

SQL and XQuery tutorial for IBM DB2, Part 4: Data analysis

Using advanced queries to analyze data

Skill Level: Introductory

[Pat Moffatt \(pmoffatt@ca.ibm.com\)](mailto:pmoffatt@ca.ibm.com)

Information Management Program Manager, IBM Academic Initiative
IBM

[Bruce Creighton \(bcreight@ca.ibm.com\)](mailto:bcreight@ca.ibm.com)

Skills Segment Planner
IBM

[Jessica Cao \(jcao@ca.ibm.com\)](mailto:jcao@ca.ibm.com)

Training Tools Developer
IBM

17 Aug 2006

This tutorial describes how to write queries that require basic data analysis. Many of the queries contain sequential calculations, or calculations that operate on an ordered set of rows--queries frequently encountered during business analysis. On-Line Analytical Processing (OLAP) functions provide the ability to return ranking, row numbering and existing column function information as a scalar value in a query result. This tutorial is Part 4 of the [SQL & XQuery tutorial for IBM® DB2®](#) series.

Section 1. Before you start

About this series

This tutorial series teaches basic to advanced SQL and basic XQuery topics and shows how to express commonly asked business questions as database queries by using SQL queries or XQueries. Developers and database administrators can use this tutorial to enhance their database query skills. Academic Initiative members can use this tutorial series as a part of their database curriculum.

All the examples in this document are based on **Aroma**, a sample database that contains sales data for coffee and tea products sold in stores across the United States. Each example consists of three parts:

- A business question, expressed in everyday language
- One or more example queries, expressed in SQL or XQuery
- A table of results returned from the database

This guide is designed to allow participants to learn the SQL language and XQuery. As with any learning, it is important to supplement it with hands-on exercises. This is facilitated by the table definitions and data.

For students using this as part of an academic class, obtain from your instructor the instructions to connect to the Aroma database and learn about any differences between the guide and your local set up.

This tutorial was written for DB2 Express-C 9 for UNIX®, Linux® and Windows® (formerly known as Viper).

About this tutorial

This tutorial describes how to write queries that require some kind of data analysis. Many of the queries contain sequential calculations, or calculations that operate on an ordered set of rows, queries frequently encountered during business analysis. For example:

- What is the cumulative total (or running sum) by month?
- What is the moving average by week?
- How do monthly sales figures rank with one another?
- What is the ratio of current month sales to annual sales?

IBM DB2 provides an efficient way to answer these kinds of questions using standard SQL OLAP functions that are included as part of DB2 9. On-Line Analytical Processing (OLAP) functions provide the ability to return ranking, row numbering, and existing column function information as a scalar value in a query result. An OLAP function can be included in expressions in a select-list or the ORDER BY clause of a select-statement.

This tutorial contains a series of examples with the business query and associated syntax presented in each case. Before running these queries, read the full descriptions of the OLAP functions in the [SQL Reference Guide](#).

This tutorial also shows how to use scalar functions to calculate and extract date information, such as day of week and month name, from DATE columns.

Many of the queries in this chapter rely on *aggregated* sales totals. Because the

Sales table stores daily totals, your database design might include aggregate tables to answer these queries.

Connecting to a database

You need to connect to a database before you can use SQL statements to query or manipulate data. The `CONNECT` statement associates a database connection with a user name.

Find out from your instructor the database name that you will need to be connected to. For this series, the database name is `aromadb`.

To connect to the `aromadb` database, type the following command in the DB2 command line processor:

```
CONNECT TO aromadb USER userid USING password
```

Replace the user ID and password with the user ID and password that you received from your instructor. If no user ID and password are required, simply use the following command:

```
CONNECT TO aromadb
```

The following message tells you that you have made a successful connection:

```
Database Connection Information
Database server      = DB2/NT 9.0.0
SQL authorization ID = USERID
Local database alias = AROMADB
```

Once you are connected, you can start using the database.

Section 2. Cumulative totals

Question

What were the daily sales figures for Aroma Roma coffee during January, 2006?
What were the cumulative subtotals for dollars and quantities during the month?

