted IP address.
database.name=the name of the database for OmniHealthData and
HealthViews.
database.username=the user name to log in to the database with.
database.src_schema=the schema for OmniHealthData.  Typically public.
database.targ_schema=the schema for HealthViews.  Typically healthviews.
```

Executing the Scripts

The build_all.sh file takes a directory listing of every .sql script in the current directory and runs them in order, using the configuration information defined above. Scripts are run singly, no scripts run concurrently.

To execute the scripts, navigate to the directory containing build_all.sh and the HealthViews scripts. At the prompt, type:

```
./build_all.sh &> [logfile_name].log
```

It is recommended to pipe the output of build_all.sh (including stderr) to a log file so that any errors can be noted and debugged.

Automating the HealthViews Process

Generally, the HealthViews batch process is manually run on a regular basis. To automate this, use a cron job to schedule the running of the build_all.sh. For example:

```
[hv_user@hv_machine]$ crontab –e
```

opens the crontab for the current user for editing

```
01 00 * * 1,2,3,4,5 * \hv_dir\build_all.sh &> hv_dir/log/
build_healthviews.${L_TODAY}.log)
```

This creates a job scheduled to run at 12:01 AM every Monday, Tuesday, Wednesday, Thursday, and Friday with a log created with the date of the log in the file name in a \log directory. The first two digits (01) represent the minutes, the second two (00) represent the hour (using a 24-hour clock), the two consecutive asterisks indicate the job should be executed all months and all days of the month, the 1,2,3,4,5 represent the days of the week (Sunday is represented by either 0 or 7), and the final asterisk indicates the job should run all years. The entry then gives the command to be run, including sending the output to a log file.

Chapter 4

HealthViews Scripts

Each HealthViews script creates a materialized view using data from Omni-HealthData. In addition to the standard HealthViews materialized views, custom scripts can be used to populate dashboards, update important metrics, and so on. The Information Builders team can work with your organization to identify specific needs and create scripts to meet those needs. Contact your Information Builders representative to find out more information.

In this chapter:

Scripts and Tables

The scripts and the tables and views they create are as follows:

001_aa_t_hv_availability.sql - Script to record the start time of the HealthViews process, which can be used to track performance.

001_date_dimension.sql - Creates a materialized view for system dates in HealthViews.

002_time_dimension.sql - Creates a materialized view for system times in HealthViews.

003_t_codeset_lookup.sql - Creates a materialized view for all codesets in the system. This view is used by subsequent scripts to expand the code ID to include description, parent, and parent description.

003_t_dim_codemap_lookup.sql - Creates a materialized view for all codemaps in the system, including the type of code map, the concept (drug, diagnosis, and so on), the mapping Created (for example, SNOMED CT to ICD-9), and so on.

005_t_dim_patient.sql - Creates a materialized view for all patient records in the system, including demographic and contact information.

005_t_dim_patient_master.sql - Creates a materialized view for all patient records in the system, including demographic and contact information. Although the script is intended for mastered data, it will also support early stage deployments when patient mastering has typically not been implemented yet.

006_t_dim_provider.sql - Creates a materialized view for all provider records in the system, including contact and demographic information, as well as status (active, inactive, and so on). Although the script is intended for mastered data, it will also support early stage deployments when provider mastering has typically not been implemented yet.

007_t_dim_facility.sql - Creates a materialized view for all facility records in the system including location and status (active or inactive).

007_t_dim_facility_master.sql - Creates a materialized view for all mastered facility records in the system, including location, status, and type.

007_t_facility_location.sql - Creates a materialized view for all facility location records in the system.

008_t_dim_organization.sql - Creates a materialized view for all organization records in the system.

008_t_dim_organization_master.sql - Creates a materialized view for all mastered organization records in the system.

008_t_organizational_unit.sql - Creates a materialized view for all organizational unit records in the system.

010_t_encounter.sql - Creates a materialized view for all encounter records in the system, including the type of encounter, the patient, the arrival, and departure time, and so on.

010_t_provider_specialty.sql - Creates a materialized view for all provider specialty records in the system.

010_t_provider_specialty_master.sql - Creates a materialized view for all mastered provider specialty records in the system.

011_t_admission_event.sql - Creates a materialized view for all admission event records in the system, including the patient, provider, and facility, as well as the event time. All admission events are also joined to their corresponding encounters.

011_t_admission_event_provider.sql - Creates a materialized view of all admission event provider records in the system, including the admission event ID which can be used to join back to the event.

012_t_death_event.sql - Creates a materialized view for all death event records in the system, including the patient, provider, and facility, as well as the event time. All death events are also joined to their corresponding encounters.

012_t_death_event_provider.sql- Creates a materialized view for all death event provider records in the system, including the death event ID which can be used to join back to the event.

