|  |
| --- |
| , RD Dep. |
| **Oracle SQL for Aggregation in Data Warehouses** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial Version | Volha Kutsevol | 13-Feb-2012 |  |  |
|  |  |  |  |  |  |

Contents

[1. Overview of SQL for Aggregation in Data Warehouses 3](#_Toc316991322)

[1.1. SQL Aggregation functions 3](#_Toc316991323)

[1.2. Sales History (SH) 3](#_Toc316991324)

[2. SQL Grouping Extensions 4](#_Toc316991325)

[2.1. SQL GROUP BY Statement 4](#_Toc316991326)

[2.2. ROLLUP Extension to GROUP BY 4](#_Toc316991327)

[2.3. CUBE Extension to GROUP BY 5](#_Toc316991328)

[2.4. GROUPING Functions: GROUPING, GROUPING\_ID, GROUP\_ID 5](#_Toc316991329)

[3. Grouping Sets 6](#_Toc316991330)

[3.1. Grouping Sets Expression 6](#_Toc316991331)

[3.2. Composite Columns 7](#_Toc316991332)

[3.3. Concatenated Groups 7](#_Toc316991333)

# Overview of SQL for Aggregation in Data Warehouses

## SQL Aggregation functions

Aggregation is a fundamental part of data warehousing. To improve aggregation performance in your warehouse, Oracle Database provides the following extensions to the GROUP BY clause:

* **CUBE** and **ROLLUP** extensions to the GROUP BY clause
* **GROUPING** functions
* **GROUPING SETS** expression

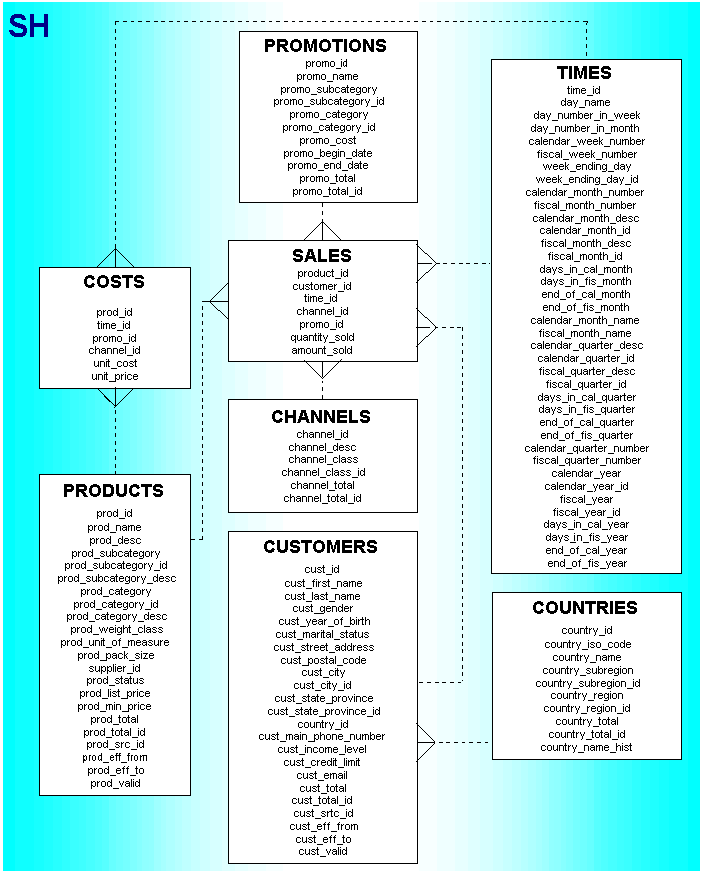
The CUBE, ROLLUP, and GROUPING SETS extensions to SQL make querying and reporting easier and faster. CUBE, ROLLUP, and grouping sets produce a single result set that is equivalent to a UNION ALL of differently grouped rows. ROLLUP calculates aggregations such as SUM, COUNT, MAX, MIN, and AVG at increasing levels of aggregation, from the most detailed up to a grand total. CUBE is an extension similar to ROLLUP, enabling a single statement to calculate all possible combinations of aggregations. The CUBE, ROLLUP, and the GROUPING SETS extension lets you specify just the groupings needed in the GROUP BY clause. This allows efficient analysis across multiple dimensions without performing a CUBE operation. Computing a CUBE creates a heavy processing load, so replacing cubes with grouping sets can significantly increase performance.

To enhance performance, CUBE, ROLLUP, and GROUPING SETS can be parallelized: multiple processes can simultaneously execute all of these statements. These capabilities make aggregate calculations more efficient, thereby enhancing database performance, and scalability.

The three GROUPING functions help you identify the group each row belongs to and enable sorting subtotal rows and filtering results.

## Sales History (SH)

The **SH** schema logically depends on the OE schema, although nothing prevents you from creating this schema on its own, without the four other schemas.