OLAP query

```

SELECT date, SUM(dollars) AS total_dollars,
SUM(SUM(dollars)) OVER(ORDER BY date ROWS UNBOUNDED PRECEDING) AS run_dollars,
SUM(quantity) AS total_qty,
SUM(SUM(quantity)) OVER(ORDER BY date ROWS UNBOUNDED PRECEDING) AS run_qty
FROM aroma.period a, aroma.sales b, aroma.product c
WHERE a.perkey = b.perkey
AND c.prodkey = b.prodkey
AND c.classkey = b.classkey
AND year = 2006
AND month = 'JAN'
AND prod_name = 'Aroma Roma'
GROUP BY date
ORDER BY date;

```

Result

Date	Total_Dollars	Run_Dollars	Total_Qty	Run_Qty
2006-01-02	855.50	855.50	118	118
2006-01-03	536.50	1392.00	74	192
2006-01-04	181.25	1573.25	25	217
2006-01-05	362.50	1935.75	50	267
2006-01-06	667.00	2602.75	92	359
2006-01-07	659.75	3262.50	91	450
2006-01-08	309.50	3572.00	54	504
2006-01-09	195.75	3767.75	27	531
2006-01-10	420.50	4188.25	58	589
2006-01-11	547.50	4735.75	78	667
2006-01-12	536.50	5272.25	74	741
2006-01-13	638.00	5910.25	88	829
2006-01-14	1057.50	6967.75	150	979
2006-01-15	884.50	7852.25	122	1101
2006-01-16	761.25	8613.50	105	1206
2006-01-17	455.50	9069.00	66	1272
2006-01-18	768.50	9837.50	106	1378
2006-01-19	746.75	10584.25	103	1481
2006-01-20	261.00	10845.25	36	1517
2006-01-21	630.75	11476.00	87	1604
2006-01-22	813.75	12289.75	115	1719
...				

OLAP SUM function

The presence of the OVER() clause differentiates a simple set function (SUM, MIN,

MAX, COUNT, AVG) from an OLAP aggregation function.

The OLAP SUM function produces running totals when the window frame specifies:

```
ROWS UNBOUNDED PRECEDING
```

This instruction tells the system to perform the OLAP function, in this case a SUM function, over all of the preceding rows in the result set. You can also specify a subset of rows using other limitations such as GROUP-BETWEEN. For detailed information about window frames, refer to the [SQL Reference Guide](#).

The OLAP ORDER BY clause is critical. This specification ensures that the input rows for the OLAP SUM function are sorted correctly (by **Date**, in this case). If you fail to use this instruction, the input rows may be out of logical order and your results for a running total will be meaningless. The final ORDER BY clause in the query affects only to how the result set will be displayed; it is separate and distinct from the ORDER BY clause for the OLAP function.

OLAPROW_NUMBER function

You can even use OLAP functions for simple tasks such as providing line numbers in a result set, as in:

```
SELECT ROW_NUMBER() OVER() AS row_num, order_no, price
FROM aroma.orders;
```

ROW_NUM	ORDER_NO	PRICE
1	3600	1200.46
2	3601	1535.94
3	3602	780.00
...		

Section 3. Resetting cumulative totals

Question

What were the cumulative Aroma Roma sales figures during each week of January, 2006?

OLAP query

```

SELECT date, SUM(dollars) AS total_dollars,
       SUM(SUM(dollars)) OVER(PARTITION BY week ORDER BY date
       ROWS UNBOUNDED PRECEDING) AS run_dollars,
       SUM(quantity) AS total_qty,
       SUM(SUM(quantity)) OVER(PARTITION BY week ORDER BY date
       ROWS UNBOUNDED PRECEDING) AS run_qty
FROM aroma.period a, aroma.sales b, aroma.product c
WHERE  a.perkey = b.perkey
       AND c.prodkey = b.prodkey
       AND c.classkey = b.classkey
       AND year = 2006
       AND month = 'JAN'
       AND prod name = 'Aroma Roma'
GROUP BY week, date
ORDER BY week, date;
    
```