013_t_diagnosis_event.sql - Creates a materialized view for all diagnosis event records in the system including the patient, provider, and facility, as well as the event time. All diagnosis events are also joined to their corresponding encounters.

013_t_diagnosis_event_provider.sql - Creates a materialized view for all diagnosis event provider records in the system, including the diagnosis event ID which can be used to join back to the event.

014_t_discharge_event.sql - Creates a materialized view for all discharge event records in the system, including the patient, provider, and facility, as well as the event time. All discharge events are also joined to their corresponding encounters.

014_t_discharge_event_provider.sql - Creates a materialized view of all discharge event providers in the system, including the discharge event ID which can be used to join to the event.

015_t_procedure_event.sql - Creates a materialized view for all procedure event records in the system, including the patient, provider, and facility, as well as the event time. All procedure events are also joined to their corresponding encounters.

015_t_procedure_event_provider.sql - Creates a materialized view for all procedure event providers in the system, including the procedure event ID which can be used to join back to the event.

016_t_observation_event.sql - Creates a materialized view for all observation event records in the system, including the patient, provider, and facility, as well as the event time. All observation events are also joined to their corresponding encounters.

016_t_observation_event_provider.sql - Creates a materialized view of all observation event providers in the system including the observation event ID which can be used to join back to the event.

017_t_medication_admin_event.sql - Creates a materialized view for all medication administration event records in the system, including the patient, prescription information, route for the medication, and so on. All medication administration events are also joined to their corresponding encounters.

017_t_medication_admin_event_provider.sql - Creates a materialized view for all medication administration event providers in the system, including the medication administration event ID which can be used to join back to the event.

017_t_medication_order_event.sql - Creates a materialized view for all medication order event records in the system, including the patient, provider, and facility, as well as the event time. All medication order events are also joined to their corresponding encounters.

017_t_medication_order_event_provider.sql - Creates a materialized view for all medication order event provider records in the system, including the medication order event ID which can be used to join back to the event.

018_t_diagnostic_test_order_event.sql - Creates a materialized view for all diagnostic test order events in the system, including the patient, the diagnostic test code, the diagnostic test procedure, and so on. All diagnostic test order events are also joined to their corresponding encounters.

018_t_diagnostic_test_order_event_provider.sql - Creates a materialized view for all diagnostic test order event providers in the system, including the diagnostic test order event ID which can be used to join back to the event.

019_t_diet_order_event.sql - Creates a materialized view for all diet order events in the system, including the patient, the diet specification code, the service period, and so on. All diet order events are also joined to their corresponding encounters.

019_t_diet_order_event_provider.sql - Creates a materialized view for all diet order event provider records in the system, including the diet order event ID which can be used to join back to the event.

020_t_movement_event.sql - Creates a materialized view for all movement events in the system, including the patient, the facility, the type of move, and so on. All movement events are also joined to their corresponding encounters.

020_t_movement_event_provider.sql - Creates a materialized view for all movement event providers in the system, including the movement event ID which can be used to join back to the event.

021_t_pharmacy_dispense_event.sql - Creates a materialized view for all pharmacy dispense events in the system, including the patient, the prescription, the amount dispensed, and so on. All pharmacy dispense events are also joined to their corresponding encounters.

021_t_pharmacy_dispense_event_provider.sql - Creates a materialized view for all pharmacy dispense event providers in the system, including the pharmacy dispense event ID which can be used to join back to the event.

022_t_medical_supply_order_event.sql - Creates a materialized view for all medical supply order events in the system, including part number, the quantity, and so on. All medical supply order events are also joined to their corresponding encounters.

022_t_medical_supply_order_event_provider.sql - Creates a materialized view for all medical supply order event providers in the system, including the medical supply order event ID which can be used to join back to the event.

023_t_procedure_order_event.sql - Creates a materialized view for all procedure order events in the system, including the procedure code, the patient, and so on. All procedure order events are joined to their corresponding encounters.

023_t_procedure_order_event_provider.sql - Creates a materialized view for all procedure order event providers in the system, including the procedure order event ID which can be used to join back to the event.

024_t_radiology_event.sql - Creates a materialized view for all radiology events in the system, including the patient, the body part to receive treatment, the modality, and so on. All radiology order events are joined to their corresponding encounters.

024_t_radiology_event_provider.sql - Creates a materialized view for all radiology event providers in the system, including the radiology order event ID which can be used to join back to the event.

025_t_referral_order_event.sql - Creates a materialized view for all referral order events in the system, including the patient, the referral description, the indicating condition, and so on. All referral order events are joined to their corresponding encounters.

025_t_referral_order_event_provider.sql - Creates a materialized view for all referral order event providers, including the referral order event ID which can be used to join back to the event.

