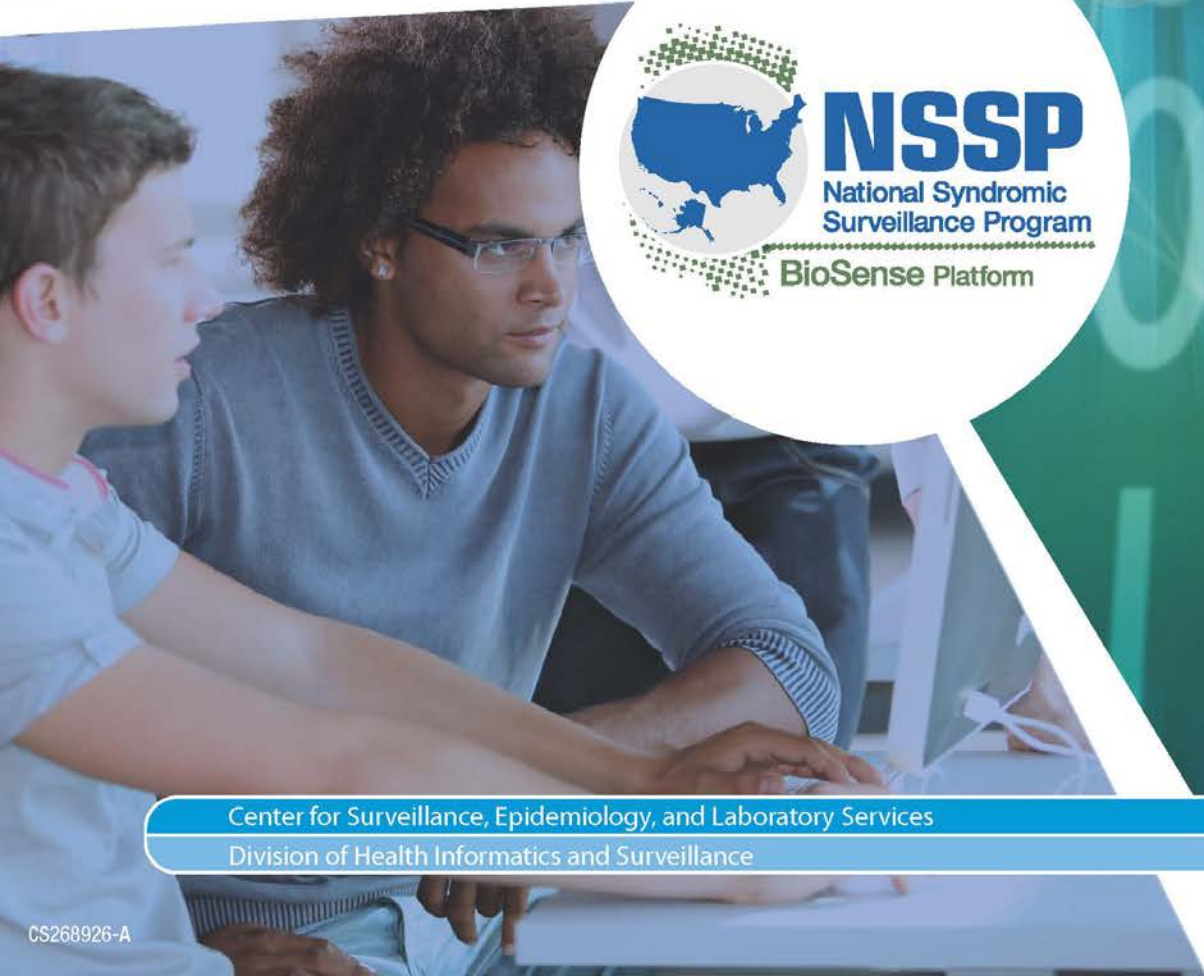


BioSense Platform Quick Start Guide to Using

Adminer

October 2016



Center for Surveillance, Epidemiology, and Laboratory Services
Division of Health Informatics and Surveillance



CONTENTS

1 Overview, 1

Log In to Adminer, 1

2 Basic Navigation, 2

Review Database Tables, 2

Create Free-form Query, 4

3 Frequently Asked Questions, 5

Appendix—Adminer SQL Query Examples, 6

Technical Assistance: support.syndromicsurveillance.org

The National Syndromic Surveillance Program (NSSP) promotes and advances development of the cloud-based BioSense Platform, a secure integrated electronic health information system that hosts standardized analytic tools and facilitates collaborative processes. The BioSense Platform is a product of the Centers for Disease Control and Prevention (CDC).