Result

Date	Total_Dollars	Run_Dollars	Total_Qty	Run_Qty
2006-01-02	855.50	855.50	118	118
2006-01-03	536.50	1392.00	74	192
2006-01-04	181.25	1573.25	25	217
2006-01-05	362.50	1935.75	50	267
2006-01-06	667.00	2602.75	92	359
2006-01-07	659.75	3262.50	91	450
2006-01-08	309.50	3572.00	54	504
2006-01-09	195.75	195.75	27	27
2006-01-10	420.50	616.25	58	85
2006-01-11	547.50	1163.75	78	163
2006-01-12	536.50	1700.25	74	237
2006-01-13	638.00	2338.25	88	325
2006-01-14	1057.50	3395.75	150	475
2006-01-15	884.50	4280.25	122	597
2006-01-16	761.25	761.25	105	105
2006-01-17	455.50	1216.75	66	171
2006-01-18	768.50	1985.25	106	277
2006-01-19	746.75	2732.00	103	380
2006-01-20	261.00	2993.00	36	416
2006-01-21	630.75	3623.75	87	503
2006-01-22	813.75	4437.50	115	618
...				

OLAP window partitions

The OLAP PARTITION BY clause inside the OVER() clause provides a means of resetting calculations when values in the partitioned columns change. You can partition OLAP calculations by one or more columns.

In this query you did not include WEEK as a column to be displayed even though you used it in the PARTITION BY clause. The system derives the week number from the DATE column. Your query results might be easier to interpret, however, if you do include the WEEK value in your select list plus format the result table the same as other result.

Below is the query updated to include the week column in the select statement and the resulting output.

```
SELECT date, SUM(dollars) AS total_dollars,
           SUM(SUM(dollars)) OVER (PARTITION BY week ORDER BY date
ROWS UNBOUNDED PRECEDING) AS run_dollars,
           SUM(quantity) AS total_qty,
           SUM(SUM(quantity)) OVER (PARTITION BY week ORDER BY date
ROWS UNBOUNDED PRECEDING) AS run_qty, week
FROM aroma.period a, aroma.sales b, aroma.product c
WHERE  a.perkey = b.perkey
      AND c.prodkey = b.prodkey
      AND c.classkey = b.classkey
      AND year = 2006
      AND month = 'JAN'
      AND prod_name = 'Aroma Roma'
GROUP BY week, date
ORDER BY week, date
```

Result

DATE	TOTAL_DOLLA	RUN_DOLLAR\$	TOTAL_QTY	RUN_QTY	WEEK
2006-01-02	855.50	855.50	118	118	2
2006-01-03	536.50	1392.00	74	192	2
2006-01-04	181.25	1573.25	25	217	2
2006-01-05	362.50	1935.75	50	267	2
2006-01-06	667.00	2602.75	92	359	2
2006-01-07	659.75	3262.50	91	450	2
2006-01-08	309.50	3572.00	54	504	2
2006-01-09	195.75	195.75	27	27	3
2006-01-10	420.50	616.25	58	85	3
2006-01-11	547.50	1163.75	78	163	3
2006-01-12	536.50	1700.25	74	237	3
2006-01-13	638.00	2338.25	88	325	3
2006-01-14	1057.50	3395.75	150	475	3

2006-01-15	884.50	4280.25	122	597	3
2006-01-16	761.25	761.25	105	105	4
2006-01-17	455.50	1216.75	66	171	4
2006-01-18	768.50	1985.25	106	277	4
2006-01-19	746.75	2732.00	103	380	4
2006-01-20	261.00	2993.00	36	416	4
2006-01-21	630.75	3623.75	87	503	4
2006-01-22	813.75	4437.50	115	618	4
...					

Section 4. Using the arithmetic operators

Question

What was the average price per sale of each product during 2004? Calculate the average as the total sales dollars divided by the total sales quantity.

