ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN

VIETNAM - KOREA UNIVERSITY OF INFORMATION AND COMMUNICATION TECHNOLOGY

한-베정보통신기술대학교

Nhân bản – Phụng sự – Khai phóng

Chapter 5
Physical Data Warehouse Design

Physical Data Warehouse Design

5.1 Physical modeling of Datawarehouse

Overview of the three classic techniques for improving data warehouse performance:

- Materialized views

- Indexing

- Partitioning

Physical Data Warehouse Design

Materialized views: is a view that is physically stored in a database.

- A typical problem of materialized views is updating since all modifications to the underlying base tables must be propagated into the view.

- An important problem in designing a data warehouse is the selection of materialized views.

- Query rewriting: once the views to be materialized have been defined, the queries addressed to a data warehouse must be rewritten in order to best exploit such views to improve query response time.

Physical Data Warehouse Design

Indexing: two common types of indexes for data warehouses are bitmap indexes and join indexes.

- **Bitmap indexes** are a special kind of index, particularly useful for columns with a low number of distinct values (i.e., low cardinality attributes), although several compression techniques eliminate this limitation.

- **Join indexes** materialize a relational join between two tables by keeping pairs of row identifiers that participate in the join.

Physical Data Warehouse Design

Partitioning or fragmentation: is a mechanism frequently used in relational databases to reduce the execution time of queries.

- Vertical partitioning: splits the attributes of a table into groups that can be independently stored.

- Horizontal partitioning: divides the records of a table into groups according to a particular criterion.

5.2 Materialized Views

A materialized view is a view that is physically stored in the database.

The process of updating a materialized view in response to changes in the base relations is called view maintenance.

Compute changes to the view caused by changes in the underlying relations without recomputing the entire view from scratch. This is called incremental view maintenance.

Physical Data Warehouse Design

The view maintenance problem can be analyzed through four dimensions:

• ***Information***: Refers to the information available for view maintenance, like integrity constraints, keys, access to base relations, and so on.

• ***Modification***: Refers to the kinds of modifications that can be handled by the maintenance algorithm, namely, insertions, deletions, and updates; the latter are usually treated as deletions followed by insertions.

• ***Language***: Refers to the language used to define the view, most often SQL. Aggregation and recursion are also issues in this dimension.

• ***Instance***: Refers to whether or not the algorithm works for every instance of the database or for a subset of all instances.

Physical Data Warehouse Design

Two main classes of algorithms for view maintenance have been studied in the database literature:

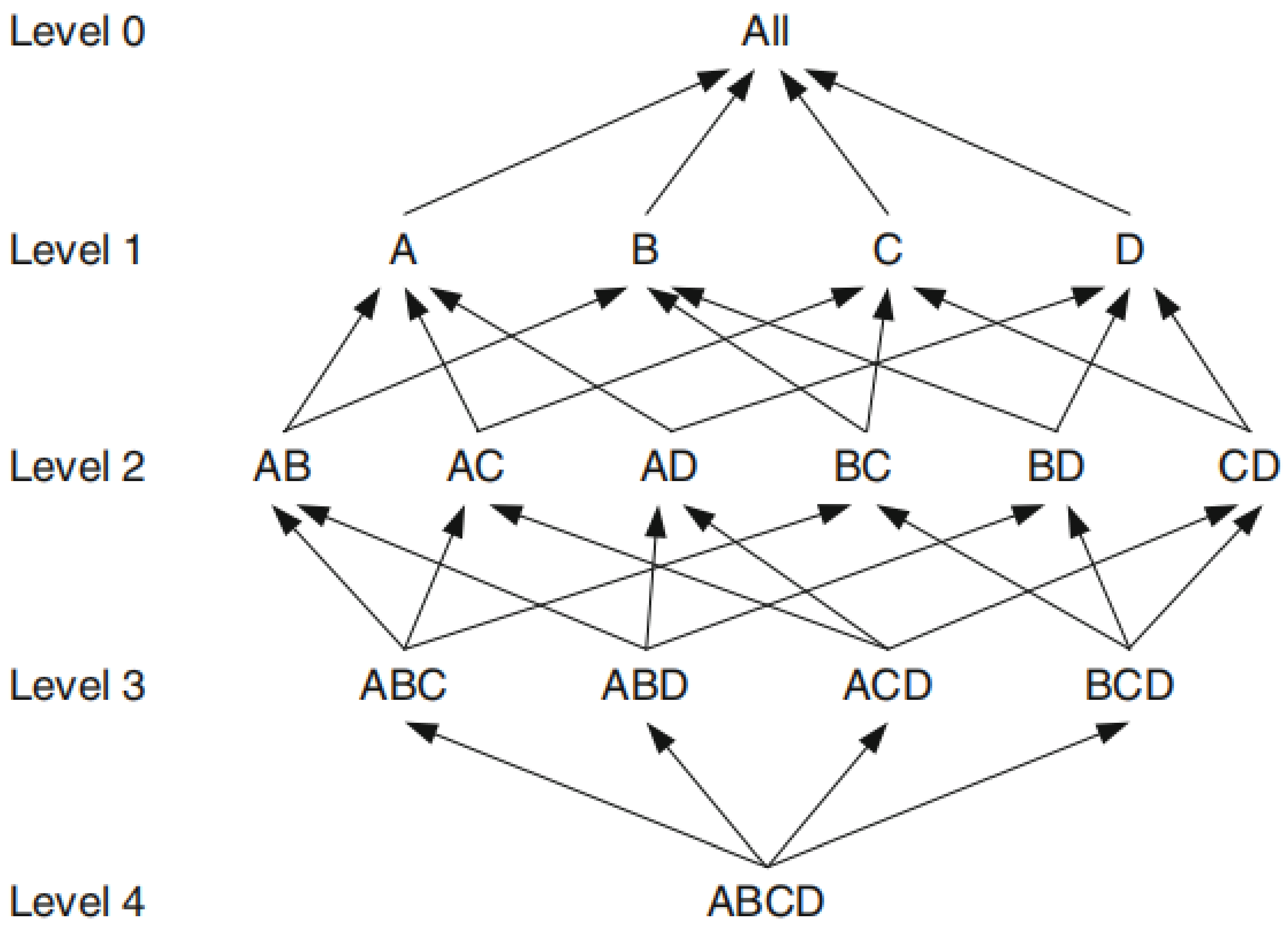
- Algorithms using full information, which means the views and the base relations.