Quick Start Guide to Using Adminer

1. Overview

The Adminer tool lets you view SQL data stored on the BioSense Platform by using a web browser. You can use Adminer to verify your data within the BioSense Platform archive (raw, processed, and exceptions tables) and confirm information in your Master Facility Table (MFT).

Adminer also lets you to access a single database on the **BioSense_Platform** that contains views into all of your site’s data. When using Adminer, you will only have to log in to a single database to query and view multiple databases and data tables that contain data for your site.

Log In to Adminer

Go to the Adminer sign-in page located here:

<https://webquery.syndromicsurveillance.org/adminer/>

Adminer requires five inputs to successfully log in:

1. System: Always set to **MS SQL**
2. Server: Always set to **DATAMART**
3. Username: Format as **BIOSENSE\username** (replace “username” with your username)
4. Password: Input your password
5. Database: **BioSense_Platform**

Click on the permanent log-in check box to preserve your log-in credentials as a browser cookie.

Who should use Adminer?

Adminer is designed for users who are comfortable creating and running SQL queries. This guide will share the basics of Adminer and assumes an entry-level understanding of SQL.

Login

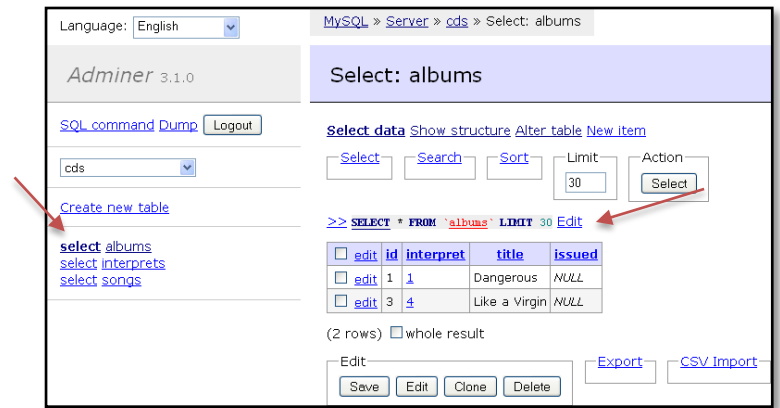
System	MS SQL ▾
Server	DATAMART
Username	BIOSENSE\username
Password
Database	BioSense_Platform

Permanent login

2. Basic Navigation

Review Database Tables

The left navigation bar will display all of the database tables to which you have access. To view the contents of a table, click the word “select” next to the table name. This will perform a generic SELECT* from that table and display the results. Results are limited to 50 records per page.



Once you select a table to view, you can modify the query that displays the table to meet your needs. Click “Edit” next to the query description. This will display a text box with a basic SQL query that you can modify.

The following table describes the information available in Adminer (i.e., the tables in the left navigation panel). The “XX” will be replaced with your site’s information. You will have access to both production and staging versions of your data tables. Production tables contain production-quality data sent to the BioSense Platform. Staging tables contain staging, or “test,” data that you may use during the facility and feed onboarding process. For questions about onboarding, contact the NSSP Service Desk at support.syndromicsurveillance.org.

