|  |
| --- |
| , RD Dep. |
| **Dataset Transformations:**  **Pivoting and Unpivoting** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial Version | Aliaksei Belablotski | 17-Feb-2012 |  |  |
|  |  |  |  |  |  |

Contents

[1. Pivoting Operations 3](#_Toc317442619)

[1.1. Pivoting on Multiple Columns 3](#_Toc317442620)

[1.2. Pivoting: Multiple Aggregates 3](#_Toc317442621)

[1.3. Distinguishing PIVOT-Generated Nulls from Nulls in Source Data 3](#_Toc317442622)

[2. Unpivoting Operations 4](#_Toc317442623)

[3. Wildcard and Subquery Pivoting 4](#_Toc317442624)

# Pivoting Operations

The data returned by business intelligence queries is often most usable if presented in a crosstabular format. The [pivot\_clause](http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_10002.htm#CHDCEJJE) of the SELECT statement lets you write crosstabulation queries that rotate rows into columns, aggregating data in the process of the rotation. Pivoting is a key technique in data warehouses. In it, you transform multiple rows of input into fewer and generally wider rows in the data warehouse. When pivoting, an aggregation operator is applied for each item in the pivot column value list. The pivot column cannot contain an arbitrary expression. If you need to pivot on an expression, then you should alias the expression in a view before the PIVOT operation. The basic syntax is as follows:

SELECT ....

FROM <table-expr>

PIVOT

(

aggregate-function(<column>)

FOR <pivot-column> IN (<value1>, <value2>,..., <valuen>)

) AS <alias>

WHERE .....

## Pivoting on Multiple Columns

You can pivot on more than one column. The following statement illustrates a typical multiple column pivot:

SELECT \*

FROM

(SELECT product, channel, quarter, quantity\_sold

FROM sales\_view

) PIVOT (SUM(quantity\_sold)

FOR (channel, quarter) IN

((5, '02') AS CATALOG\_Q2,

(4, '01') AS INTERNET\_Q1,

(4, '04') AS INTERNET\_Q4,

(2, '02') AS PARTNERS\_Q2,

(9, '03') AS TELE\_Q3

)

);

Note that this example specifies a multi-column IN-list with column headings designed to match the IN-list members.

## Pivoting: Multiple Aggregates

You can pivot with multiple aggregates, as shown in the following example:

SELECT \*

FROM

(SELECT product, channel, amount\_sold, quantity\_sold

FROM sales\_view

) PIVOT (SUM(amount\_sold) AS sums,

SUM(quantity\_sold) AS sumq

FOR channel IN (5, 4, 2, 9)

)

ORDER BY product;

Note that the query creates column headings by concatenating the pivot values (or alias) with the alias of the aggregate function, plus an underscore.

## Distinguishing PIVOT-Generated Nulls from Nulls in Source Data

You can distinguish between null values that are generated from the use of PIVOT and those that exist in the source data. The following example illustrates nulls that PIVOT generates.

The following query returns rows with 5 columns, column prod\_id, and pivot resulting columns Q1, Q1\_COUNT\_TOTAL, Q2, Q2\_COUNT\_TOTAL. For each unique value of prod\_id, Q1\_COUNT\_TOTAL returns the total number of rows whose qtr value is Q1, that is, and Q2\_COUNT\_TOTAL returns the total number of rows whose qtr value is Q2.

Assume we have a table sales2 of the following structure:

PROD\_ID QTR AMOUNT\_SOLD

------- --- -----------

100 Q1 10

100 Q1 20

100 Q2 NULL

200 Q1 50

SELECT \*

FROM sales2

PIVOT

( SUM(amount\_sold), COUNT(\*) AS count\_total

FOR qtr IN ('Q1', 'Q2')

);

PROD\_ID "Q1" "Q1\_COUNT\_TOTAL" "Q2" "Q2\_COUNT\_TOTAL"

------- ---- ---------------- --------- ----------------

100 20 2 NULL <1> 1

200 50 1 NULL <2> 0

From the result, we know that for prod\_id 100, there are 2 sales rows for quarter Q1, and 1 sales row for quarter Q2; for prod\_id 200, there is 1 sales row for quarter Q1, and no sales row for quarter Q2.So, in Q2\_COUNT\_TOTAL, you can identify that **NULL<1> comes from a row in the original table whose measure is of null value, while NULL<2> is due to no row being present in the original table** for prod\_id 200 in quarter Q2.

