**An Introduction to Teradata OLAP Capabilities**

The Teradata SQL commands, using Teradata SQL Assistant, used for illustrating Teradata OLAP capabilities are based on the following data structure.



**Dillard’s Department Stores Sales Transactions**

Academic Units, faculty and students, that are members of the Teradata University Network -- free membership at <http://www.teradata.com/t/page/137474/index.html>, have access to a year of retail sales data from Dillard’s. Sensitive data has been removed but the data is very realistic and a rich environment for students and faculty for learning data base, data warehousing and OLAP concepts. Practice with larger more realistic datasets provides enriched learning opportunities not otherwise available. The ua\_dillards database consists of 5 tables with more than 120 million rows already populated in the TRNSACT table for your use. The data was provided by Dillard’s Department Stores, Inc. and contains the sales transaction data for August 2004 through July 2005 for 453 Dillard’s stores.

**Use this link** <http://enterprise.waltoncollege.uark.edu/membership.asp?show=tunregistration>

**to get a University of Arkansas Account.** From the drop down under Teradata University Network, click Member Information and complete the request forms. You may also refer to the “How to become a TUN member…” documentation up in the University of Arkansas TUN website for more information and Teradata basics. Link specified above.

Once you receive your University of Arkansas Teradata account, access will be via remote desktop connection. Remote access documentation is at the following link: <http://enterprise.waltoncollege.uark.edu/Remote_Desktop_TUN.pdf>

 **Example 1 -** (Examples 1-3 do not use any special OLAP features)

Dillard’s management wishes to know the best performing stores (by city) in terms of total sales for the period of August 2004 through July 2005 (note that the Dillard’s covers these dates)

**SELECT** s.store, s.city, sum(amt) **as** TotalSales

**FROM** ua\_dillards.trnsact t

 **INNER** **JOIN** ua\_dillards.strinfo s

 **ON** t.store = s.store

**GROUP** **BY** city, s.store

**ORDER** **BY** 3 **DESC**;

|  |  |  |  |
| --- | --- | --- | --- |
|  | **STORE**  | **CITY**  | **TotalSales**  |
| **1**  | 8402  | METAIRIE  | 27058653.42  |
| **2**  | 504  | LITTLE ROCK  | 25469899.01  |
| **3**  | 1607  | DALLAS  | 24553924.18  |
| **4**  | 2707  | MCALLEN  | 24124962.49  |
| **5**  | 9103  | LOUISVILLE  | 22787327.90  |
| **6**  | 7507  | HOUSTON  | 21536191.60  |
| **7**  | 2203  | OVERLAND PARK  | 20896515.90  |
| **8**  | 2007  | SAN ANTONIO  | 20396931.95  |
| **9**  | 9304  | OKLAHOMA CITY  | 20350217. |

**Example 2 -- using a WHERE clause to join instead of INNER JOIN**

**SELECT** s.store, s.city, sum(amt) **as** TotalSales

**FROM** trnsact t, strinfo s

**WHERE** t.store = s.store

**GROUP** **BY** city, s.store

**ORDER** **BY** 3 **DESC**;

Note that in both cases, the database name was used to qualify the table. In Teradata, one can specify the database which then allows creation of the SQL statements without the database name qualification. The following SQL illustrates this capability.

**DATABASE** ua\_dillards;

**SELECT** s.store, s.city, sum(amt) **as** TotalSales

**FROM** trnsact t, strinfo s

**WHERE** t.store = s.store

**GROUP** **BY** city, s.store

**ORDER** **BY** 3 **DESC**;

The DATABASE statement specifies the current database and remains in effect until another DATABASE statement is executed.

**Example 3**

Dillard’s management wishes to know the vendors and associated dollar amount of sales for the brand “Liz Clairborne”. The results should be from largest sales to smallest sales.