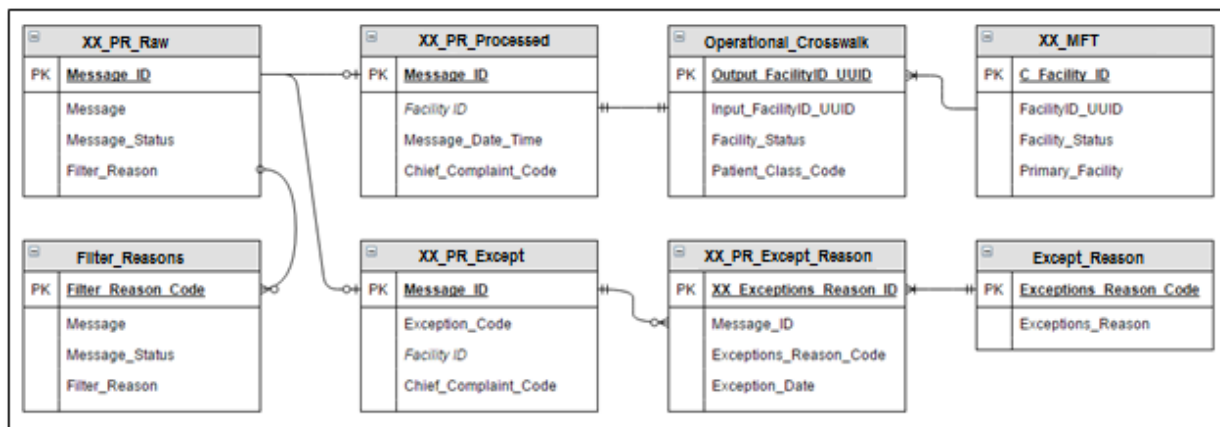
Before your site’s transition to the BioSense Platform, you may want to familiarize yourself with the following tables:

- *XX_MFT: the MFT loaded to the BioSense Platform after the MFT clean-up process*
- *XX_MFT_Except: MFT records that were unable to be uploaded*
- *XX_ST_Raw: the raw messages that will be processed against the new data flow*
- *XX_ST_Processed: the processed messages from the raw table*
- *XX_ST_Except and related tables: the exceptions messages from the raw table*
- *Filter_Reason: join to XX_ST_Raw to understand why a record was filtered (i.e., not attempted to be processed)*
- *Exceptions_Reason: join to XX_ST_Except_Reason to understand why a record was exceptioned*

Tables Available in Adminer	
Facility Information	
Name	Description
XX_MFT	Contains your current Master Facility Table (MFT). The baseline facility information in this table is populated from the MFT produced during clean-up. <i>To update or modify your MFT, contact the NSSP Service Desk.</i>
XX_MFT_Except	Contains facility records that could not be loaded to the MFT.
XX_Operational_Crosswalk	Contains the crosswalk that is used during production. Here, older or inactive facilities are mapped to newer and potentially active facilities.
XX_Site_Contacts (Coming Soon)	Contains contact information of all known site administrators for your site. (Coming Soon)

Tables Available in Adminer	
Production-quality Data	
<i>Name</i>	<i>Description</i>
XX_PR_Raw	Contains original message you delivered to the platform and some metadata about that message. If a message was filtered and designated invalid for syndromic surveillance, the Filter_Reason column will contain the code for why it was filtered.
XX_PR_Processed	Contains the processed message received and the calculated values built from the elements. This table will only contain messages that met the minimum processing criteria. Incomplete or invalid messages will be sent to XX_PR_Except.
XX_PR_Except	Contains the processed messages that did not meet minimum criteria for processing. The structure of the XX_PR_Except table and XX_PR_Processed table is identical. <i>To understand why a message appears in the XX_PR_Except table, you may want to join to the XX_PR_Except_Reason table.</i>
XX_PR_Except_Reasons	Contains the message_ID and any number of reasons for placing that record in the XX_PR_Except table. <i>To view the exception code descriptive values, join to Exceptions_Reason table.</i>
Staging Data (e.g., test data for feed/facility onboarding)	
<i>Name</i>	<i>Description</i>
XX_ST_Raw	Contains the original message you delivered to the platform and some metadata about that message. If a message was filtered and designated invalid for syndromic surveillance, the Filter_Reason column will contain the code for why it was filtered.
XX_ST_Processed	Contains the processed message received and the calculated values built from the elements. This table will only contain messages that met the minimum processing criteria. Incomplete or invalid messages will be sent to XX_ST_Except.
XX_ST_Except	Contains the processed messages that did not meet minimum criteria for processing. The structure of the XX_ST_Except table and XX_ST_Processed table is identical. <i>To understand why a message appears in the XX_ST_Except table, you may want to join to the XX_ST_Except_Reason table.</i>
XX_ST_Except_Reasons	Contains the message_ID and all the reasons for placing the record in the XX_ST_Except table. <i>To view the exception code descriptive values, join to Except_Reason table.</i>
Reference Tables	
<i>Name</i>	<i>Description</i>
Filter_Reasons	Provides a mapping from the Filter Reason Code found in the raw table to its descriptive text.
Exceptions_Reasons	Provides a mapping from the Exception Reason Code found in the Exceptions_Reasons table to its descriptive text.