- Algorithms using partial information, namely, the materialized views and the key constraints.

5.3 Data Cube Maintenance

In data warehouses, materialized views that include aggregate functions are called summary tables.

5.4 Computation of a Data Cube



Physical Data Warehouse Design

The simplest optimizations for computing the cube lattice are:

• **Smallest-parent:** Computes each view from the smallest previously computed one.

• **Cache-results:** Caches in memory an aggregation from which other ones can be computed.

• **Amortize-scans:** Computes in memory as many aggregations as possible, reducing the amount of table scans.

• **Share-sorts:** Applies only to methods based on sorting and aims at sharing costs between several aggregations.

• **Share-partitions:** These are specific to algorithms based on hashing. When the hash table is too large to fit in main memory, data are partitioned and aggregation is performed for each partition that fits in memory. The partitioning cost can be shared across multiple aggregations.

Physical Data Warehouse Design

PipeSort Algorithm

INPUT: A search lattice with the A() and S() edges costs

OUTPUT: An evaluation plan to compute all nodes in the search lattice

```
For level k = 0 to level N - 1
Generate-Plan(k + 1 → k);
For each node i in level k + 1
Fix the sort order of i as the order of the level k node
connected to i by an A() edge;
Generate-Plan(k + 1 → k);
Create k additional copies of each level k + 1 node;
Connect each copy node to the same set of level k nodes as the
original node;
Assign cost A(eij ) to edges eij from the original nodes and cost
S(eij)
to edges from the copy nodes;
Find the minimum cost matching on the transformed level k + 1 with
level k;
```

Physical Data Warehouse Design

```
View Materialization Benefit Algorithm
INPUT: A lattice $L$, each view node labeled with its expected number of rows
    A node $v$, not yet selected to materialize
    A set $S$ containing the nodes already selected to materialize
OUTPUT: The benefit of materializing $v$ given $S$
BEGIN
    For each view $w \preceq v$, $w \notin S$, $Bw$ is computed by
        Let $u$ be the view of least cost in $S$ such that $w \preceq u$
        If $C(v) < C(u)$, $Bw = C(u) - C(v)$, otherwise $Bw = 0$
    $B(v, S) = \sum_{w \preceq v} Bw$
END
```

Physical Data Warehouse Design

```
View Selection Algorithm
INPUT: A lattice $L$, each view node $v$ labeled with its expected number of rows
    The number of views to materialize, $k$
OUTPUT: The set of views to materialize
BEGIN
    $S = \{$The bottom view in $L\}$
    FOR $i = 1$ TO $k$ DO
        Select a view $v$ not in $S$ such that $B(v, S)$ is maximized
        $S = S \cup \{v\}$
        END DO
    $S$ is the selection of views to materialize
END
```

5.5 Indexes for Data Warehouses

A popular way to speed data access is known as indexing. An index provides a quick way to locate data of interest.

Almost all the queries asking for data that satisfy a certain condition are answered with the help of some index.

As an example, consider the following SQL query:

```
SELECT *
FROM Employee
WHERE EmployeeKey = 1234
```

Indexes for Data Warehouses

A popular way to speed data access is known as indexing. An index provides a quick way to locate data of interest.