Example query

```
SELECT prod_name, SUM(dollars) AS total_sales, SUM(quantity) AS total_qty,
       DEC(sum(dollars)/sum(quantity), 7, 2) AS price
FROM aroma.product a, aroma.sales b, aroma.period c
WHERE  a.prodkey = b.prodkey
       AND a.classkey = b.classkey
       AND c.perkey = b.perkey
       AND year = 2004
GROUP BY prod_name
ORDER BY price;
```

Result

Prod_Name	Total_Sales	Total_Qty	Price
Gold Tips	38913.75	11563	3.36
Special Tips	38596.00	11390	3.38
Earl Grey	41137.00	11364	3.61
Assam Grade A	39205.00	10767	3.64
Breakfast Blend	42295.50	10880	3.88
English Breakfast	44381.00	10737	4.13
Irish Breakfast	48759.00	11094	4.39

Coffee Mug	1054.00	213	4.94
Darjeeling Number 1	62283.25	11539	5.39
Ruby's Allspice	133188.50	23444	5.68
Assam Gold Blend	71419.00	11636	6.13
Colombiano	188474.50	27548	6.84
Aroma Roma	203544.00	28344	7.18
La Antigua	197069.50	26826	7.34
Veracruzano	201230.00	26469	7.60
Espresso XO	224020.00	28558	7.84
Aroma baseball cap	15395.35	1953	7.88
Lotta Latte	217994.50	26994	8.07
Cafe Au Lait	213510.00	26340	8.10
Aroma Sounds Cassette	5206.00	620	8.39
Xalapa Lapa	251590.00	29293	8.58
NA Lite	231845.00	25884	8.95
Demitasse Ms	282385.25	28743	9.82
Aroma t-shirt	20278.50	1870	10.84
Travel Mug	1446.35	133	10.87
Darjeeling Special	127207.00	10931	11.63
Spice Sampler	6060.00	505	12.00
Aroma Sounds CD	7125.00	550	12.95
French Press, 2-Cup	3329.80	224	14.86
Spice Jar	4229.00	235	17.99
French Press, 4-Cup	3323.65	167	19.90
Tea Sampler	13695.00	550	24.90
...			

Using the arithmetic operators: (), +, -, *, /

You can perform arithmetic operations within a select list or within a search condition. A complete set of arithmetic operators is listed in the following table. The order of evaluation precedence is from highest to lowest (top to bottom) and, within a given level, left to right, in the table:

Operator	Name
()	Forces order of evaluation

+, -	Positive and negative
*, /	Multiplication and division
+, -	Addition and subtraction

If you have any doubt about the order of evaluation for a given expression, group the expression with parentheses. For example, the server evaluates $(4 + 3 * 2)$ as 10 but evaluates the grouped expression $((4 + 3) * 2)$ as 14.

Usage notes

The DEC function is used to remove all but two of the decimal places from each **Price** value:

```
dec(sum(dollars)/sum(quantity), 7, 2) AS price
```

For more information about the DEC function and other datatype format functions, refer to the [SQL Reference Guide](#).

Section 5. Comparing running totals with OLAP

Question

How do the Western and Southern regions compare in terms of running daily sales totals for March 2006?

OLAP query

```
SELECT t1.date, sales_cume_west, sales_cume_south,
       sales_cume_west - sales_cume_south AS west_vs_south
FROM   (SELECT date, SUM(dollars) AS total_sales,
              SUM(SUM(dollars)) OVER(ORDER BY date
              ROWS UNBOUNDED PRECEDING) AS sales_cume_west
        FROM   aroma.market a,
              aroma.store b,
              aroma.sales c,
              aroma.period d
        WHERE  a.mktkey = b.mktkey
              AND b.storekey = c.storekey
              AND d.perkey = c.perkey
              AND year = 2006
              AND month = 'MAR'
              AND region = 'West'
        GROUP BY date) AS t1
JOIN   (SELECT date, SUM(dollars) AS total_sales,
              SUM(SUM(dollars)) OVER(ORDER BY date
```

```

        ROWS UNBOUNDED PRECEDING) AS sales_cume_south
FROM aroma.market a,
     aroma.store b,
     aroma.sales c,
     aroma.period d
WHERE  a.mktkey = b.mktkey
      AND b.storekey = c.storekey
      AND d.perkey = c.perkey
      AND year = 2006
      AND month = 'MAR'
      AND region = 'South'
GROUP BY date) AS t2
ON t1.date = t2.date
ORDER BY date;

```

Result

DATE	Sales_Cume_West	Sales_Cume_South	WEST_VS_SOUTH
2006-03-01	2529.25	2056.75	472.50
2006-03-02	6809.00	4146.75	2662.26
2006-03-03	9068.75	6366.55	2702.20
...			
2006-03-29	100513.85	62891.35	37622.50
2006-03-30	104267.40	65378.75	38888.65
2006-03-31	107222.15	68100.75	39121.40

OLAP ORDER BY clause

One advantage of the OLAP approach to running sequential calculations is the ability to place them inside subqueries. OLAP ordering is part of the function itself, and each OLAP function has its own ORDER BY clause, independent of the query's final ORDER BY clause.