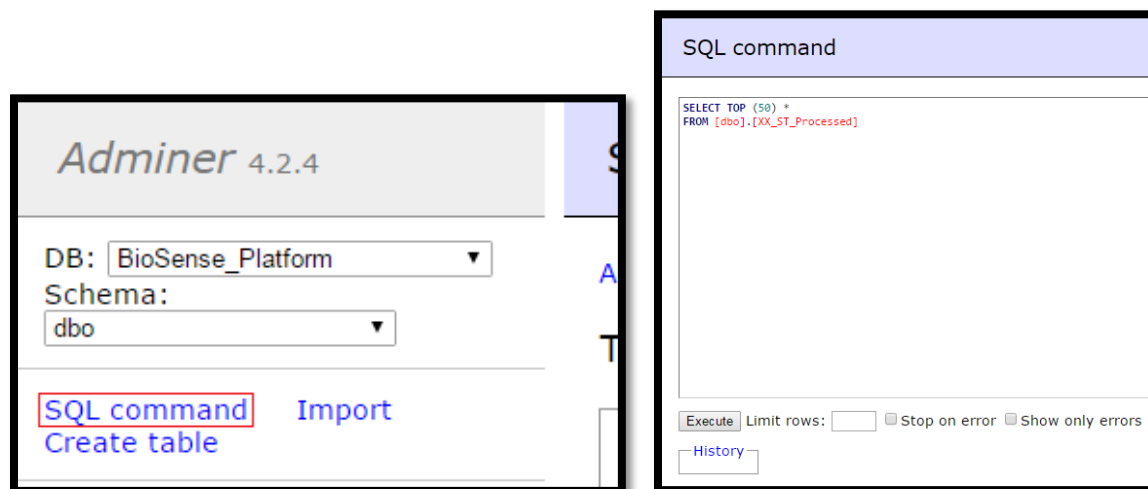
Below is an Entity Relationship Diagram of available BioSense Platform archive tables. You may find this helpful as you join tables to perform advanced queries.



Create Free-form Query

If you prefer to write your own SQL queries, first click the SQL command button.

This will open a blank text box where you can write and execute a free-form SQL query. Input your query, and then type a number in the Limit rows text box to control how many rows will be displayed. **A number less than 1000 is highly recommended.** Click Execute to carry out the query and view the results.



3. Frequently Asked Questions

Why did my Query fail?

Adminer is designed to provide a glimpse into the SQL tables that house your MFT and BioSense Platform archive syndromic surveillance data. For robust analytics, we recommend using other tools available through the BioSense Platform, including SAS Web Studio and R Professional.

To view your data in Adminer, it is highly recommended to limit the number of results returned by any SQL query. We recommend returning no more than the top 5,000 rows when designing your query. This can be done by placing a TOP (X) statement in your query to restrict the results to the number or records you place in “X.” For example, the following query would return the first 100 results from your processed table:

```
SELECT TOP (100) * FROM XX_ST_Processed
```

Where is my MFT?

Your facility information (MFT) is stored in the XX_MFT table that is accessible by clicking “Select” next to that table name.

As you review your facility information, remember to notify the Service Desk of changes or updates by submitting a ticket through support.syndromicsurveillance.org.

How do I add facility characteristics from my MFT to my archive processed data?

To add facility characteristics to your archive processed data, you will need to join your MFT table to your archive processed table. You should join on the variable C_BioSense_Facility_ID. For example:

```
SELECT TOP (100) * FROM XX_ST_Processed  
JOIN XX_Facility_Master  
ON XX_ST_Processed.C_BioSense_Facility_ID = XX_Facility_Master
```


Appendix—Adminer SQL Query Examples

Introduction

This appendix will help you and your team during the User Acceptance Testing (UAT) of the Adminer tool, part of the suite of tools available on the BioSense Platform. The appendix includes examples of basic SQL queries, which may serve as a helpful reference as you select and design your own queries for UAT. This appendix is not a comprehensive list of SQL queries. Instead, the examples will help you test basic querying scenarios that might be needed during typical interactions when using syndromic surveillance data.

