THE UNIVERSITY OF QUEENSLAND
AUSTRALIA
CREATE CHANGE

Advanced Database Systems (INFS3200)

Lecture 6: Online Analytical Processing

Lecturer: Prof ShaziaSadiq
School of Electrical Engineering and Computer Science (EECS)
Faculty of Engineering, Architecture and Information Technology
The University of Queensland

Topics

OLAP Operations

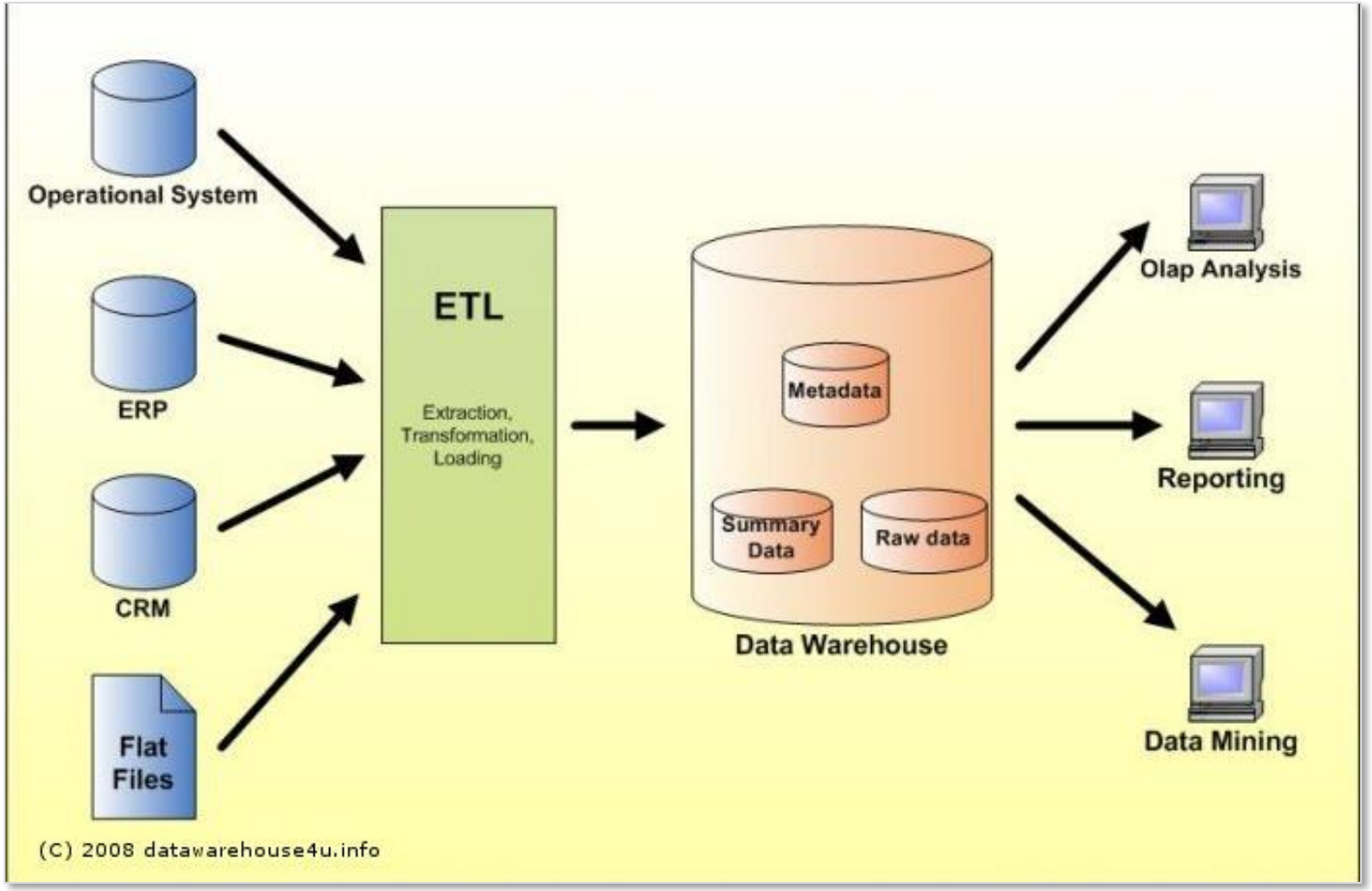
DW Performance for OLAP

- Models
- Indexes

DW Maintenance for OLAP

- View Materialisation
- Data Lineage

Data Warehouse Overview - Revision



OLTP (Online Transaction Processing) vs. **OLAP** (Online Analytical Processing)

OLTP system is a database system used to record current transactional operations.

OLAP database stores historical data that has been collected from OLTP databases

- OLTP is an online **transaction** system whereas, OLAP is an online **data retrieval and analysis** system.
- **Transactional data** is the source of OLTP, whereas different OLTP databases are the source of OLAP.
- OLTP's main operations are **insert, update** and **delete** whereas, OLAP's main operation is to **extract multidimensional data for analysis**.
- OLTP has **short but frequent** transactions whereas, OLAP has **long and less frequent** transaction.
- Processing time for the OLAP's transaction is more as compared to OLTP.
- OLAPs queries are more **complex** with respect OLTPs.

