SQL Query Statements

Many useful Microsoft SQL Server query statements can be used when creating Microsoft Systems Management Server (SMS) 2003 reports, and they are briefly described in this section. To follow this discussion, you should have a basic level of SQL query statement knowledge and the ability to write queries such as the following:

```
SELECT Name, Comment, CollectionID
FROM v_Collection
WHERE Name LIKE 'All Windows%'
ORDER BY Name
```

For information about how to write basic queries, see Query Fundamentals in the Microsoft SQL Server 2000 Books Online.

Related Links

- Appendix D: SQL Server Reference
- Microsoft SQL Server TechCenter
- SQL Server 2000 Books Online
- SQL Server 2005 Books Online

Aggregate Functions

Aggregate functions (such as SUM, AVG, COUNT, COUNT(*), MAX, and MIN) generate summary values in query result sets. An aggregate function (with the exception of COUNT(*)) processes all the selected values in a single column to produce a single result value.

Aggregate functions can be applied to all rows in a view, to a subset of the view specified by a WHERE clause, or to one or more groups of rows in the view. When an aggregate function is applied, a single value is generated from each set of rows.

Important

Be aware that NULL values are not included in aggregate results. For example, if you have 100 records and 8 of them have a NULL column value for the property that you are counting, the count will return only 92 results.

An example of using the COUNT(*) aggregate function is displayed in the following query (from the **Count clients for each site** SMS 2003 built-in report) and example result set.

```
SELECT v_Site.SiteCode, v_Site.SiteName, v_Site.ReportingSiteCode,
Count(SMS_Installed_Sites0) AS 'Count'
FROM v_Site, v_RA_System_SMSInstalledSites InsSite
WHERE v_Site.SiteCode = InsSite.SMS_Installed_Sites0
GROUP BY SiteCode, SiteName, ReportingSiteCode
ORDER BY SiteCode
```

SiteCode	SiteName	ReportingSiteCode	Count
ABC	ABC Site		928
123	123 Site	ABC	1010

Top of page

Date and Time Functions

Many built-in reports use the Date and Time functions. The most common functions used are the GETDATE, DATEADD, DATEDIFF, and DATEPART.

GETDATE ()

The GETDATE function produces the current date and time in SQL Server internal format for datetime values. GETDATE takes the *NULL* parameter ().

The following example results in the current system date and time:

```
SELECT GETDATE()
```

(no column name)
2005-05-29 10:10:03.001

DATEADD (datepart, number, date)

The DATEADD function returns a new datetime value based on adding an interval to the specified date.

Datepart is the parameter that specifies on which part of the date to return a new value (for example, year, month, day, hour, minute, and so forth), *number* is the value used to increment datepart, and *date* is the starting date.

The following example results in a date that is two days from May 29, 2005:

```
SELECT DATEADD([day], 2, '2005-05-29 10:10:03.001')
```

(no column name)
2005-05-31 10:10:03.001

DATEDIFF (datepart , startdate , enddate)

The DATEDIFF function returns the number of date and time boundaries crossed between two specified dates.

Datepart is the parameter that specifies on which part of the date to return a new value (for example, year, month, day, hour, minute, and so forth), *startdate* is the starting date, *enddate* is the ending date.

The following example results in the number of minutes between the first and second dates:

```
SELECT DATEDIFF (minute, '2005-05-29 10:10:03.001',
'2005-06-12 09:28:11.111')
```

(no column name)
20118

DATEPART (datepart , date)

The DATEPART function returns an integer representing the specified datepart of the specified date.

Datepart is the parameter that specifies on which part of the date to return, and *date* is the specified date.

The following example results in the month in the specified date:

```
SELECT DATEPART (month, '2005-05-29 10:10:03.001')
```

(no column name)
5

Combining Date and Time Functions

It is typical to use a combination of the Date and Time functions in SMS 2003 reports.

The following example results in the current date and time (2005-05-29 10:10:03.001 in this example) minus 100 days:

```
SELECT DATEADD([day], - 100, GETDATE())
```

(no column name)
2005-02-18 10:10:03.001

SMS Example Query Using Date and Time Functions

The following SMS query results in the total count of status messages for a one-day period. In this query, the COUNT, GETDATE, and DATEADD functions are used as well as the BETWEEN logical operator and the GROUP BY and ORDER BY clauses.

```
SELECT SiteCode, MessageID, COUNT(MessageID) AS [count],
GETDATE() AS [End Date]
FROM vStatusMessages
WHERE ([Time] BETWEEN DATEADD([day], -1, GETDATE()) AND GETDATE())
AND (MessageID BETWEEN '0' AND '10000')
GROUP BY SiteCode, MessageID
ORDER BY SiteCode, MessageID
```

Site Code	MessageID	Count	End Date
ABC	500	190	2005-05-29 10:10:03.001
ABC	501	130	2005-05-29 10:10:03.001
ABC	502	190	2005-05-29 10:10:03.001
ABC	1105	85	2005-05-29 10:10:03.001
ABC	1106	5	2005-05-29 10:10:03.001
...	...	...	...

Top of page

JOINS

To create effective reports in SMS, you need to understand how to join different views to get the expected data. There are three types of joins: inner, outer, and cross. In addition, there are three types of outer joins: left, right, and full. The self join utilizes any of the above joins, but joins records from the same view.