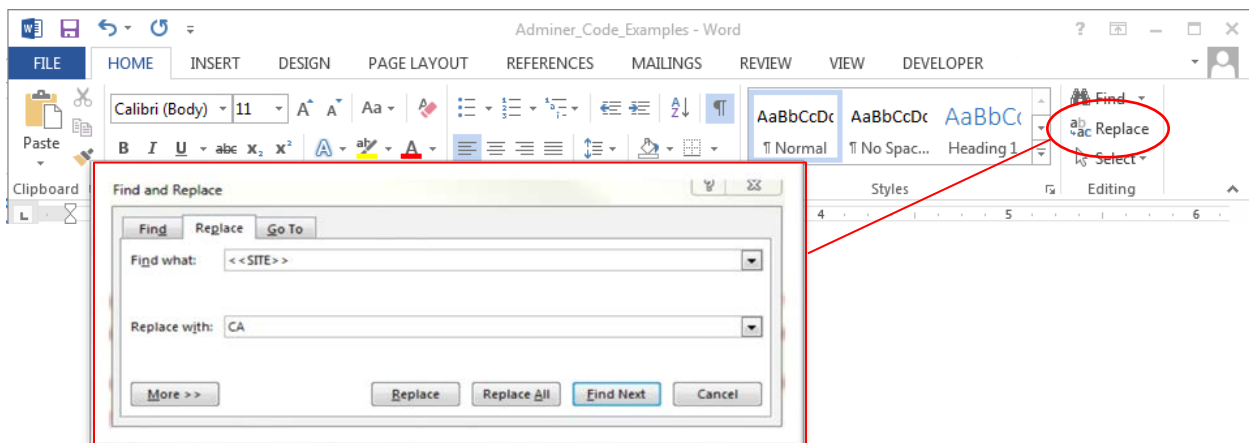
Sample queries use the three primary data tables available during UAT: *Master Facility Table (MFT)*, which is stored in the *Staging_Master_Facility* database; *raw* table; and *processed* table, which are stored in the *Staging_Archive* database. Here is a description of the data contained in each table:

- The *MFT* stores up-to-date information about site facilities. Your site provided this information to NSSP. Site-specific SQL views are provided for site access to the MFT.
- The *raw* table holds unprocessed, or “raw,” HL7 messages with minimal metadata included. Examples include time and date stamps along with the feed containing the original message.
- The *processed* table holds data from your HL7 messages that have been transformed according to data flow rules. Information about transformations that take place during data processing can be accessed through the International Society for Disease Surveillance (ISDS) website. Site-specific SQL tables are provided.

How to Use SQL Queries

First, go to the Adminer tool to enter SQL queries. Identify the query you want to use. Then substitute the query code for <<SITE>> and add your site abbreviation. Copy the query into the Adminer SQL editor. At the same time, you can adjust the dates as you see fit.

Tip: You can use the Find and Replace feature of Microsoft Word to replace all instances of <<SITE>> in this document with your site abbreviation.



Navigation Links to Examples of Queries

Queries Applicable to Master Facility Table

- [Count Active Facilities](#)
- [Count Facilities by Facility Status](#)
- [Count Facilities by Facility Status and Patient Class](#)
- [Count Facilities by Patient Class](#)
- [Join MFT to Operational Crosswalk](#)

Queries Applicable to the Raw Table

- [Select First 50 Records by Date Range](#)
- [Select Records from a Single Day](#)
- [Select Records by Feed for a Given Period](#)
- [Retrieve Most Recent Message Date](#)

Queries Applicable to Processed Table

- [Select a Set Number of Records from a Specified Period](#)
- [Count the Number of Visits on a Specified Date](#)
- [Count the Number of Visits Each Day in a Specified Period](#)
- [Select a Set Number of Records for a Single Day](#)
- [Select a Set Number of Records Based on a Chief Complaint](#)
- [Count Records Containing a Specified Chief Complaint](#)
- [Count Records Containing a Specified Chief Complaint Based on a Specified Date Range, Stratified by Another Variable](#)
- [Retrieve the Last Visit Date](#)
- [Count Visits by Facility](#)
- [Count Visits by Date in Descending Order](#)
- [Join Processed and MFT Tables—Counts, Min/Max Dates, by Facility Name](#)
- [Use the New “ source” Columns](#)