# Unpivoting Operations

An unpivot does not reverse a PIVOT operation. Instead, it rotates data from columns into rows. If you are working with pivoted data, an UNPIVOT operation cannot reverse any aggregations that have been made by PIVOT or any other means.

To illustrate unpivoting, first create a pivoted table that includes four columns, for quarters of the year:

CREATE TABLE pivotedTable AS

SELECT \*

FROM (SELECT product, quarter, quantity\_sold

FROM sales\_view

)

PIVOT

(

SUM(quantity\_sold)

FOR quarter IN ('01' AS Q1, '02' AS Q2, '03' AS Q3, '04' AS Q4));

The following UNPIVOT operation will rotate the quarter columns into rows. For each product, there will be four rows, one for each quarter.

SELECT \*

FROM pivotedTable

UNPIVOT INCLUDE NULLS

(

quantity\_sold

FOR quarter IN (Q1, Q2, Q3, Q4))

ORDER BY product, quarter;

Note the use of INCLUDE NULLS in this example. You can also use EXCLUDE NULLS, which is the default setting.

# Wildcard and Subquery Pivoting

It’s impossible to use wildcards for specifying set of column for PIVOT operation.

For example, pivot the dataset below:

SELECT PROD\_NAME, CHANNEL\_DESC, sum(QUANTITY\_SOLD)

FROM SALES S, CHANNELS C, PRODUCTS P

WHERE S.PROD\_ID=P.PROD\_ID AND S.CHANNEL\_ID=C.CHANNEL\_ID

GROUP BY PROD\_NAME, CHANNEL\_DESC;

Pivoting is going on 'Partners', 'Direct Sales', 'Internet' and 'Tele Sales' channels:

SELECT \*

FROM

(SELECT PROD\_NAME as PRODUCT, CHANNEL\_DESC as CHANNEL, QUANTITY\_SOLD

FROM SALES S, CHANNELS C, PRODUCTS P

WHERE S.PROD\_ID=P.PROD\_ID AND S.CHANNEL\_ID=C.CHANNEL\_ID

) PIVOT (SUM(QUANTITY\_SOLD)

FOR CHANNEL IN ('Partners', 'Direct Sales', 'Internet', 'Tele Sales')

);

But according [the specification](http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_10002.htm#CHDCEJJE) it’s possible to use ANY wildcard or even subquery in FOR CHANNEL IN (…) construction.

SELECT \*

FROM

(SELECT PROD\_NAME as PRODUCT, CHANNEL\_DESC as CHANNEL, QUANTITY\_SOLD

FROM SALES S, CHANNELS C, PRODUCTS P

WHERE S.PROD\_ID=P.PROD\_ID AND S.CHANNEL\_ID=C.CHANNEL\_ID

) PIVOT (SUM(QUANTITY\_SOLD)

FOR CHANNEL IN (ANY)

);

Or

SELECT \*

FROM

(SELECT PROD\_NAME as PRODUCT, CHANNEL\_DESC as CHANNEL, QUANTITY\_SOLD

FROM SALES S, CHANNELS C, PRODUCTS P

WHERE S.PROD\_ID=P.PROD\_ID AND S.CHANNEL\_ID=C.CHANNEL\_ID

) PIVOT (SUM(QUANTITY\_SOLD)

FOR CHANNEL IN (SELECT DISTINCT CHANNEL\_DESC FROM CHANNELS)

);

Unfortunately both queries will return an error “ORA-00936: missing expression”.

If you want to use a wildcard argument or subquery in your pivoting columns, you can do so with PIVOT XML syntax. With PIVOT XML, the output of the operation is properly formatted XML.

The following example illustrates using the wildcard keyword, ANY. It will output XML that includes all channel values:

SELECT \*

FROM

(SELECT PROD\_NAME as PRODUCT, CHANNEL\_DESC as CHANNEL, QUANTITY\_SOLD

FROM SALES S, CHANNELS C, PRODUCTS P

WHERE S.PROD\_ID=P.PROD\_ID AND S.CHANNEL\_ID=C.CHANNEL\_ID

) PIVOT XML(SUM(quantity\_sold)

FOR channel IN (ANY)

);

Note that the keyword ANY is available in PIVOT operations only as part of an XML operation. This output includes data for cases where the channel exists in the data set. Also note that aggregation functions must specify a GROUP BY clause to return multiple values, yet the pivot\_clause does not contain an explicit GROUP BY clause. Instead, the [pivot\_clause](http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_10002.htm#CHDCEJJE) performs an implicit GROUP BY.