In this query the ORDER BY DATE clause is included within each OLAP function to ensure the correct values are used to compute the running totals. At the end of the query there is a further ORDER BY DATE clause which controls how the result set is displayed.

Notes on the query

The running comparison between Western and Southern sales is calculated using simple arithmetic in the SELECT statement.

Section 6. Moving averages

Sales figures fluctuate over time; when they fluctuate radically, they obscure underlying, long-range trends. Moving averages are used to smooth the effects of these fluctuations. For example, a three-week moving average divides the sum of the last three consecutive weekly aggregations by three.

Question

What was the three-week moving average of product sales at San Jose and Miami stores during the third quarter of 2005?

OLAP query

```
SELECT city, week, SUM(dollars) AS sales,
       DEC(AVG(SUM(dollars)) OVER(partition by city
                                ORDER BY city, week ROWS 2 PRECEDING),7,2) AS mov_avg,
       SUM(SUM(dollars)) OVER(PARTITION BY city
                               ORDER BY week ROWS unbounded PRECEDING) AS run_sales
FROM   aroma.store a,
       aroma.sales b,
       aroma.period c
WHERE  a.storekey = b.storekey
       AND c.perkey = b.perkey
       AND qtr = 'Q3_05'
       AND city IN ('San Jose', 'Miami')
GROUP BY city, week;
```

In the following result set, note that the averages in the first two rows for each city are not a three-week moving average; there is not yet sufficient data to complete such a calculation. Instead, those two averages are calculated over the first row (one week average) and the first and second rows (two week average), respectively.

Result

City	Week	Sales	Mov_avg	Run_sales
Miami	27	1838.55	1838.55	1838.55
Miami	28	4482.15	3160.35	6320.70
Miami	29	4616.70	3645.80	10937.40
Miami	30	4570.35	4556.40	15507.75
Miami	31	4681.95	4623.00	20189.70
...				
Miami	38	5500.25	5235.00	49493.35
Miami	39	4891.40	5346.71	54384.75
Miami	40	3693.80	4695.15	58078.55
...				
San Jose	27	3177.55	3177.55	3177.55

San Jose	28	5825.80	4501.67	9003.35
San Jose	29	8474.80	5826.05	17478.15
San Jose	30	7976.60	7425.73	25454.75
San Jose	31	7328.65	7926.68	32783.40
San Jose	32	6809.75	7371.66	39593.15
San Jose	33	7116.35	7084.91	46709.50
...				

The DEC function is used to define how many digits will be shown in the values returned for the **Mov_Avg** column. For details about this function, refer to the [SQL Reference Guide](#).

OLAP AVG function

The OLAP AVG function is used in conjunction with the following window frame:

```
ROWS n PRECEDING
```

where n is a number that represents the required smoothing factor. In earlier queries this value was generally set using the keyword `unbounded`. In this query, where you want to base the moving average on 3 rows, you insert the value **2**, indicating the current row plus the two preceding rows.

The OLAP ORDER BY clause ensures that the AVG function is applied to the correct moving sequence of rows (in this case, **WEEK**). The PARTITION BY clause identifies the value by which the AVG function will be reset (in this case, **CITY**).

Section 7. Moving sums

Question

What was the seven-day moving sum of quantities of Demitasse Ms coffee sold during March, 2006?

OLAP query

```
SELECT date, SUM(quantity) AS day_qty,
       DEC(SUM(SUM(quantity)) OVER(ORDER BY date
                                   ROWS 6 PRECEDING),7,2) AS mov_sum
```

```

FROM aroma.sales a, aroma.period b, aroma.product c
WHERE   b.perkey = a.perkey
        AND c.classkey = a.classkey
        AND c.prodkey = a.prodkey
        AND year = 2006
        AND month = 'MAR'
        AND prod_name = 'Demitasse Ms'
GROUP BY date
ORDER BY date;

```

The following OLAP result set contains moving sums for all of the rows. As in the moving average query earlier, the actual values for the first six rows do not display the expected seven-day moving sum; you need seven rows to accurately display that value.