# SQL Grouping Extensions

## SQL GROUP BY Statement

The **GROUP BY** statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

**GROUP BY Syntax:**

## ROLLUP Extension to GROUP BY

**ROLLUP** enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions. It also calculates a grand total. ROLLUP is a simple extension to the GROUP BY clause, so its syntax is extremely easy to use. The ROLLUP extension is highly efficient, adding minimal overhead to a query.

The action of ROLLUP is straightforward: it creates subtotals that roll up from the most detailed level to a grand total, following a grouping list specified in the ROLLUP clause. ROLLUP takes as its argument an ordered list of grouping columns. First, it calculates the standard aggregate values specified in the GROUP BY clause. Then, it creates progressively higher-level subtotals, moving from right to left through the list of grouping columns. Finally, it creates a grand total.

ROLLUP creates subtotals at n+1 levels, where n is the number of grouping columns. You might want to compress your data when using ROLLUP. This is particularly useful when there are few updates to older partitions.

Use the ROLLUP extension in tasks involving subtotals.

* It is very helpful for subtotaling along a hierarchical dimension such as time or geography. For instance, a query could specify a ROLLUP(y, m, day) or ROLLUP (country, state, city).
* For data warehouse administrators using summary tables, ROLLUP can simplify and speed up the maintenance of summary tables.

**ROLLUP Syntax:**

## CUBE Extension to GROUP BY

**CUBE** takes a specified set of grouping columns and creates subtotals for all of their possible combinations. In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions. If you have specified CUBE(time, region, department), the result set will include all the values that would be included in an equivalent ROLLUP statement plus additional combinations.

Consider Using CUBE in any situation requiring cross-tabular reports. The data needed for cross-tabular reports can be generated with a single SELECT using CUBE. Like ROLLUP, CUBE can be helpful in generating summary tables. Note that population of summary tables is even faster if the CUBE query executes in parallel.

CUBE is typically most suitable in queries that use columns from multiple dimensions rather than columns representing different levels of a single dimension.

**CUBE Syntax:**

## GROUPING Functions: GROUPING, GROUPING\_ID, GROUP\_ID

Two challenges arise with the use of ROLLUP and CUBE. First, how can you programmatically determine which result set rows are subtotals, and how do you find the exact level of aggregation for a given subtotal? You often need to use subtotals in calculations such as percent-of-totals, so you need an easy way to determine which rows are the subtotals. Second, what happens if query results contain both stored NULL values and NULL values created by a ROLLUP or CUBE? How can you differentiate between the two?

**GROUPING Function**

GROUPING handles these problems. Using a single column as its argument, GROUPING returns 1 when it encounters a NULL value created by a ROLLUP or CUBE operation. That is, if the NULL indicates the row is a subtotal, GROUPING returns a 1. Any other type of value, including a stored NULL, returns a 0.

**GROUPING Syntax:**

**GROUPING\_ID Function**

To find the GROUP BY level of a particular row, a query must return GROUPING function information for each of the GROUP BY columns. If we do this using the GROUPING function, every GROUP BY column requires another column using the GROUPING function. For each row, GROUPING\_ID takes the set of 1's and 0's that would be generated if you used the appropriate GROUPING functions and concatenates them, forming a bit vector. The bit vector is treated as a binary number, and the number's base-10 value is returned by the GROUPING\_ID function.

**GROUPING\_ID Syntax:**

**GROUP\_ID Function**

It's possible to write queries that return the duplicate subtotals, which can be a little confusing. The GROUP\_ID function assigns the value "0" to the first set, and all subsequent sets get assigned a higher number. The following query forces duplicates to show the GROUP\_ID function in action.

**GROUP\_ID Syntax:**

# Grouping Sets

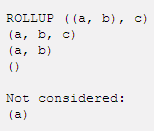
## Grouping Sets Expression

You can selectively specify the set of groups that you want to create using a **GROUPING SETS** expression within a GROUP BY clause. This allows precise specification across multiple dimensions without computing the whole CUBE.

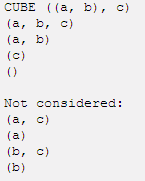
**GROUPING SETS Syntax:**

## Composite Columns

Composite columns allow columns to be grouped together with braces so they are treated as a single unit when determining the necessary groupings. In the following ROLLUP columns "a" and "b" have been turned into a composite column by the additional braces. As a result the group of "a" is not longer calculated as the column "a" is only present as part of the composite column in the statement.



In a similar way, the possible combinations of the following CUBE are reduced because references to "a" or "b" individually are not considered as they are treated as a single column when the groupings are determined.



## Concatenated Groups

Concatenated groupings offer a concise way to generate useful combinations of groupings. Groupings specified with concatenated groupings yield the cross-product of groupings from each grouping set. The cross-product operation enables even a small number of concatenated groupings to generate a large number of final groups. The concatenated groupings are specified simply by listing multiple grouping sets, cubes, and rollups, and separating them with commas.

