



## What Is a Query Activity

A query is an [activity](#) to retrieve [data extension](#) or data view information that matches your criteria and include that information in a data extension. You use SQL to create the query you use in the query activity.

When you create a query activity, you provide name, external key, and description to identify and describe the activity within the application interface and for API calls.

The default timeout time for your SQL statements is 30 minutes. In certain circumstances, you may be able to request a different timeout time, up to 120 minutes. To request approval of an override timeout time, contact your account manager or customer service.

## Query

When you create a query activity, you write the SQL statement that defines the query. You can write the query against any existing data extension or the data views provided in the system.

A Query Activity SQL statement is an isolated statement that cannot take input parameters from other activities or other processes.

If your account is a child account in an Enterprise, you can query data extensions in your parent account. Prefix the data extension name in the query with **ENT**. For example:

```
Select email from ent.customers
```

In this example, **email** is the column name and **customers** is the data extension residing in the parent account.

You can query the database for the following information:

- [Data View: Bounce](#)
- [Data View: Click](#)
- [Data View: Complaint](#)
- [Data View: Coupon](#)
- [Data View: EnterpriseAttribute](#)
- [Data View: Social Network Impressions](#)
- [Data View: Social Network Tracking](#)
- [Data View: FTAF](#)
- [Data View: Job](#)
- [Data View: ListSubscribers](#)
- [Data View: Open](#)
- [Data View: PublicationSubscriber](#)
- [Data View: Sent](#)
- [Data View: Subscribers](#)
- [Data View: SubscriberSMS](#)
- [Data View: SurveyResponse](#)
- [Data View: Unsubscribe](#)

You can retrieve up to six months' of data from all of the tables. See the table layout for each data view you can query.

The query activity interface provides a tool to check the syntax of your SQL.

If your query activity includes a SubscriberKey column, you can set that data type to Text unless the SubscriberKey is an email address. In that case, you can set the data type to Email.

## Target

When you create a query activity you identify the data extension to contain the results of the query. You must create the data extension before you create the activity. You also indicate whether the system overwrites all of the rows in the data extension, updates the existing rows, or appends to the rows of the data extension.

## How to Create a Query Activity

Before you begin this procedure, you must create the [data extension](#) to contain the results of the query. After you complete this procedure, you can [start the activity](#) or include the [activity](#) in a [program](#).

Use the following steps to create a query activity:

1. Click the **Interactions** tab on the navigation bar.
2. Click **Activities**.
3. Click **Query**.

The **Queries** workspace appears.

4. Click **Create** from the toolbar.
5. Complete the information in the **Properties** section:

**Name** - The name of the activity. You use this name to identify the activity in the application. Subscribers cannot see the name.

**Key** - A value you choose that uniquely identifies the activity. You use this value to identify the activity when using the API.

**Description** - The description of the activity. You use this description to further help identify the activity within the application. Subscribers cannot see the description.

**Query** - The SQL that makes up the query. You can click the Check Syntax button to check your SQL.

6. Complete the information in the Target section:

**Select data extension to populate** - The data extension to contain the result of the query.

**Update Type** - Determines how the system updates the data extension with new data. Value values include:

. Overwrite - The system deletes the existing records in the data extension and adds the results of the query.

. Update - The system updates the existing records in the data extension with information that results from the query and appends non-matching records.

. Append - The system adds the results of the query to the data extension after the existing records.

7. Click **Save**.

## SQL Support

The SQL support for the Query Activity is based on SQL Server 2005 capabilities.