Result

Date	Day_Qty	Mov_Sum
2006-03-01	65	65.00
2006-03-02	19	84.00
2006-03-03	92	176.00
2006-03-04	91	267.00
2006-03-05	106	373.00
2006-03-06	92	465.00
2006-03-07	102	567.00
2006-03-08	21	523.00
2006-03-09	74	578.00
2006-03-10	81	567.00
2006-03-11	77	553.00
2006-03-12	127	574.00
2006-03-13	169	651.00
2006-03-14	31	580.00
2006-03-15	56	615.00
2006-03-16	40	581.00
2006-03-17	84	584.00
2006-03-18	34	541.00
2006-03-19	128	542.00
2006-03-20	97	470.00
2006-03-21	50	489.00
2006-03-22	147	580.00
...		

A moving sum function, like a moving average, is used to smooth the effects of

fluctuations. For example, a seven-day moving sum is calculated by summing seven consecutive days.

The DEC scalar function is used to define the decimal values returned for the **Mov_Sum** column. For details about this function, refer to the [SQL Reference Guide](#). Note that in this example, the DEC causes the results to be displayed in decimal format even though the underlying data is in integer format.

OLAP SUM function

The OLAP SUM function produces moving sums when the window frame specifies:

```
ROWS n PRECEDING
```

The **FOLLOWING** keyword can also be used.

The OLAP ORDER BY clause is critical. This specification ensures that the input rows for the OLAP SUM function are sorted correctly (by ascending **Date**, in this case).

Section 8. Ranking data

Question

What were the March 2005 rankings of stores in the Western region, in terms of total dollar sales?

OLAP query

```
SELECT store_name, district, SUM(dollars) AS total_sales,
       RANK() OVER(ORDER BY SUM(dollars) DESC) AS sales_rank
FROM   aroma.market a,
       aroma.store b,
       aroma.sales c,
       aroma.period d
WHERE  a.mktkey = b.mktkey
      AND b.storekey = c.storekey
      AND d.perkey = c.perkey
      AND year = 2005
      AND month = 'MAR'
      AND region = 'West'
GROUP BY store_name, district;
```

Result

Store_Name	District	Total_Sales	Sales_Rank
Cupertino Coffee Supply	San Francisco	18670.50	1
Java Judy's	Los Angeles	18015.50	2
Beaches Brew	Los Angeles	18011.55	3
San Jose Roasting Company	San Francisco	17973.90	4
Instant Coffee	San Francisco	15264.50	5
Roasters, Los Gatos	San Francisco	12836.50	6

The daily totals from the **Sales** table that meet the search conditions in the WHERE clause are summed, then ranked.

OLAP RANK function

IBM DB2 database supports a set of OLAP ranking functions, including RANK() and DENSE_RANK(). These functions do not require any arguments; instead, the OLAP ORDER BY clause defines the column or expression on which the rankings are based. Note that the default sort order for OLAP functions is ascending (ASC), which is often not the required ranking for business queries. You must specify the **DESC** keyword in the OLAP ORDER BY clause to assign ranks from high to low (where 1 is the highest value).

The DENSE_RANK function differs from RANK in one respect: When you use the RANK function, if there is a tie between two or more rows the rank value skips down. If two rows are tied with a value of 2, the next ranking would be 4 (as in 1,2,2,4). When you use DENSE_RANK, there is no gap in the sequence of ranked values. For example, if two rows are ranked 8, the next ranking is still 9, as shown in the following example:

```
SELECT prod_name, dollars, DENSE_RANK() OVER(ORDER BY dollars DESC) AS dense_rank,
       RANK() OVER(ORDER BY dollars DESC) AS tie_rank
FROM   aroma.product a,
       aroma.sales b,
       aroma.period c
WHERE  a.prodkey = b.prodkey
       AND a.classkey = b.classkey
       AND c.perkey = b.perkey
       AND date = '01-03-2006';
```

Result

PROD_NAME	DOLLARS	DENSE_RANK	TIE_RANK
-----------	---------	------------	----------

Espresso Machine Italiano	499.75	1	1
Cafe Au Lait	392.00	2	2
Veracruzano	360.00	3	3
Lotta Latte	328.00	4	4
NA Lite	306.00	5	5
Colombiano	283.50	6	6
Darjeeling Special	207.00	7	7
Colombiano	202.50	8	8
Colombiano	202.50	8	8
Espresso XO	201.50	9	10
Xalapa Lapa	195.50	10	11
...			

