BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DATA WAREHOUSING SS-G515
LAB-9 [OLAP]

DATE: 31/03/2023 **TIME: 02 Hours**

OLAP (Online Analytical Processing) is a category of database processing that facilitates business intelligence.
OLAP provides analysts, managers, and executives with the information they need to make effective decisions about an organization's strategic directions. OLAP can provide valuable insights into how their business is performing, as well as how they can make improvements. OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. OLAP can be used to find trends and get a big picture view of the data. It can also be used for complex number crunching, and to create "what if" scenarios for forward planning.
Typical OLAP applications include business reporting for sales, marketing, management reporting, business process management, budgeting and forecasting, financial reporting, and more.

The analysis is performed as follow:

The data is collected from multiple sources

↓

The data is stored in the data warehouse

↓

The data is sorted and organized in different data cubes and categorized by dimensions

OLAP vs. Data Warehouse-

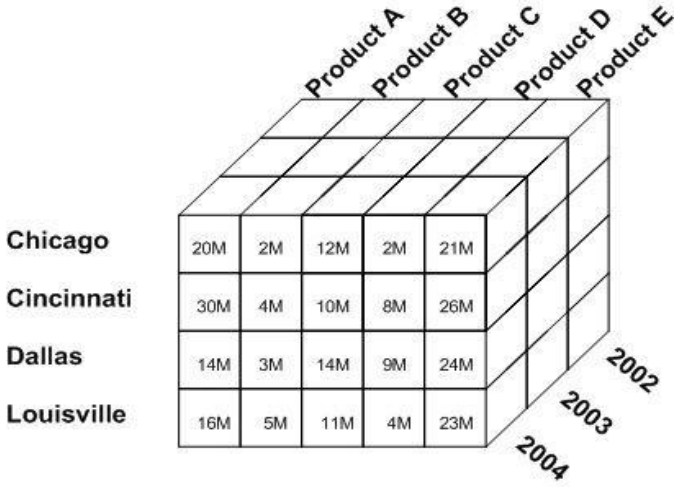
OLAP and data warehouses are two different things. However, OLAP can be used to transform the data from a data warehouse into strategic information. A data warehouse stores and manages data, typically in relational databases. These could be extremely large databases with enormous amounts of data.
OLAP data, on the other hand, is stored in a multidimensional database. In a multidimensional database, each data attribute (such as product, region, time period, etc) is considered a separate "dimension". OLAP tools can be used to extract data from the intersections of such dimensions. For example, all products sold in the Northeast region during the last quarter.

OLAP CUBE:

An OLAP cube is a data structure that overcomes the limitations of relational databases by providing rapid analysis of data. An OLAP cube is typically a multi-dimensional array of data. If we visualize a three-dimensional cube, we could have products along the x axis, regions along

the y axis, and time period along the z axis (as shown in the figure given below). So an OLAP tool could be used to summarize sales data by product, region, and time period. However, OLAP cubes are not restricted to three dimensions. An OLAP cube could have any number of dimensions. In these cases, such a cube is sometimes referred to as a hypercube. The sides of an OLAP cube are not necessarily of equal length – OLAP cubes are not cubes in the strictly mathematical sense. However, the term is a commonly used one when working with OLAP.

OLAP cubes can display and sum large amounts of data while also providing users with searchable access to any data points so that the data can be rolled up, sliced, and diced as needed to handle the widest variety of questions that are relevant to a user's area of interest.



CUBE OPERATOR:

The **CUBE** operator computes a union of GROUP BY's on every subset of the specified attribute types. Its result set represents a multidimensional cube based on the source table. In other words, it allows us to summarize the data based on every permutation of the columns passed to the CUBE operator. Generally, if we have 'n' number of columns listed in the CUBE, the statement will create 2^(n) subtotal combinations.

Syntax-

```
SELECT
        col1, col2, AGGREGATE_FUNCTION(c3)
FROM
        Table_name
GROUP BY CUBE(c1,c2);
```

For illustration, create a table named 'inventory':

```
CREATE TABLE inventory (
   warehouse VARCHAR(255),
```

```
   product VARCHAR(255) NOT NULL,
   model VARCHAR(50) NOT NULL,
   quantity INT,
   PRIMARY KEY (warehouse,product,model)
);


-------------------------------- Insert Values ---------------------------------

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Jose', 'iPhone','6s',100);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Francisco', 'iPhone','6s',50);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Jose','iPhone','7',50);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Francisco', 'iPhone','7',10);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Jose','iPhone','X',150);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Francisco', 'iPhone','X',200);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Jose','Samsung','Galaxy S',200);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Francisco','Samsung','Galaxy S',200);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Francisco','Samsung','Note 8',100);

INSERT INTO inventory(warehouse, product, model, quantity)
VALUES('San Jose','Samsung','Note 8',150);
```

CUBE with single column:

Example: To find out total sales of all warehouses, the traditional query using Group by clause (shown below) will return total sales of every warehouse individually.

```
SELECT warehouse, SUM(quantity) AS Total_Qty
FROM inventory
```

GROUP BY warehouse;

To find out sales from all warehouses, we will use CUBE operator. Using CUBE extension adds a total inventory row with a null value in the warehouse column.

```
SELECT warehouse, SUM(quantity) AS Total_Qty
FROM inventory
GROUP BY CUBE(warehouse)
ORDER BY warehouse;
```

CUBE with multiple columns:

Example: Retrieve total inventory by warehouse and product.