- Only SELECT statements to data extension or data views in an account or in the parent account
  - ? Nested Queries
  - ? UNION
  - ? JOIN
  - ? GROUP BY
- Conditional Statements
  - ? IF Constructs
  - ? CASE Statements
- Functions
  - ? Most functions (i.e. MIN, MAX, etc) are supported
  - ? CAST and CONVERT
- Unsupported elements
  - ? Variables
  - ? Cursors
  - ? User Defined Functions
  - ? Transaction and Locking
  - ? GOTO
  - ? PRINT
  - ? Any sp\_\* stored procedure
  - ? EXEC
  - ? Temporary Tables and Common Table Expressions
  - ? TEXT and IMAGE Functions

## Known Issues

- Data Types
  - ? Decimal: If a Data Extension contains a decimal field and that field is used in a query, the output will not be a decimal output
- Data Views and Enterprise 2.0 Business Units
  - ? Only data extensions are accessible for queries written at the Enterprise 2.0 business unit

## Querying Data Extensions

- In an enterprise (standard or 2.0) environments, you can decorate the data extension name with a "ENT." prefix to get access to data extension that reside in the enterprise parent account.
- A shared data extension in an Enterprise 2.0, should be accessed using the "ENT." prefix.
- Use the data extension Name in SQL queries not the External Key.

## Querying Subscriber Data

Using the `_Subscriber` data view

```
SELECT *
from _subscribers
```

**Using the `_EnterpriseAttribute` data view**

```
SELECT *
from _EnterpriseAttribute
```

This query can be used in Enterprise 2.0 edition accounts only. Core, Advanced, or Enterprise do not support querying subscriber attributes.

**Combining the `_EnterpriseAttribute` data view and the `_Subscriber` data view**

```
SELECT
b.[first name],
b.[last name],
b.[age],
a.[Status] as 'Subscription Status',
a.[EmailAddress] as 'Email'
FROM
_subscribers a
INNER JOIN
_EnterpriseAttribute b
ON
a.subscriberID = b._subscriberID
```

This example assumes the `[first name]`, `[last name]`, and `[age]` attributes exist in the account in which the query is run. You could choose to query for any attribute that you created in your own account.

**Performance Considerations**

Queries have a 30 minute timeout. If a query is timing out, it may be possible to make the query perform faster with indexes applied. Contact customer support if you believe we would like to have your query validated by our internal teams for performance.

**Sample Queries**

Sample queries that demonstrate the possible functionality of the query activity.

**Samples****Get a Send List Based on Aggregate Row Count**

Use Case: Generating a Sendable list from a detailed data set

```
select visitorid from [cart abandonment segment] group by visitorid
having count(visitorid) > 1
```

**Random Sample of 33 Percent Exclusion**

Use Case: A/B split testing and general segmentation

```
Select top 33 percent
offerid, score, name, description, url
from Offers
inner join Offer_Ext1 a on a.offerid <> offers.offerid
order by newid()
```

**Random Sample of 33 Percent**

Use Case: A/B split testing and general segmentation

Select top 33 percent

offerid, score, name, description, url

from Offers  
order by newid()**Random Sample of 100**

Use Case: A/B split testing and general segmentation

Select top 100

offerid, score, name, description, url

from Offers  
order by newid()**Unique Values for a Data Set**

Use Case: Testing

```

Select distinct
'109406145' as linkage_id,
'109406145' as ticket_id,
'28480048' as family_id,
'73334' as merchant_id,
'25.32' as Original_Transaction_Amt,
transaction_type_cd,
Max(TRANSACTION_DATE) as Transaction_Date,
TRANSACTION_STATUS,
MERCHANT_TYPE,
'0' as Group_Points,
'.00' as Group_Comp,
'.00' as Group_Credit,
'.00' as Group_Dues,
'0' as Group_Miles,
'.00' as Group_BonusComp,
'102' as Group_BonusMiles,
'0' as Group_BonusPoints,
'.00' as Group_BonusCrdt,
'.00' as Total_Comp_Earned,
'.00' as Total_Credit_Earned,
'10' as Total_Miles_Earned,
'103' as Total_Points_Earned,
MERCHANT_CATEGORY,
RA_SEQUENCE,
'FFAK' as fam_campaign_cd

FROM Subscriber_Activity

```

```
GROUP BY transaction_type_cd, TRANSACTION_STATUS, MERCHANT_TYPE,
RA_SEQUENCE, MERCHANT_CATEGORY
```

### Geo-Targeting

Prerequisites: Zip Code Data Extension. See the attached [zipcode.csv](#) file.