**SELECT** k.brand, k.vendor, sum(amt) **as** TotalSales

**FROM** ua\_dillards.trnsact t

 **INNER** **JOIN** ua\_dillards.skuinfo k **ON**

 t.sku = k.sku

**GROUP** **BY** k.brand, k.vendor

**ORDER** **BY** 3 **DESC**;

|  |  |  |  |
| --- | --- | --- | --- |
|  | **BRAND**  | **VENDOR**  | **TotalSales**  |
| **1**  | CLINIQUE  | 5511283  | 244726813.93  |
| **2**  | POLO FAS  | 5715232  | 208298981.49  |
| **3**  | LANCOME  | 0113645  | 165503299.30  |
| **4**  | EMMA JAM  | 3313116  | 74356782.77  |
| **5**  | **LIZ CLAI**  | **5531254**  | **34496517.43**  |
| **6**  | POLO FAS  | 5745232  | 33268376.36  |
| **7**  | BROWN SH  | 0060904  | 32418606.10  |
| **8**  | HART SCH  | 7045883  | 30127404.30  |
| **9**  | CHANEL I  | 6041161  | 29571162.76  |
| **10**  |  |  |  |

OR

**SELECT** k.brand, k.vendor, sum(amt) **as** TotalSales

**FROM** ua\_dillards.trnsact t

 **INNER** **JOIN** ua\_dillards.skuinfo k

 **ON** t.sku = k.sku

**WHERE** k.brand **LIKE** '%LIZ CLAI%' *(to retrieve only Liz Claiborne)*

**GROUP** **BY** k.brand, k.vendor

**ORDER** **BY** 3 **DESC**;

****

**OLAP – On-Line Analytical Processing Functions**

**Using OLAP to Analyze Data**

* On-Line Transactions Processing (OLTP) for recorded transactions from terminals
* On-Line Complex Processing (OLCP) for very complex queries
* On-Line Analytical Processing (OLAP) provide the ability to analyze large amounts of data (historical, transactions, trends, …) and provide data mining capabilities

**Similar to Functions, but more…**

* Like aggregate functions, OLAP operates on groups of rows and permit qualification and filtering.
* Unlike aggregate functions, OLAP return the individual row detail data and not just a final aggregate value.

**Basic Teradata OLAP Functions**:

CSUM – (Cumulation)

MAVG – (Moving Average)

MSUM – (Moving Sum)

MDIFF – (Moving Differences)

RANK – (Rankings)

QUANTILE – (Quantiles)

SAMPLE – (Sampling)

MLINREG – (Moving Linear Regression)

ROLLUP –subtotaling groups

CUBE – provides data warehouse type capabilities

One purpose of the Teradata OLAP functions is to allow data mining on a database using SQL. Note that

* OLAP functions are similar to aggregate functions
	+ Operate on groups of rows (like GROUP BY clause)
	+ Allow filtering groups using QUALIFY (like HAVING clause)
* OLAP functions are unlike aggregate functions
	+ Return data value for each qualifying now—not group
	+ May not be performed within subqueries
* OLAP functions may be performed on
	+ Tables
	+ Views
	+ INSERT/SELECT populations

**OLAP Examples\***

**Cumulative SUM - Cumulative Sum of Sales for Store 5203 before January 1**

General Form: **CSUM (colname, sort\_item1, sort\_item2…)**

 **Example 4**

Obtain the sale date, store, department and cumulative sales for Dillard’s department stores for January 1, 2005 for department 1704 in Abilene.

**SELECT** saledate, amt, city, brand, CSUM(amt, saledate)

**FROM** ua\_dillards.trnsact t, ua\_dillards.skuinfo k, ua\_dillards.strinfo s

**WHERE** t.sku = k.sku

**AND** t.store = s.store

**AND** t.saledate **BETWEEN** '2005-01-01' **AND** '2005-01-02'

**AND** k.dept=1704

 **AND** s.city='ABILENE';



**Moving Averages**

General form**: MAVG(colname, n, sort\_item1, sort\_item2, etc)**

**Example 5**

This simple example shows a moving average on a 7 day window for sales for SKU of ‘0000180’ which is in department ‘1704’ which happens to be a Ralph Lauren product.