Almost all the queries asking for data that satisfy a certain condition are answered with the help of some index.

As an example, consider the following SQL query:

```
SELECT *
FROM Employee
WHERE EmployeeKey = 1234
```

The most popular indexing technique in relational databases is the B+- tree.

Physical Data Warehouse Design

Some indexing requirements for a data warehouse system are as follows:

- Symmetric partial match queries
- Indexing at multiple levels of aggregation
- Efficient batch update
- Sparse data

5.6 Evaluation of Star Queries

Queries over star schemas are called star queries since they make use of the star schema structure, joining the fact table with the dimension tables.

```
SELECT C.CustomerName, P.ProductName,
SUM(S.SalesAmount)
FROM Sales S, Customer C, Product P
WHERE S.CustomerKey = C.CustomerKey AND
S.ProductKey = P.ProductKey AND P.Discontinued =’Yes'
GROUP BY C.CustomerName, P.ProductName
```

5.7 Data Warehouse Partitioning

In a database, partitioning or fragmentation divides a table into smaller data sets (each one called a partition) to better support the management and processing of very large volumes of data.

Partitioning can be applied to tables as well as to indexes.

Physical Data Warehouse Design

There are two ways of partitioning a table: vertically and horizontally.

Vertical partitioning splits the attributes of a table into groups that can be independently stored.

Horizontal partitioning divides a table into smaller tables that have the same structure than the full table but fewer records.

we study partitioning, followed by a description of how the three types of multidimensional data representation introduced in Chap. 5, namely, ROLAP, MOLAP, and HOLAP, are implemented in Analysis Services.

Queries in Partitioned Databases

There are two classic techniques of partitioning related to query evaluation.

Partition pruning is the typical way of improving query performance using partitioning, often producing performance enhancements of several orders of magnitude.

The execution time of **joins** can also be enhanced by using partitioning.

Partitioning Strategies

There are three most common partitioning strategies in database systems: range partitioning, hash partitioning, and list partitioning.

Physical Data Warehouse Design

The most usual type of partitioning is **range partitioning**, which maps records to partitions based on ranges of values of the partitioning key.

Hash partitioning maps records to partitions based on a hashing algorithm applied to the partitioning key.

List partitioning enables to explicitly control how rows are mapped to partitions specifying a list of values for the partitioning key.

5.8 Physical Design in SQL Server and Analysis Services

Indexed Views

Basically, an indexed view consists in the creation of a unique clustered index on a view,thus precomputing and materializing such view.

```
CREATE VIEW EmployeeSales WITH SCHEMABINDING AS (
SELECT EmployeeKey, SUM(UnitPrice * OrderQty * Discount)
AS TotalAmount, COUNT(*) AS SalesCount
FROM Sales
GROUP BY EmployeeKey )
CREATE UNIQUE CLUSTERED INDEX CI EmployeeSales ON
EmployeeSales (EmployeeKey)
```

Physical Data Warehouse Design

An indexed view can be used in two ways: when a query explicitly references the indexed view and when the view is not referenced in a query but the query optimizer determines that the view can be used to generate a lower-cost query plan.

```
SELECT EmployeeKey, EmployeeName, . . .
FROM Employee, EmployeeSales WITH (NOEXPAND)
WHERE . . .
```

Physical Data Warehouse Design

Partition-Aligned Indexed Views

If a partitioned table is created in SQL Server and indexed views are built on this table, SQL Server automatically partitions the indexed view by using the same partition scheme as the table. An indexed view built in this way is called a partition-aligned indexed view.

```
CREATE PARTITION FUNCTION [PartByYear] (INT)
AS RANGE LEFT FOR VALUES (184, 549, 730);
```

Once the partition function has been defined, the partition scheme is created as follows:

```
CREATE PARTITION SCHEME [SalesPartScheme]
AS PARTITION [PartByYear] ALL to ( [PRIMARY] );
```

Physical Data Warehouse Design

The Sales fact table is created as a partitioned table as follows:

```
CREATE TABLE Sales (CustomerKey INT, EmployeeKey INT,
OrderDateKey INT, . . . ) ON SalesPartScheme(OrderDateKey)

CREATE VIEW SalesByDateProdEmp WITH SCHEMABINDING AS (
SELECT OrderDateKey, ProductKey, EmployeeKey, COUNT(*) AS Cnt,
SUM(SalesAmount) AS SalesAmount
FROM Sales
GROUP BY OrderDateKey, ProductKey, EmployeeKey )

CREATE UNIQUE CLUSTERED INDEX UCI SalesByDateProdEmp
ON SalesByDateProdEmp (OrderDateKey, ProductKey, EmployeeKey)
ON SalesPartScheme(OrderDateKey)
```

Column-Store Indexes

SQL Server provides column-store indexes, which store data by column. In a sense, column-store indexes work like a vertical partitioning commented above and can dramatically enhance performance for certain kinds of queries.

```
CREATE NONCLUSTERED COLUMNSTORE INDEX CSI
Sales2012
ON Sales2012 (DueDateKey, EmployeeKey,
SalesAmount)
```

Partitions in Analysis Services

In Analysis Services, a partition is a container for a portion of the data of a measure group. Defining a partition requires to specify:

• Basic information, like name of the partition, the storage mode, and the processing mode.