Todo: Join Zip Codes to a Subscriber Data Extension

Use Case: Find all cities/zips (and therefore subscribers) within 15 kilometers of a zip code.

```
-- 6378.137 earth circumference
```

```
SELECT
  city,
  zip,
  ROUND(6378.137 * ACOS(
    CASE
      WHEN (SIN(RADIANS((SELECT latitude FROM [ZipCode] where zip =
46254))) * SIN(RADIANS(geo.Latitude))) + (COS(RADIANS((SELECT latitude
FROM [ZipCode] where zip = 46254))) * COS(RADIANS(geo.Latitude)) *
COS(RADIANS(geo.Longitude) - RADIANS((SELECT longitude FROM [ZipCode]
where zip = 46254)))) > 1 THEN 1
      WHEN (SIN(RADIANS((SELECT latitude FROM [ZipCode] where zip =
46254))) * SIN(RADIANS(geo.Latitude))) + (COS(RADIANS((SELECT latitude
FROM [ZipCode] where zip = 46254))) * COS(RADIANS(geo.Latitude)) *
COS(RADIANS(geo.Longitude) - RADIANS((SELECT longitude FROM [ZipCode]
where zip = 46254)))) < -1 THEN -1
      ELSE (SIN(RADIANS((SELECT latitude FROM [ZipCode] where zip =
46254))) * SIN(RADIANS(geo.Latitude))) + (COS(RADIANS((SELECT latitude
FROM [ZipCode] where zip = 46254))) * COS(RADIANS(geo.Latitude)) *
COS(RADIANS(geo.Longitude) - RADIANS((SELECT longitude FROM [ZipCode]
where zip = 46254))))
    END),0) AS Distance
FROM
  [ZipCode] AS geo
WHERE
  ROUND(6378.137 * ACOS(
    CASE
      WHEN (SIN(RADIANS((SELECT latitude FROM [ZipCode] where zip =
46254))) * SIN(RADIANS(geo.Latitude))) + (COS(RADIANS((SELECT latitude
FROM [ZipCode] where zip = 46254))) * COS(RADIANS(geo.Latitude)) *
COS(RADIANS(geo.Longitude) - RADIANS((SELECT longitude FROM [ZipCode]
where zip = 46254)))) > 1 THEN 1
      WHEN (SIN(RADIANS((SELECT latitude FROM [ZipCode] where zip =
46254))) * SIN(RADIANS(geo.Latitude))) + (COS(RADIANS((SELECT latitude
FROM [ZipCode] where zip = 46254))) * COS(RADIANS(geo.Latitude)) *
COS(RADIANS(geo.Longitude) - RADIANS((SELECT longitude FROM [ZipCode]
where zip = 46254)))) < -1 THEN -1
      ELSE (SIN(RADIANS((SELECT latitude FROM [ZipCode] where zip =
46254))) * SIN(RADIANS(geo.Latitude))) + (COS(RADIANS((SELECT latitude
FROM [ZipCode] where zip = 46254))) * COS(RADIANS(geo.Latitude)) *

```

```
COS(RADIANS(geo.Longitude) - RADIANS((SELECT longitude FROM [ZipCode]
where zip = 46254))))
    END),0) <= 15
```

**Pulling Tracking Information for a Triggered Send**

Use Case: Viewing available tracking information regarding a triggered send

```
SELECT *
FROM _Open
WHERE TriggeredSendCustomerKey = 'External Key of Triggered Send'
```

This page was last updated by [Josh Cloud](#) on Tue, 10 Apr 2012 13:42:55 GMT.

If you're having an application issue, please [contact Global Support](#). To send Josh direct feedback, fill out the form below:

Was This Page Helpful?	<input type="radio"/>  <input type="radio"/> 
Suggestions or Comments:	<input type="text"/>
Name (optional):	<input type="text"/>
Email Address (optional):	<input type="text"/>
Enter 19035 backwards:	<input type="text"/>
	<input type="button" value="Send Josh Feedback"/>