Section 9. Using DATE mathematics

Question

Calculate a date 90 days prior to and 90 days after a given date.

Example query

```
SELECT date - 90 DAYS AS due_date,
       date AS cur_date,
       date + 90 DAYS AS past_due
FROM aroma.period
WHERE year = 2004
      AND month = 'JAN';
```

Result

Due_Date	Cur_Date	Past_Due
2003-10-03	2004-01-01	2004-03-31
2003-10-04	2004-01-02	2004-04-01
2003-10-05	2004-01-03	2004-04-02
...		

Incrementing or decrementing dates

You can either add to or delete from a date. To do so you must specify the following information:

- Value to be incremented or decremented (column name or datetime expression)
- Datepart that specifies the increment measure(s). These can be day, month, year or a combination of these values.
- Positive or negative increment value

About the query

The example query calculates a date 90 days before and 90 days after a given date. The function returns the value in the ANSI SQL 92 datetime format.

Be certain which value you want to adjust. For example, the value "90 days" is not identical to "3 months," in most cases. If you changed the above query to 3 months, then, you would likely get a different result set:

```
SELECT date - 3 MONTHS AS due_date,
       date AS cur_date,
       date + 3 MONTHS AS past_due
FROM aroma.period
WHERE year = 2004
      AND month = 'JAN';
```

Result

Due_Date	Cur_Date	Past_Due
2003-10-01	2004-01-01	2004-04-01
2003-10-02	2004-01-02	2004-04-02
2003-10-03	2004-01-03	2004-04-03
...		

These variances can become confusing if you are not careful. You should pay attention to special cases such as Leap Year to make certain you get the information you need.

You can also adjust a combination of values, as in:

```
SELECT date - 3 MONTHS - 4 DAYS AS due_date,
       date AS cur_date,
       date + 3 MONTHS + 4 DAYS AS past_due
FROM aroma.period
WHERE year = 2004
```

```
AND month = 'JAN';
```

Result

Due_Date	Cur_Date	Past_Due
2003-09-27	2004-01-01	2004-04-05
2003-09-28	2004-01-02	2004-04-06
2003-09-29	2004-01-03	2004-04-07
...		

You can similarly adjust aspects of any TIME datatype (HOURS, MINUTES, and/or SECONDS) or any TIMESTAMP datatype (YEARS, MONTHS, DAYS, HOURS, MINUTES, and/or SECONDS). For more information about DATE, TIME, and TIMESTAMP arithmetic, refer to the [SQL Reference Guide](#).

Section 10. Using the HAVING clause to exclude groups

Question

Which products had total sales of less than \$25,000 during 2005?

Example query

```
SELECT prod_name, sum(dollars) AS total_sales
FROM aroma.product a, aroma.sales b, aroma.period c
WHERE a.prodkey = b.prodkey
      AND a.classkey = b.classkey
      AND c.perkey = b.perkey
      AND year = 2005
GROUP BY prod_name
HAVING sum(dollars) < 25000
ORDER BY total_sales DESC;
```

Result

Prod_Name	Total_Sales
Aroma t-shirt	21397.65
Espresso Machine Royale	18119.80
Espresso Machine Italiano	17679.15
Coffee Sampler	16634.00
Tea Sampler	14907.00

Aroma baseball cap	13437.20
Aroma Sheffield Steel Teapot	8082.00
Spice Sampler	7788.00
Aroma Sounds CD	5937.00
Aroma Sounds Cassette	5323.00
French Press, 4-Cup	4570.50
Spice Jar	4073.00
French Press, 2-Cup	3042.75
Travel Mug	1581.75
Easter Sampler Basket	1500.00
Coffee Mug	1258.00
Christmas Sampler	1230.00

Conditions on groups: HAVING clause

Although dividing data into groups reduces the amount of information returned, queries often still return more information than you need. You can use a HAVING clause to exclude groups that fail to satisfy a specified condition, such as sums of dollars that are less than or higher than a given number.