• Slicing definition, which is an MDX expression specifying a tuple or a set.

• Aggregation design, which is a collection of aggregation definitions that can be shared across multiple partitions.

Physical Data Warehouse Design

ROLAP Storage

In the ROLAP storage mode, the aggregations of a partition are stored in indexed views in the relational database specified as the data source of the partition.

In the ROLAP storage, the query response time and the processing time are generally slower than with the MOLAP or HOLAP storage modes

ROLAP Storage

- The partition cannot contain measures that use the MIN or MAX aggregate functions.
- Each table in the schema of the ROLAP partition must be used only once.
- All table names in the partition schema must be qualified with the owner name, for example, [dbo].[Customer].
- All tables in the partition schema must have the same owner.
- The source columns of the partition measures must not be nullable.

MOLAP Storage

In the MOLAP storage mode, both the aggregations and a copy of the source data are stored in a multidimensional structure. Such structures are highly optimized to maximize query performance.

HOLAP Storage

The HOLAP storage mode combines features of the previously explained MOLAP and ROLAP modes. Like MOLAP, in HOLAP the aggregations of the partition are stored in a multidimensional data structure.

Physical Data Warehouse Design

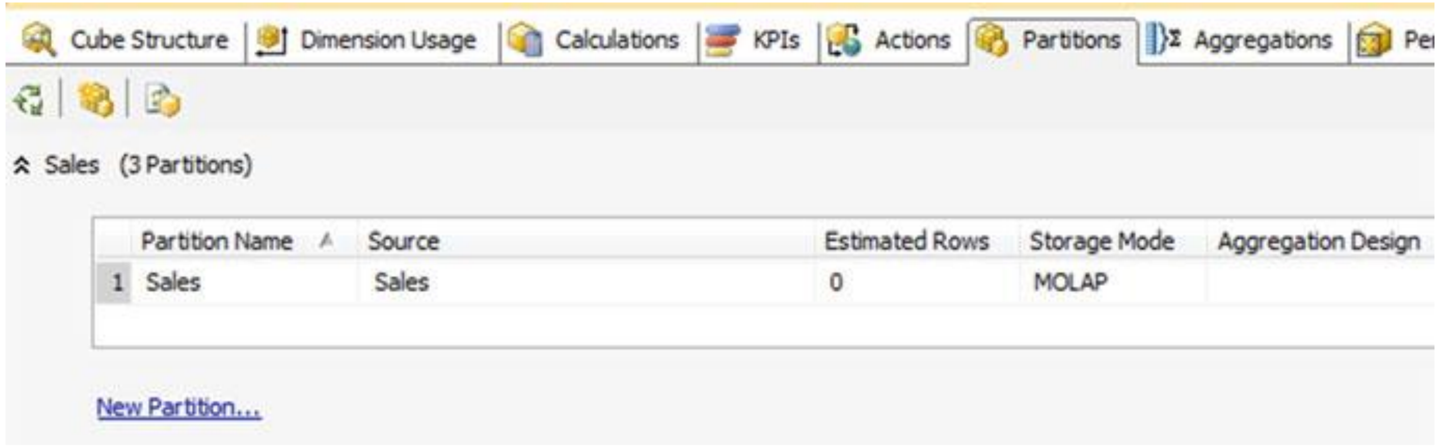
In summary, partitions stored as HOLAP are smaller than the equivalent MOLAP partitions since they do not contain source data.

On the other hand, they can answer faster than ROLAP partitions for queries involving summary data. Thus, this mode tries to capture the best of both worlds.

Defining Partitions in Analysis Services

We show next how MOLAP, ROLAP, and HOLAP partitions over measure groups can be defined in Analysis Services.