For the first product (1.44MB External 3.5" Diskette) pivot set will be

<PivotSet>

<item>

<column name = "CHANNEL">Direct Sales</column>

<column name = "SUM(QUANTITY\_SOLD)">14189</column>

</item>

<item>

<column name = "CHANNEL">Internet</column>

<column name = "SUM(QUANTITY\_SOLD)">2464</column>

</item>

<item>

<column name = "CHANNEL">Partners</column>

<column name = "SUM(QUANTITY\_SOLD)">6455</column>

</item>

</PivotSet>

For forth product (18" Flat Panel Graphics Monitor) pivot set will be slighly different (Tele Sales channel will appear):

<PivotSet>

<item>

<column name="CHANNEL">Direct Sales</column>

<column name="SUM(QUANTITY\_SOLD)">2775</column>

</item>

<item>

<column name="CHANNEL">Internet</column>

<column name="SUM(QUANTITY\_SOLD)">1127</column>

</item>

<item>

<column name="CHANNEL">Partners</column>

<column name="SUM(QUANTITY\_SOLD)">1076</column>

</item>

<item>

<column name="CHANNEL">Tele Sales</column>

<column name="SUM(QUANTITY\_SOLD)">227</column>

</item>

</PivotSet>

The following example illustrates using a subquery. It will output XML that includes all channel values and the sales data corresponding to each channel:

SELECT \*

FROM

(SELECT PROD\_NAME as PRODUCT, CHANNEL\_DESC as CHANNEL, QUANTITY\_SOLD

FROM SALES S, CHANNELS C, PRODUCTS P

WHERE S.PROD\_ID=P.PROD\_ID AND S.CHANNEL\_ID=C.CHANNEL\_ID

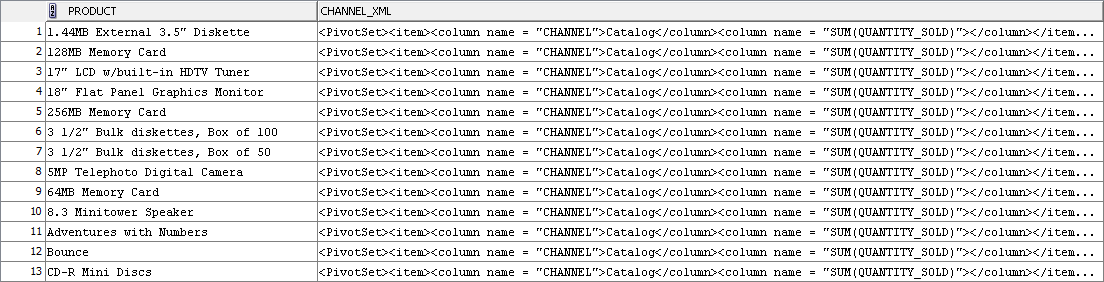
) PIVOT XML(SUM(QUANTITY\_SOLD)

--FOR channel IN (ANY)

FOR CHANNEL IN (SELECT DISTINCT CHANNEL\_DESC FROM CHANNELS)

)

ORDER BY 1;



For the first product (1.44MB External 3.5" Diskette) pivot set will be

<PivotSet>

<item>

<column name="CHANNEL">Catalog</column>

<column name="SUM(QUANTITY\_SOLD)"></column>

</item>

<item>

<column name="CHANNEL">Direct Sales</column>

<column name="SUM(QUANTITY\_SOLD)">14189</column>

</item>

<item>

<column name="CHANNEL">Internet</column>

<column name="SUM(QUANTITY\_SOLD)">2464</column>

</item>

<item>

<column name="CHANNEL">Partners</column>

<column name="SUM(QUANTITY\_SOLD)">6455</column>

</item>

<item>

<column name="CHANNEL">Tele Sales</column>

<column name="SUM(QUANTITY\_SOLD)"></column>

</item>

</PivotSet>

Note that the output densifies the data to include all possible channels for each product.