Queries Applicable to Master Facility Table

Count Active Facilities

This query counts the number of active, primary facilities within a site and stratifies the count by Patient Class. (Note. The MFT Patient_Class_Code column will be renamed to C_MFT_Patient_Class in the next round of updates to emphasize that this is a *patient class inferred from facility type associated with the MFT entry* and not associated with the patient class reported in the visit message.)

```
select Patient_Class_Code as C_MFT_Patient_Class, count(*) as Facilities
from <<SITE>>_MFT
where facility_status='Active' and primary_facility='Y'
group by Patient_Class_Code
```

Count Facilities by Facility Status

This query generates counts of the number of facilities and stratifies those counts by Facility Status.

```
select Facility_Status as Status, count(*) as Facilities
from <<SITE>>_MFT
group by Facility_Status
```

Count Facilities by Facility Status and Patient Class

This query generates a summary table containing counts of facilities by Facility Status and Patient Class.

```
select Facility_Status as Status, Patient_Class_Code as C_MFT_Patient_Class , count(*) as
Facilities
from <<SITE>>_MFT
group by Facility_Status, Patient_Class_Code
```

Count Facilities by Patient Class

This query generates a count of active facilities by MFT Patient Class.

```
select Patient_Class_Code as C_MFT_Patient_Class, count(*) as Facilities
from <<SITE>>_MFT
where facility_status='Active'
group by Patient_Class_Code
```

Join MFT to Operational Crosswalk

Review history of various facility identifiers associated with the same facility and the current Output_FacilityID_UUID that the various identifiers map to in the Crosswalk.

```
SELECT
  b.Facility_Name,
  a.Input_FacilityID_UUID,
  a.Output_FacilityID_UUID,
  a.C_BioSense_Facility_ID
FROM <Site>_Operational_Crosswalk a
JOIN <Site>_MFT b
  on a.C_Biosense_Facility_ID = b.C_Biosense_Facility_ID
  order by b.Facility_Name
```

Queries Applicable to Raw Table

Select First 50 Records by Date Range

This query retrieves the first 50 records from a date range specified in your raw table. In this example, the range specified is March 1, 2016, through March 31, 2016. Please note that different operators and date/time precision can result in slight variations in the data returned.

```
select top(50) *
from <<SITE>>_ST_Raw
where Arrived_Date_Time >= '20160301' and Arrived_Date_Time <'20160401'
```

If Time is not specified in the selection criteria, Adminer will default to midnight. In this example, we successfully capture records with our intended date range of March 1, 2016, through March 31, 2016, because we selected records with an arrival date time of March 1, 2016, at midnight, up to April 1, 2016, at midnight. Had we specified Arrived_Date_Time <= '20160331', we would not capture all the records of interest:

```
where Arrived_Date_Time >= '20160301' and Arrived_Date_Time <= '20160331'
```

(This will not work because the end range captures records with date value through midnight on 03/31/2016 but not beyond midnight on 03/31/2016.)

Notes about Date Ranges:

Several columns are formatted as a date time (datetime2). Some even have companion columns formatted as a date. For example, C_Visit_Date_Time has a companion date portion C_Visit_Date. In such instances, consider using the date version of the column in your query selection. In our example that uses Arrived_Date_Time, there is no companion date column. Therefore, we need to be mindful of how we form the query selection criteria.

Using comparison operators—When using date time columns in queries that use comparison operators (e.g., >, >=, <, <=, etc.), set the high end of the date range to less than one day after your end date. In our example, the query will select all records with arrival date time of March 1, 2016, at midnight, up to but not including April 1, 2016, at midnight:

```
select top(50) *
from <<SITE>>_ST_Raw
where Arrived_Date_Time >= '20160301' and Arrived_Date_Time <'20160401'
```

Using “between”—Another way to write this query would include “between.” In this instance, we can use CAST to convert the date time column to a date to ensure we pull records in the requested date range:

```
select top(50) *
from <<SITE>>_ST_Raw
where cast(Arrived_Date_Time as date) between '20160301' and '20160331'
```

If you use the *between* operator, it is beneficial to cast the date values as dates. The underlying variables include a time definition. Therefore, by casting the date values with the *between* operator, the SQL query will capture all data from your date range. Without casting the dates, as shown in the example below, the records would be captured only through midnight on the last date of your date range.

```
where Arrived_Date_Time between '20160301' and '20160331'
```

(This will not work because the end range captures records with date value through midnight on 03/31/2016 but not beyond midnight on 03/31/2016.)