Fig. 7.14 Initial partition for the Sales measure group

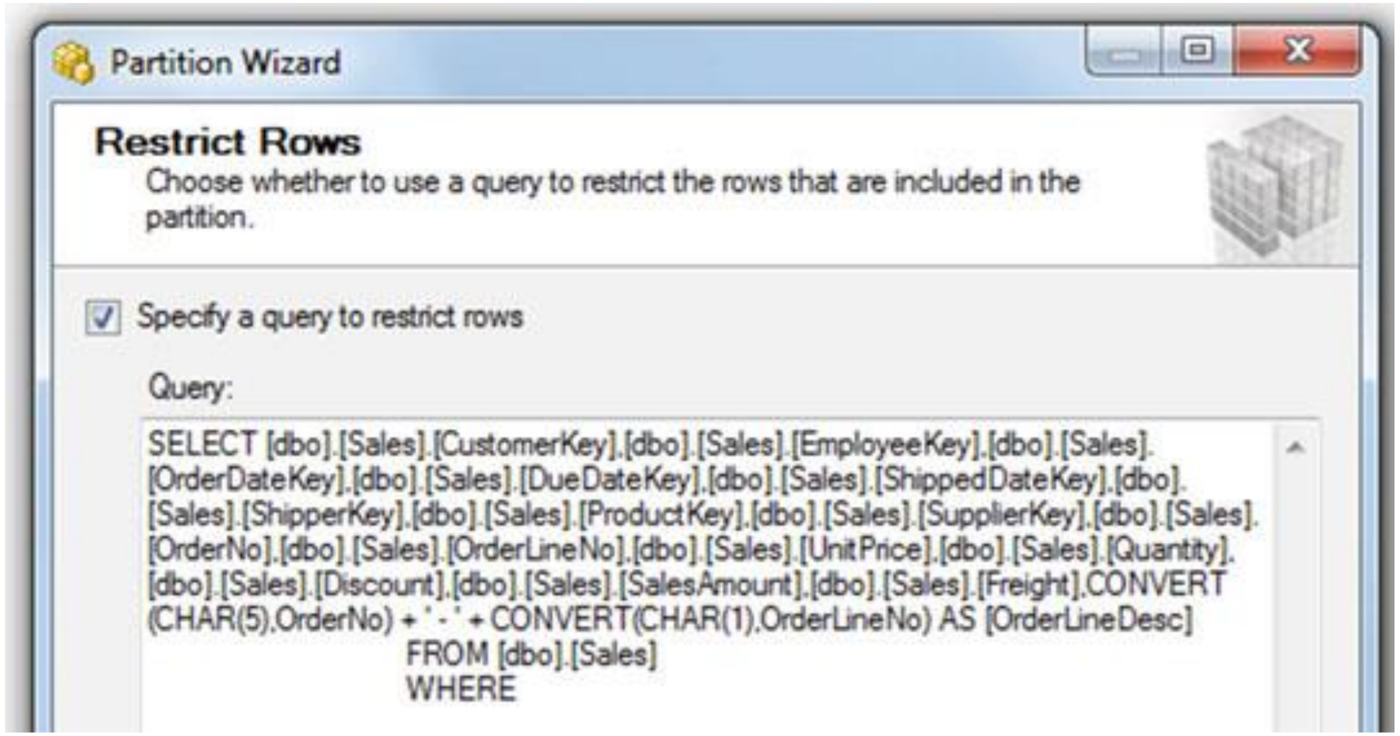


Sales (3 Partitions)

	Partition Name	Source	Estimated Rows	Storage Mode	Aggregation Design
1	Sales	Sales	0	MOLAP	

New Partition...

Fig. 7.15 Template query that defines a partition



Restrict Rows

Choose whether to use a query to restrict the rows that are included in the partition.