Traditional query-

```
SELECT warehouse, product, SUM(quantity)
FROM inventory
GROUP BY warehouse,product
ORDER BY warehouse,  product;
```

Query using CUBE operator-

```
SELECT warehouse, product, SUM(quantity)
FROM inventory
GROUP BY CUBE(warehouse,product)
ORDER BY warehouse, product;
```

OLAP Operations:

1. **ROLL UP**
   The ROLLUP is an extension of the GROUP BY clause. The ROLLUP option allows us to include extra rows that represent the subtotals, which are commonly referred to as super-aggregate rows, along with the grand total row. By using the ROLLUP option, one can use a single query to generate multiple grouping sets. The ROLLUP assumes a hierarchy among the input columns. For example, if the input column is (c1,c2), the hierarchy c1 > c2. The ROLLUP generates all grouping sets that make sense considering this hierarchy.

```
SELECT
        col1,col2,AGGREGATE_FUNCTION(c3)
FROM
        Table_name
GROUP BY ROLLUP(c1,c2);
```

Example: To retrieve total products in all warehouses

```
SELECT warehouse, SUM(quantity)
FROM inventory
GROUP BY ROLLUP(warehouse);
```

2. **Drill Down**

Drill down refers to zooming in to more detailed data i.e. viewing it at a level of increased detail.
Example: Retrieve total quantity of iPhone sold per model.

```
SELECT model, SUM(quantity)
FROM inventory
WHERE product = 'iPhone'
GROUP BY model;
```

3. **Slicing**

It selects a single dimension from the OLAP cube which results in a new sub-cube creation. In the following example, slice is performed on warehouse = ‘San Francisco’.

```
SELECT product, model, SUM(quantity)
FROM inventory
WHERE warehouse='San Francisco'
GROUP BY product,model
```

4. **Dicing**

It selects a sub-cube from the OLAP cube by selecting two or more dimensions. In the cube given in the overview section, a sub-cube is selected by selecting following dimensions with criteria:

- Warehouse= “San Jose”
- product= “iPhone” or “Samsung”
- Item = “6s” or “Note 8”

```
SELECT warehouse, product, model, quantity
FROM Inventory
WHERE warehouse='San Jose' AND product IN ('iPhone','Samsung') AND model IN('6s','Note 8');
```

5. **PIVOT**

PIVOT clause allows you to write a cross-tabulation. This means that you can aggregate your results and rotate rows into columns.
Syntax-

```
SELECT first_column AS <first_column_alias>, [pivot_value1], [pivot_value2], ... [pivot_value_n]
FROM (<source_table>) AS <source_table_alias>
PIVOT
(
        aggregate_function(<aggregate_column>)
         FOR <pivot_column>
         IN ([pivot_value1], [pivot_value2], ... [pivot_value_n])
) AS <pivot_table_alias>;
```

Example:

```
CREATE TABLE departments
( dept_id INT NOT NULL,
  dept_name VARCHAR(50) NOT NULL,
  CONSTRAINT departments_pk PRIMARY KEY (dept_id)
);

CREATE TABLE employees
( employee_number INT NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  first_name VARCHAR(50) NOT NULL,
  salary INT,
  dept_id INT,
  CONSTRAINT employees_pk PRIMARY KEY (employee_number)
);

                    INSERTION
INSERT INTO departments
(dept_id, dept_name)
VALUES (30, 'Accounting');

INSERT INTO departments
(dept_id, dept_name)
VALUES (45, 'Sales');

INSERT INTO employees
(employee_number, last_name, first_name, salary, dept_id)
VALUES (12009, 'Sutherland', 'Barbara', 54000, 45);

INSERT INTO employees
```

```
(employee_number, last_name, first_name, salary, dept_id)
VALUES (34974, 'Yates', 'Fred', 80000, 45);

INSERT INTO employees
(employee_number, last_name, first_name, salary, dept_id)
VALUES (34987, 'Erickson', 'Neil', 42000, 45);

INSERT INTO employees
(employee_number, last_name, first_name, salary, dept_id)
VALUES (45001, 'Parker', 'Salary', 57500, 30);

INSERT INTO employees
(employee_number, last_name, first_name, salary, dept_id)
VALUES (75623, 'Gates', 'Steve', 65000, 30);
```

To create a pivot table to display the total salary for *dept_id* 30 and *dept_id* 45-

```
SELECT 'TotalSalary' AS TotalSalaryByDept, [30], [45]
FROM
(
SELECT dept_id, salary FROM employees
) AS SourceTable
PIVOT
(
 SUM(salary)
 FOR dept_id IN ([30], [45])
) AS PivotTable;
```

Exercise:-

The ROLLUP operator is used with the GROUP BY clause. It is used to create subtotals and grand totals for a set of columns. The summarized amounts are created based on the columns passed to the ROLLUP operator.

The CUBE operators, like the ROLLUP operator produces subtotals and grand totals as well. But unlike the ROLLUP operator it produces subtotals and grand totals for every permutation of the columns provided to the CUBE operator.

Create table PurchaseItem

PurchaseId, Supplier, PurchaseType, PurchaseAmt, PurchaseDate Insert

some sample data and perform below queries.

1. Calculate subtotals for all the different PurchaseTypes and then at the end calculate a GrandTotal for all the PurchaseTypes combined using Rollup.
2. Calculate subtotals by ProductType by month, as well as Monthly Total amount at the end of every month also calculate Grant Total amount of all product sales at the end using RollUp.
3. Calculate subtotals for each PurchaseType by month, followed by the Grand Total for each PurchaseType. Once each PurchaseType is calculated by month with their Grand Total amounts calculate “Grand Total” amount for all purchases. Lastly calculate monthly subtotals.