Select Records from a Single Day

This query generates visit counts for a single day. The example uses March 11, 2016.

```
select count(*) as count
from <<SITE>>_ST_Raw
where cast (Arrived_Date_Time as date) = '20160311'
```

The query would not have returned all records of interest had we not converted the date time to a date.

```
select count(*) as count
from <<SITE>>_ST_Raw
where (Arrived_Date_Time as date) = '20160311'
```

(This will not work because the query will only capture a specific date time of midnight on 03/11/2016.)

Select Records by Feed for a Given Period

This query generates record counts for each feed in your raw table and then sorts the feeds by count in descending order. In this example, we use March 1, 2016, through March 31, 2016.

```
select Feed_Name, count(*) as Recs
from <<SITE>>_ST_Raw
where Arrived_Date_Time >= '20160301' and Arrived_Date_Time < '20160401'
group by Feed_Name
order by Recs desc
```

Retrieve Most Recent Message Date

This query returns the date only for the most recent message in the raw table.

```
select max(cast(Arrived_Date_Time as date))
from <<SITE>>_ST_Raw
```

computed
03-22-2016

For a precise returned date with time stamp, use the following query:

```
select max(Arrived_Date_Time)
from <<SITE>>_ST_Raw
```

computed
2016-03-22 19:57:38

Queries Applicable to Processed Table

Select a Set Number of Records from a Specified Period

This query retrieves the first 50 records with a visit date for a specified period. In this example, the query pulls from a week's worth of data between March 1, 2016, and March 8, 2016.

```
select top (50)*  
from <<Site>>_ST_Processed  
where C_Visit_Date between '20160301' and '20160308'
```

(Note. This works because C_Visit_Date is being used and is a date, not a date time. That is, C_Visit_Date is the date portion of the C_Visit_Date_Time column. Had C_Visit_Date_Time been used, this would not have worked because the end range captures records with date value up to midnight on 03/08/2016. Refer to Notes about Date Ranges to learn more about queries using date time columns.)

Count the Number of Visits on a Specified Date

This query returns the counts of visits on a specific date. In this example, the query is retrieving the number of visits on March 11, 2016.

```
select count(*) as Recs  
from <<Site>>_ST_Processed  
where cast(C_Visit_Date_Time as date) = '20160311'
```

Count the Number of Visits Each Day in a Specified Period

This query returns the number of visits for each day during a specified period. In this example, the query counts visits from March 1, 2016, to March 8, 2016.

```
select C_Visit_Date, count(*) as Visits  
from <<Site>>_ST_Processed  
where C_Visit_Date between '20160301' and '20160308'  
group by C_Visit_Date
```

Select a Set Number of Records for a Single Day

This query retrieves the first 50 records from a single day. In this example, the query is retrieving records from March 1, 2016.

```
select top(50)*  
from <<Site>>_ST_Processed  
where cast (C_Visit_Date_Time as date) = '20160301'
```

Select a Set Number of Records Based on a Chief Complaint

This query returns the first 50 records from your processed table by using a specified calculated visit date range that contains “respiratory” in the Chief Complaint field.

```
select top(50)*
from <<Site>>_ST_Processed
where lower(C_Chief_Complaint) like '%respiratory%' and cast(C_Visit_Date_Time as date)
between '20160301' and '20160316'
```

Another way to write this query would be

```
select top(50)*
from <<Site>>_ST_Processed
where lower(C_Chief_Complaint) like '%respiratory%' and
C_Visit_Date_Time >= '20160301' and C_Visit_Date_Time < '20160317'
```

Alternatively, you could use the companion date-formatted column for calculated visit date, C_Visit_Date:

```
select top(50)*
from <<Site>>_ST_Processed
where lower(C_Chief_Complaint) like '%respiratory%' and C_Visit_Date between '20160301'
and '20160316'
```

Count Records Containing a Specified Chief Complaint

This query generates counts of records for which Chief Complaint contains a specified phrase for a specified period. This example uses Chief Complaints containing the phrase “respiratory” for the period March 1, 2016, through March 15, 2016.

```
select count(Chief_Complaint_Text) as Recs
from <<Site>>_ST_Processed
where lower(C_Chief_Complaint) like '%respiratory%' and cast(C_Visit_Date_Time as date)
between '20160301' and '20160315'
```