☑ Specify a query to restrict rows

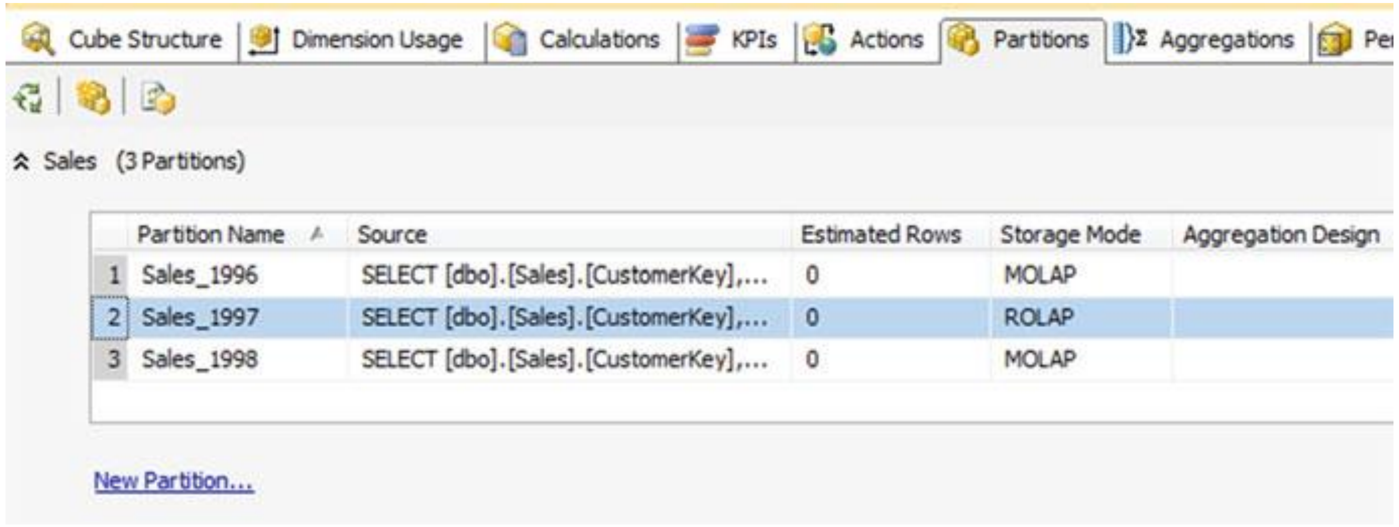
Query:

```
SELECT [dbo].[Sales].[CustomerKey],[dbo].[Sales].[EmployeeKey],[dbo].[Sales].
[OrderDateKey],[dbo].[Sales].[DueDateKey],[dbo].[Sales].[ShippedDateKey],[dbo].
[Sales].[ShipperKey],[dbo].[Sales].[ProductKey],[dbo].[Sales].[SupplierKey],[dbo].[Sales].
[OrderNo],[dbo].[Sales].[OrderLineNo],[dbo].[Sales].[UnitPrice],[dbo].[Sales].[Quantity],
[dbo].[Sales].[Discount],[dbo].[Sales].[SalesAmount],[dbo].[Sales].[Freight],CONVERT
(CHAR(5),OrderNo) + ' - ' + CONVERT(CHAR(1),OrderLineNo) AS [OrderLineDesc]
                    FROM [dbo].[Sales]
                    WHERE
```

Query such as the following one must be addressed to the data warehouse:

```
SELECT MIN(TimeKey), MAX(TimeKey)
FROM Time
WHERE Date >= '1997-01-01' AND Date <= '1997-12-31'
```

Fig. 7.16 Final partitions for the Sales measure group

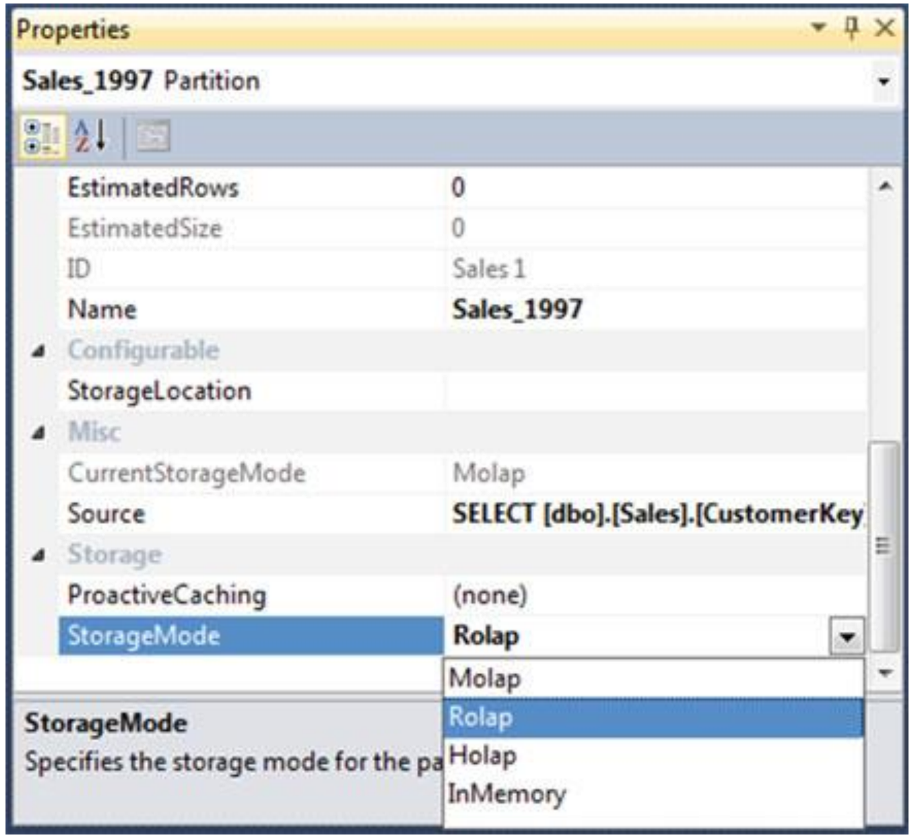


Sales (3 Partitions)

	Partition Name	Source	Estimated Rows	Storage Mode	Aggregation Design
1	Sales_1996	SELECT [dbo].[Sales].[CustomerKey],...	0	MOLAP	
2	Sales_1997	SELECT [dbo].[Sales].[CustomerKey],...	0	ROLAP	
3	Sales_1998	SELECT [dbo].[Sales].[CustomerKey],...	0	MOLAP	

New Partition...