026_t_service_order_event.sql - Creates a materialized view for all service order events in the system, including the patient, the service code, and so on. All service order events are joined to their corresponding encounters.

026_t_service_order_event_provider.sql - Creates a materialized view for all service order event providers in the system, including the service order event ID which can be used to join back to the event.

027_t_transfer_event.sql - Creates a materialized view for all transfer event records in the system, including the patient, the type of transfer (for example, admit), the destination facility, and so on. All transfer events are also joined to their corresponding encounters.

027_t_transfer_event-provider.sql - Creates a materialized view for all transfer event providers in the system, including the transfer event ID which can be used to join back to the event.

028_t_transfusion_order_event.sql - Creates a materialized view for all transfusion order events in the system, including the patient, the blood type, the amount of blood, and so on. All transfusion order events are joined to their corresponding encounters.

028_t_transfusion_order_event_provider.sql - Creates a materialized view for all transfusion order event providers, including the transfusion order event ID which can be used to join back to the event.

029_t_vaccination_admin_event.sql - Creates a materialized view for all vaccination administration events, including the patient, the vaccine, the manufacturer, and so on. All vaccination administration events are joined to their corresponding encounters.

029_t_vaccination_admin_event_provider.sql - Creates a materialized view for all vaccination administration event providers, including the vaccination admin event ID which can be used to join back to the event.

030_t_account.sql - Creates a materialized view for all account records in the system, including account type, the associated patient, and so on.

031_t_account_guarantor.sql - Creates a materialized view for all account guarantor records in the system.

033_t_account_transaction.sql - Creates a materialized view for all account transaction records in the system, including the amount of the transaction, the payer ID, the account associated with the transaction, the date and time of the transaction, any adjustments, and the cost center associated with the transaction.

036_t_charge.sql - Creates a materialized view for all charge records in the system, including the charge amount, the cost center, the associated account, and so on.

037_t_provider_claim.sql - Creates a materialized view for all provider claim records in the system, including the account, the patient, the claim amount, the status, and so on.

038_t_patient_health_plan.sql - Creates a materialized view for all patient health plan records in the system, including the plan number, the subscriber, the relation of the subscriber to the patient, and so on.

039_t_provider_practice_specialty_master.sql - Creates a materialized view for all mastered provider specialty practice records in the system, including whether they are a primary care physician, accept walk-ins, and so on.

040_t_provider_practice.sql - Creates a materialized view for all provider practice records in the system, including the provider, the physical and mailing address, the provider number, and so on.

040_t_provider_practice_master.sql - Creates a materialized view for all mastered provider practice records in the system, including the provider, the physical and mailing address, the provider number, and so on.

041_t_daily_census.sql - Creates a materialized view for all daily census records in the system, including facility ID, the admissions, the discharges, and so on.

042_t_cart_item.sql - Creates a materialized view for all cart items in the system including the item, the fulfillment facility, the return facility, and so on.

043_t_patient_preferred_provider.sql - Creates a materialized view for all patient preferred provider records in the system, including the provider ID and the start and end dates.

043_t_patient_preferred_provider_master.sql - Creates a materialized view for all mastered patient preferred provider records in the system, including the patient master ID, the provider master ID, and the start and end dates.

044_t_provider_license.sql - Creates a materialized view for all provider license records in the system, including the license type, the licensing authority, the license status, and so on.

044_t_provider_license_master.sql - Creates a materialized view of all mastered provider license records in the system, including the license type, the licensing authority, the license status, and so on.

045_t_provider_identifier.sql - Creates a materialized view of all provider identifiers in the system, including the type of identifier, the issuing authority, the start and end date, and so on.

045_t_provider_identifier_master.sql - Creates a materialized view of all mastered provider identifier records in the system, including the type of identifier, the issuing authority, the start and end date, and so on.

046_t_provider_privilege.sql - Creates a materialized view for all provider privilege records in the system, including the privilege code, the start and end date, the facility, and so on.

046_t_provider_privilege_master.sql - Creates a materialized view for all mastered provider privilege records in the system, including the privilege code, the start and end date, the facility, and so on.

047_t_encounter_health_plan_coverage.sql - Creates a materialized view for all encounter health plan coverage records in the system, including the insurance coverage type code and the patient health plan ID.

080_t_health_plan.sql - Creates a materialized view for all health plan records in the system, including the plan number, plan payer, financial code, and so on.

260_t_guarantor.sql - Creates a materialized view for all guarantor records in the system, including the payer, the relation of the payer to the patient, and so on.

290_t_payer.sql - Creates a materialized view for all payer records in the system, including the payer name and contact information.

311_t_episode.sql - Creates a materialized view for all episode records in the system, including the start and end date, the episode treatment group, the episode number, and so on.