This query calculates the total sales revenue for each product in 2005, then retains only those products whose totals fall below \$25,000.

Syntax of the HAVING clause

```
SELECT column name(s)
       FROM table name(s)
       [WHERE search_condition]
       [GROUP BY group_list]
       [HAVING condition]
       [ORDER BY order_list];
```

condition	An SQL condition that can include set functions.
-----------	--

The HAVING clause differs from the WHERE clause in the following ways.

WHERE Clause	HAVING Clause
Works on rows of data prior to grouping.	Works on the result set after grouping.
Conditions cannot be expressed with set functions (for example, SUM or AVG), but column aliases for nonaggregate expressions can be used.	Conditions can be expressed with any set function or column alias.

Usage notes

Any set function can be referenced in a condition in the HAVING clause. A query with a HAVING clause must contain a GROUP BY clause, unless the select list contains only set functions. For example:

```
SELECT MIN(prodkey), MAX(classkey)
       FROM aroma.product
       HAVING MIN(prodkey) = 0;
```

Section 11. Summary

Summary

This tutorial, part 4 in a series, described how to:

- Use SQL OLAP functions to perform data analysis
- Use DATE scalar functions and DATE arithmetic to calculate and extract date information from DATETIME columns

Analytic functions

SQL OLAP functions can be used to answer a wide range of business questions that require ranks, ratios, moving sums and averages, and so on. The examples in this tutorial concentrate on just some of the calculations that can be handled by an OLAP function. In general, OLAP functions are quite flexible, and greatly simplify the complex SQL you would otherwise have to write.

Downloads

Description	Name	Size	Download method
Aroma Database	Aroma_Data.zip	1MB	HTTP

[Information about download methods](#)

Resources

Learn

- View this article series' "[Appendix A](#)" (developerWorks, August 2006).
- Read "[DB2 XML evaluation guide](#)" (developerWorks, June 2006), a step-by-step tutorial introducing the reader to the DB2 Viper data server on Windows platforms using the XML storage and searching (SQL/XML, XQuery) capabilities available to support next-generation applications.
- Check out this article and "[Get off to a fast start with DB2 Viper](#)" (developerWorks, March 2006).
- Learn how to "[Query DB2 XML data with XQuery](#)" (developerWorks, April 2006).
- Learn how to "[Query DB2 XML data with SQL](#)" (developerWorks, March 2006).
- Read the [IBM Systems Journal](#) and celebrate 10 years of XML.
- Refer to the [SQL Reference, Vol 1](#) for additional information.
- Refer to the [SQL Reference, Vol 2](#) for additional information.
- Refer to the [DB2 information Center](#) for troubleshooting.
- Visit the [DB2 XML technical enablement space](#) for links to more than 25 papers on DB2 XML capabilities.

Get products and technologies

- Download [DB2 Express-C](#), a no-charge data server for use in development and deployment of applications .

Discuss

- [Participate in the discussion forum for this content.](#)
- Visit the [DB2 9 On-line Support Forum](#).

About the authors

Pat Moffatt

Pat Moffatt is the Information Management Program Manager for the IBM Academic Initiative. Through the Academic Initiative program, she ensures that appropriate Information Management resources are made available to help faculty integrate Information Management software into their curriculum. To learn more about this program, visit www.ibm.com/university/data.

Bruce Creighton

Bruce Creighton is a Skills Segment Planner in the Information Management Education Planning and Development department. In this role, he plans investment in educational content and balances the investment between areas where IBM can attain revenue and those where the requirement for skills development are important enough to provide free education.

Jessica Cao

Jessica Cao is an Arts and Science and Computer Science student at McMaster University. She expects to complete her combined honours degree in April 2009. Jessica is working in IBM Toronto lab's DB2 Information Management Skills Channel Planning and Enablement Program to take advantage of her interest in programming, editing, and writing.

Trademarks

IBM, DB2, Universal Database, OS/2, and pureXML are registered trademarks of IBM Corporation in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.