**SELECT** saledate, k.sku, amt, dept, MAVG(amt,7,saledate)

**FROM** ua\_dillards.trnsact t, ua\_dillards.skuinfo k

**WHERE** t.sku = k.sku

**AND** t.sku <2000

**AND** DEPT=7104

**AND** EXTRACT(MONTH **FROM** saledate) **IN**(1,2)

**AND** EXTRACT(DAY **FROM** saledate) **IN**(21,22,23,24,25)

**ORDER** **BY** saledate, t.sku

****

**Moving sum and moving difference – replace MAVG with MSUM or MDIFF.**

**Simple and Qualified Rankings**

The **Ranking** function permits a column to be ranked, either based on high or low order, against other rows in the answer set. By default, the output will be sorted in descending sequence of the ranking column. The Rank function syntax is:

 **RANK(colname)**

The **QUALIFY** clause allows restriction of the output in the final result.

 **QUALIFY RANK(amt) <=7**

****

**Example 6**

Determine the highest selling products for store 204

**SELECT** Store, sku, amt, RANK(amt)

**FROM** ua\_dillards.trnsact

**WHERE** store= 204

**Example 7**

Get the top five selling products for store 204

**SELECT** Store, sku, amt, RANK(amt)

**FROm** ua\_dillards.trnsact

**WHERE** store= 204

**QUALIFY** RANK(amt) <= 5

**Example 8**

****

Get the top three selling products for each store

**SELECT** Store, sku, amt, RANK(amt)

**FROm** ua\_dillards.trnsact

**GROUP** **BY** store

**QUALIFY** RANK(amt) <= 3

**Example 8**

Get the top 10 selling products across all stores

**SELECT** tt.sku, tt.Sumamt, RANK(tt.Sumamt)

**FROM** (**SELECT** t.sku, SUM(t.amt)

 **FROM** ua\_dillards.trnsact t

 **GROUP** **BY** 1)

 **AS** tt(sku, Sumamt)

**QUALIFY** RANK(Sumamt) <= 10

**Example 9**

Get the ten poorest selling items (greater than $10) across all stores and order by product id -- The only syntax difference between the two queries is the ASC (sort order ascending) on the RANK function.

**SELECT** tt.sku, tt.Sumamt, RANK(tt.Sumamt)

**FROM** (**SELECT** t.sku, SUM(t.amt)

 **FROM** ua\_dillards.trnsact t

 **WHERE** amt >10

 **GROUP** **BY** 1)

 **AS** tt(sku, Sumamt)

**QUALIFY** RANK(Sumamt **ASC**) <= 10

**ORDER** **BY** 1



**Rollup and Cube OLAP features**

General Form: **rollup(col1, col2..)**

**Example 10 -- rollup**

Dillard’s management wishes to know the total sales by department, brand and sku number with the highest sales first. After reviewing this information it will be easy to rerun the query for any combination of single or multiple departments, brands, and skus. One can also limit the output by selecting only a predefined number—SELECT TOP 100, for example.

**DATABASE** ua\_dillards;

**SELECT** k.dept, k.brand, k.sku, sum(t.amt)

**FROM** skuinfo k, trnsact t

**WHERE** k.sku=t.sku

**GROUP** **BY** **ROLLUP**(k.dept, k.brand, k.sku)

**ORDER** **BY** 4 **DESC**, k.dept, k.brand, k.sku

****

**Example 11 -- Cube**

Use the cube capability to obtain the top 100 items in terms of total sales for each department and brand. The ? entries means the total for that column—thus, row one is the grand total for all items.

**DATABASE** ua\_dillards;

**SELECT** **TOP** 100 k.dept, k.brand, k.sku, sum(t.amt)

**FROM** skuinfo k, trnsact t

**WHERE** k.sku=t.sku

**GROUP** **BY** **CUBE**(k.dept, k.brand, k.sku)

**Order** **by** 4 **DESC**, k.dept, k.brand, k.sku;

****