Another way to write this query would be

```
select count(Chief_Complaint_Text) as Recs
from <<Site>>_ST_Processed
where lower(C_Chief_Complaint) like '%respiratory%' and
C_Visit_Date_Time >= '20160301' and C_Visit_Date_Time < '20160316'
```

Count Records Containing a Specified Chief Complaint Based on a Specified Date Range, Stratified by Another Variable

This query provides the same results as the previous query and stratifies it by another variable. In this example, the query is stratifying by the Administrative_Sex variable.

```
select Administrative_Sex as Gender, count(Chief_Complaint_Text) as Recs
from <<Site>>_ST_Processed
where lower(C_Chief_Complaint) like '%respiratory%' and cast (C_Visit_Date_Time as date)
between '20160301'and '20160315'
group by Administrative_Sex
```

Another way to write this query would be

```
select Administrative_Sex as Gender, count(Chief_Complaint_Text) as Recs
from <<Site>>_ST_Processed
```

```

where lower(C_Chief_Complaint) like '%respiratory%' and
C_Visit_Date_Time >= '20160301' and C_Visit_Date_Time < '20160316'
group by Administrative_Sex

```

Retrieve the Last Visit Date

This query retrieves the last visit date in the processed table.

```

select max(C_Visit_Date)
from <<Site>>_ST_Processed

```

Count Visits by Facility

This query generates counts of visits by Facility.

```

select C_BioSense_Facility_ID, count(*) as Visits
from <<Site>>_ST_Processed
group by C_BioSense_Facility_ID
order by Visits desc

```

Count Visits by Date in Descending Order

This query generates counts of visits by day, grouped by Day and ordered by Date, descending.

```

select C_Visit_Date as Visits_Date, count(*) as Visits
from <<Site>>_ST_Processed
group by C_Visit_Date
order by C_Visit_Date desc

```

Join Processed and MFT Tables—Counts, Min/Max Dates, by Facility Name

This query creates a facility-level summary table depicting minimum and maximum visit dates, number of patients, and total visits (based on unique visit ID and total records for each facility and for a specified date range). In this example, visit dates from March 1, 2016, through March 31, 2016, are used. This query also pulls the facility name from Master Facility Table by using a left outer join.

```

Select
a.Site_ID,
a.C_BioSense_Facility_ID,
b.Facility_Name,
min(cast(a.Message_Date_Time as date)) as min_Msg_Date,
max(cast(a.Message_Date_Time as date)) as max_Msg_Date,
min(a.C_Visit_Date) as min_Visit_Date,
max(a.C_Visit_Date) as max_Visit_Date,
Count(distinct a.C_Unique_Patient_ID) as Patients,
Count(Distinct a.C_BioSense_ID) as Visits,
Count(*) as Recs
From <<Site>>_ST_Processed a
left outer join <<SITE>>_MFT b
on a.C_BioSense_Facility_ID=b.C_BioSense_Facility_ID
where a.C_Visit_Date_Time between '20160301' and '20160331' and b.Primary_Facility='Y'
group by a.Site_ID, a.C_BioSense_Facility_ID, b.Facility_Name

```


Use the New “_source” Columns

This query reports the source of the calculated patient class (C_Patient_Class), leveraging the companion column C_Patient_Class_Source.

```
SELECT count(*) as cnt,
  c_patient_class,
  c_patient_class_source
from <<Site>>_ST_Processed
group by c_patient_class, c_patient_class_source
```

This query includes the various fields that contribute to, or “seed,” the calculated patient class.

```
SELECT count(*) as cnt,
  C_Patient_Class,
  Patient_Class_Code,
  C_FacType_Patient_Class,
  C_MFT_Patient_Class,
  C_Patient_Class_Source
from <<Site>>_ST_Processed
group by
  C_Patient_Class,
  Patient_Class_Code,
  C_FacType_Patient_Class,
  C_MFT_Patient_Class,
  C_Patient_Class_Source
Order by cnt desc
```

This query reports the source of the calculated unique patient identifier (C_Unique_Patient_ID), leveraging the companion column C_Unique_Patient_ID_Source.

```
SELECT count(*) as cnt,
  c_unique_patient_id_source
from <<Site>>_ST_Processed
group by c_unique_patient_id_source
```