Fig. 7.17 Storage mode for the Sales 1997 partition


Properties

Sales_1997 Partition

EstimatedRows	0
EstimatedSize	0
ID	Sales 1
Name	Sales_1997
Configurable	
StorageLocation	
Misc	
CurrentStorageMode	Molap
Source	SELECT [dbo].[Sales].[CustomerKey]
Storage	
ProactiveCaching	(none)
StorageMode	Rolap

StorageMode

Specifies the storage mode for the pa

5.9 Query Performance in Analysis Services

The first step must be to optimize cube and measure group design. For this, many of the issues studied in this book apply.

For example, it is suggested to use cascading attribute relationships, like Day → Month → Quarter → Year, and define user hierarchies of related attributes within each dimension.

These are called natural hierarchies.

5.9 Query Performance in Analysis Services

The first step must be to optimize cube and measure group design. For this, many of the issues studied in this book apply. For example, it is suggested to use cascading attribute relationships, like Day → Month → Quarter → Year, and define user hierarchies of related attributes within each dimension. These are called natural hierarchies.

Aggregations are also used by Analysis Services to enhance query performance.

In summary, we must avoid designing a large number of aggregations since they may reduce query performance.

Analysis Services uses the Aggregation Usage
property to determine which attributes it should consider for aggregation.
This property can take one of four values:

full (every aggregation for the cube must include this attribute),
none (no aggregation uses the attribute),
unrestricted (the attribute must be evaluated),
default (a rule is applied to determine if the attribute must be used).

We can also optimize performance by writing efficient MDX queries and expressions.

5.10 Query Performance in Mondrian

Aggregate Tables

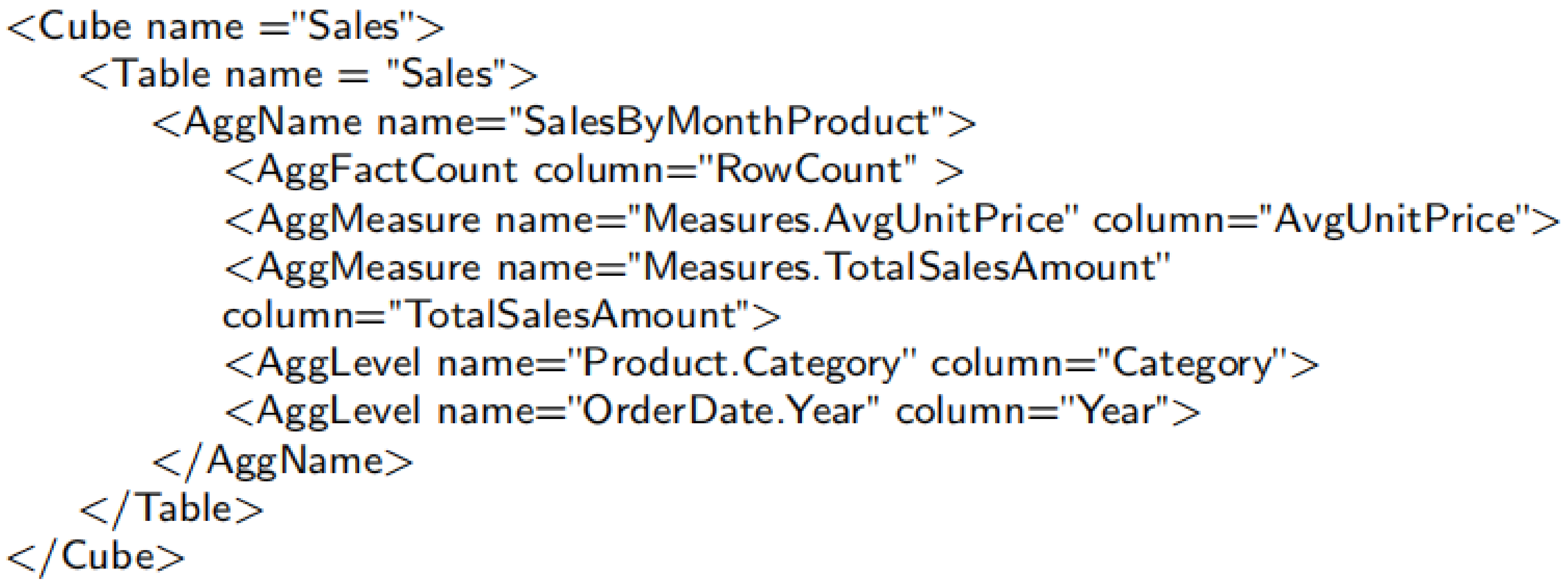
In Mondrian, materialized views and summary tables studied in this chapter are called aggregate tables.