Inner Joins

In an inner join, records from two views are combined and added to a query's results only if the values of the joined fields meet certain specified criteria. If you use an inner join by using the ResourceID to join the v_R_System and v_GS_WORKSTATION_STATUS views, the result would be a list of all systems and their last hardware scan date.

```
SELECT v_R_System.Netbios_Name0 AS MachineName,
v_GS_WORKSTATION_STATUS.LastHWScan AS [Last HW Scan]
FROM v_R_System INNER JOIN v_GS_WORKSTATION_STATUS
ON v_R_System.ResourceID = v_GS_WORKSTATION_STATUS.ResourceID
```

Machine Name	Last HW Scan

Machine Name	Last HW Scan
Client1	2005-05-29 10:10:03.001
Client3	2005-06-12 09:28:11.110

Outer Joins

An outer join returns all rows from the joined views whether or not there's a matching row between them. The ON clause supplements the data rather than filtering it. The three types of outer joins (left, right, and full) indicate the main data's source. Outer joins can be particularly helpful when you have NULL values in a view.

Left Outer Joins

When you use a left outer join to combine two views, all the rows the left view are included in the results. In the following query, the v_R_System and v_GS_WORKSTATION_STATUS views are joined using the left outer join. The v_R_System view is the first view listed in the query making it the left view. The result will include a list of all systems and their last hardware scan date. Unlike the inner join, systems that have not been scanned for hardware will still be listed with a NULL value (as seen in the result set).

```
SELECT v_R_System.Netbios_Name0 AS MachineName,
v_GS_WORKSTATION_STATUS.LastHWScan AS [Last HW Scan]
FROM v_R_System LEFT OUTER JOIN v_GS_WORKSTATION_STATUS
ON v_R_System.ResourceID = v_GS_WORKSTATION_STATUS.ResourceID
```

Machine Name	Last HW Scan
Client1	2005-05-29 10:10:03.001
Client2	NULL
Client3	2005-06-12 09:28:11.110

Right Outer Joins

A right outer join is conceptually the same as a left outer join except that all the rows from the right view are included in the results.

Full Outer Join

A full outer join retrieves all the rows from both joined views. It returns all of the paired rows where the join condition is true, plus the unpaired rows from each view concatenated with NULL rows from the other view. You usually won't want to use this type of outer join.

Cross Join

A cross join returns the product of two views, not the sum. Each row in the left view is matched up with each row in the right view. It's the set of all possible row combinations, without any filtering. However, if you add a WHERE clause, a cross join functions as an inner join—it uses the condition to filter all possible row combinations down to the ones you want.

Self Join

A self join uses any of the above join types, but where a view is joined to itself. In database diagrams, a self join is called a reflexive relationship.

Top of page

NOT IN Keyword Phrase

Subqueries with the keyword phrase NOT IN are very useful to find information about a set of data that doesn't meet a certain criteria. In the following example, the query returns the NetBIOS name of all computers that do NOT have notepad.exe installed. You must first create a query that can detect all computers that have the selected file installed as follows:

```
SELECT DISTINCT v_R_System.Netbios_Name0
FROM v_R_System INNER JOIN v_GS_SoftwareFile
ON (v_GS_SoftwareFile.ResourceID = v_R_System.ResourceId)
WHERE v_GS_SoftwareFile.FileName = 'Notepad.exe'
```

After confirming that the first query displays all of the computers that have Notepad.exe installed, the following subquery statement will use the NOT IN keyword phrase to find all computer names which do NOT have the Notepad.exe file installed:

```
SELECT DISTINCT Netbios_Name0
FROM v_R_System
WHERE Netbios_Name0 NOT IN
(SELECT DISTINCT v_R_System.Netbios_Name0
FROM v_R_System INNER JOIN v_GS_SoftwareFile
ON (v_GS_SoftwareFile.ResourceID = v_R_System.ResourceId)
WHERE v_GS_SoftwareFile.FileName = 'Notepad.exe')
ORDER by Netbios_Name0
```

Top of page

Joining to External Data

It is possible to retrieve data from two different databases in one query. The following example query will join both the SMS Client Health and SMS Site databases to get information about the state of the client and the status of a selected SMS Advertisement:

```
SELECT v_ClientAdvertisementStatus.AdvertisementID,
SMS_ClientHealth.dbo.ClientHealthResults.Name,
v_ClientAdvertisementStatus.LastStateName,
SMS_ClientHealth.dbo.ClientHealthResults.Classification,
v_ClientAdvertisementStatus.LastStatusTime,
v_ClientAdvertisementStatus.LastStatusMessageID
FROM v_ClientAdvertisementStatus INNER JOIN
SMS_ClientHealth.dbo.ClientHealthResults
ON v_ClientAdvertisementStatus.ResourceID =
SMS_ClientHealth.dbo.ClientHealthResults.ResourceID
WHERE (v_ClientAdvertisementStatus.AdvertisementID = 'ABC20001')
```

Note

This query uses **SMS_ClientHealth** as the SMS Client Health database name and **ABC20001** as the AdvertisementID and will need to be modified with values relevant to your site for it to run successfully.

1 .write a sql query to write the first week of the month

select DATENAME(DW,DATEADD(dd,-DATEPART(dd,GETDATE())+1,GETDATE())) AS FirstDay;

2. how to find 6th highest salary from employee table

select TOP 1 salary FROM (SELECT DISTINCT TOP 6 salary FROM employee ORDER BY salary DESC) a ORDER BY salary