316_t_specimen.sql - Creates a materialized view for all specimen records in the system, including the collection method, the source, the amount, and so on.

3**22_t_surgery_case.sql** - Creates a materialized view for all surgery case records in the system, including the facility, the related encounter ID, the patient, pre-operation tests, type of surgery, and so on.

324_t_surgery_case_cart.sql - Creates a materialized view for all surgery case cart records in the system.

326_t_surgery_procedure.sql - Creates a materialized view for all surgery procedure records in the system, including the type of procedure, the scheduled time for the procedure, the location, the anesthesia requirements, and so on.

325_t_surgery_movement.sql - Creates a materialized view for all surgery movement records in the system, including provider being relieved, the reason for the relief, and so on.

333_t_care_plan.sql - Creates a materialized view for all care plan records in the system, including the care plan status, the start date, and the end date.

334_t_accessibility_appointment.sql - Creates a materialized view for all accessibility appointment records in the system, including scheduling information, the location of the appointment, and so on.

335_t_practice_facility.sql - Creates a materialized view for all practice facility records in the system, including legal name for the practice and the address and phone number for the practice.

336_t_provider_practice.sql - Creates a materialized view for all provider practice records in the system, including practice name and the physical and mailing address of the practice.

337_t_referral_source.sql - Creates a materialized view for all referral source records in the system, including the patient, the provider, the requested service, and so on.

322_t_surgery_case.sql - Creates a materialized view for all surgery case records in the system, including the facility, the related encounter ID, the patient, pre-operation tests, type of surgery, and so on.

324_t_surgery_case_cart.sql - Creates a materialized view for all surgery case cart records in the system.

326_t_surgery_procedure.sql - Creates a materialized view for all surgery procedure records in the system, including the type of procedure, the scheduled time for the procedure, the location, the anesthesia requirements, and so on.

904_t_hvtabstats.sql - Registers record counts for all tables in HealthViews, which can be used to track database growth over time.

905_ohd_counts.sql - Records the record counts for all tables in the omnihealthdata database.

999_t_hv_availability.sql - Records the stop time and calculates the duration for the HealthViews process.

Code Sets

HealthViews has its own code set table, t_dim_codeset_lookup, which is used to expand upon the codes provided in Omni-HealthData tables. For example, the Omni-HealthData table, og_patient, contains a column for tax_id_type_code. HealthViews, in the v_dim_patient view, expands upon the code ID to also include the code description and the parent code ID and description. As a result, HealthViews views will contain more columns than the Omni-HealthData table from which they are derived.

Mastered and Non-Mastered Subjects

Certain subject areas with Omni-HealthData and HealthViews can contain mastered data, These include Patient, Provider, Facility, and Organization. Typically, customers do not master Patient or Provider at the beginning of their deployments. The mastering rules take time to develop and there needs to be a sufficient number of patients and providers for mastering to be meaningful. HealthViews represents the master ID for Patient and Provider, but will substitute the instance ID if those subjects have not been mastered. When mastering is implemented, no change in code is necessary. HealthViews will automatically switch to using the mastered values for those fields.

Selective Running of Scripts

In most Omni-HealthData deployments, daily data loads do not update every subject area. Logic is built into select HealthViews scripts to determine whether there is new data that needs to be loaded or not. If there is no new data, the script does not run. The logic compares record counts, the maximum omni_created_date, and the maximum omni_modified_date, to determine whether there are new records. This capability is currently found in the following four scripts:

- 030_t_account.sql
- 031_t_account_guarantor.sql
- 033_t_account_transaction.sql
- 036_t_charge.sql

Whether or not to check for new data is controlled through the set check_source= parameter in build_all.cmd. If this is set to Y, then the script will first check for the need to run. If it is set to anything other than Y, the script will run regardless of whether there is new data or not.

Reporting Indexes

Many scripts include one or more reporting indexes that are intended to improve the performance of other Omni-HealthData products, such as Cohort Builder. These indexes can be commented out if they are not needed and building the indexes impact performance. All reporting indexes are prefixed with a comment line indicating where they start.

Feedback

Customer success is our top priority. Connect with us today!

Information Builders Technical Content Management team is comprised of many talented individuals who work together to design and deliver quality technical documentation products. Your feedback supports our ongoing efforts!

You can also preview new innovations to get an early look at new content products and services. Your participation helps us create great experiences for every customer.

To send us feedback or make a connection, contact Sarah Buccellato, Technical Editor, Technical Content Management at *Sarah_Buccellato@ibi.com.*

To request permission to repurpose copyrighted material, please contact Frances Gambino, Vice President, Technical Content Management at *Frances_Gambino@ibi.com.*

Information Builders

iWay

Omni-HealthData™ HealthViews User's Guide

Version 3.11

DN3502337.1119

Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898