Physically, aggregate tables are created in the database and populated during the ETL process.

Physical Data Warehouse Design

The table will have the following

columns: Category, Year, RowCount, AvgUnitPrice, and TotalSalesAmount.



```
<Cube name ="Sales">
    <Table name = "Sales">
        <AggName name="SalesByMonthProduct">
            <AggFactCount column="RowCount" >
            <AggMeasure name="Measures.AvgUnitPrice" column="AvgUnitPrice">
            <AggMeasure name="Measures.TotalSalesAmount"
            column="TotalSalesAmount">
            <AggLevel name="Product.Category" column="Category">
            <AggLevel name="OrderDate.Year" column="Year">
        </AggName>
    </Table>
</Cube>
```

Caching

Another feature provided by Mondrian to speed up query performance is caching data in main memory, to avoid accessing the database to retrieve schemas, dimension members, and facts.

Mondrian provides three different kinds of caches:

- **The schema cache**, which keeps schemas in memory to avoid reading them each time a cube is loaded.

- **The member cache**, which stores dimension members in memory. The member cache must also be synchronized with the underlying data.

- **The segment cache**, which holds data from the fact table (usually the largest table in a warehouse) and contains aggregated data, reducing the number of calculations to perform.

Review Questions

7.1 What is the objective of physical data warehouse design? Specify different techniques that are used to achieve such objective.
7.2 Discuss advantages and disadvantages of using materialized views.
7.3 What is view maintenance? What is incremental view maintenance?
7.4 Discuss the kinds of algorithms for incremental view maintenance, that is, using full and partial information.
7.5 Define self-maintainable aggregate functions. What is a self_x0002_maintainable view?
7.6 Explain briefly the main idea of the summary-delta algorithm for data cube maintenance.
7.7 How is data cube computation optimized? What are the kinds of optimizations that algorithms are based on?
7.8 Explain the idea of the PipeSort algorithm.

Review Questions

7.9 How can we estimate the size of a data cube?

7.10 Explain the algorithm for selecting a set of views to materialize. Discuss

its limitations. How can they be overridden?

7.11 Compare B-tree+ indexes, hash indexes, bitmap indexes, and join

indexes with respect to their use in databases and data warehouses.

7.12 How do we use bitmap indexes for range queries?

7.13 Explain run length encoding.

7.14 Describe a typical indexing scheme in a star and snowflake schemas.

7.15 How are bitmap indexes used during query processing?

Review Questions

7.16 How do join indexes work in query processing? For which kinds of queries are they efficient? For which kinds of queries are they not efficient?

7.17 What is partitioning? Which kinds of partitioning schemes do you know?

7.18 What are the main advantages and disadvantages of partitioning?

7.19 Discuss the characteristics of storage modes in Analysis Services.

7.20 How do indexed views compare with materialized views?

Exercises

7.1 In the Northwind database, consider the relations

computed from the full outer join of tables Employee and Orders, where Name is obtained by concatenating FirstName and LastName.

Employee(EmplID, FirstName, LastName, Title, . . .)
Orders(OrderID, CustID, EmpID, OrderDate, . . .).

Consider further a view

EmpOrders(EmpID, Name, OrderID, OrderDate)

Exercises

7.2 Consider a relation Connected(CityFrom, CityTo, Distance), which indicates pairs of cities that are directly connected and the distance between them, and a view OneStop(CityFrom, CityTo), which computes the pairs of cities (c1, c2) such that c2 can be reached from c1 passing through exactly one intermediate stop.

Answer the same questions as those of the previous exercise.

Exercises

7.3 Consider the following tables:

Store(StoreID, City, State, Manager)
Order(OrderID, StoreID, Date)
OrderLine(OrderID, LineNo, ProductID, Quantity, Price)
Product(ProductID, ProductName, Category, Supplier)
Part(PartID, PartName, ProductID, Quantity)

and the following views:

- ParisManagers(Manager) that contains managers of stores located in Paris.
- OrderProducts(OrderID, ProductCount) that contains the number of products for each order.
- OrderSuppliers(OrderID, SupplierCount) that contains the number of suppliers for each order.
- OrderAmount(OrderID, StoreID, Date, Amount) which adds to the table Order an additional column that contains the total amount of each order.
- StoreOrders(StoreID, OrderCount) that contains the number of orders for each store.
- ProductPart(ProductID, ProductName, PartID, PartName) that is obtained from the full outer join of tables Product and Part.

Exercises

7.3 (tt)

Define the above views in SQL. For each of these views, determine whether the view is self-maintainable with respect to insertions and deletions. Give examples illustrating the cases that are not self maintainable.

Enjoy the Course…!