DW Model - The Fact Table

The core of a data warehouse is a fact table

The facts are the values for the object of interest

- A fact about that data entity
- Raw data to be aggregated
- There are lots of instances of these facts

Associated with each fact is a key that is used for identifying, for example, which day, which product and which store.

A **fact** can be defined by a proposition which can be read as a complete sentence.

…facts vs dimensions

DW Model - Dimensions

Each key is a dimension – the example has three

Dimensions can have hierarchical organization

- Days grouped into weeks, months, quarters, years
- Product groups **aggregated hierarchically**
  - ✓Milk ➔ dairy ➔ perishable ➔ food
  - ✓Bread ➔ baked goods ➔ perishable ➔ food
- Stores grouped into regions hierarchically
  - ✓Toowong ➔ West Brisbane ➔Brisbane ➔ QLD ➔ Australia ➔ Oceania

Dimensions organized by dimension tables

The Star Schema

Time-Period

Day	Month	Qtr	Year
9/2/2012	Feb	1	2012
10/2/2012	Feb	1	2012

Region

Store	District	Region
Toowong	North	Brisbane
Kenmore	West	Brisbane

Sales

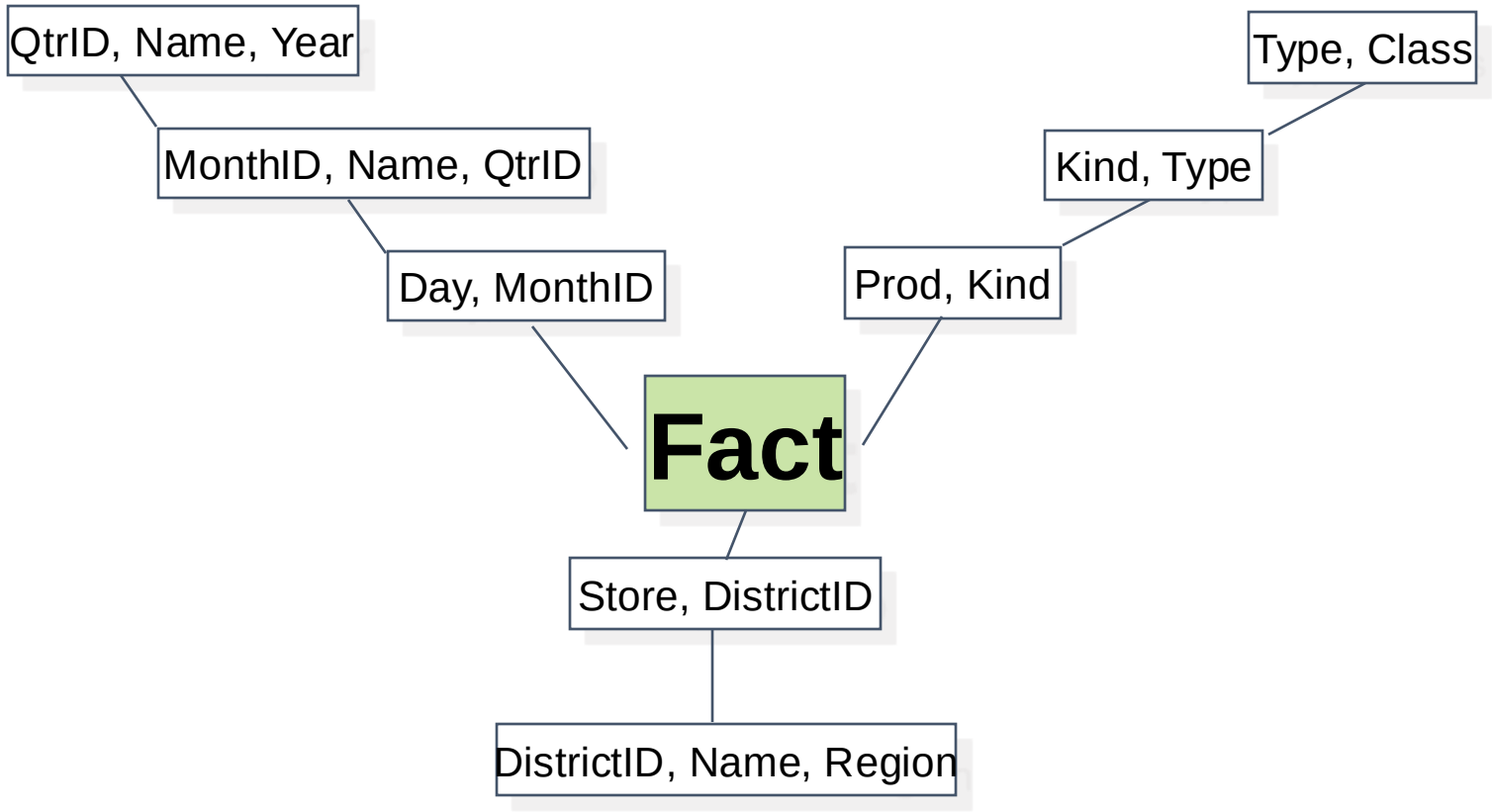
Facts

Product

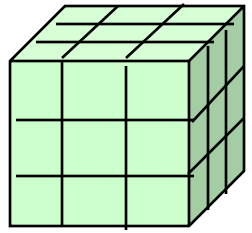
Product	Kind	Type	Class
Milk	Dairy	Perishable	Food
Bread	Bakery	Perishable	Food

A fact table is much larger than dimension tables

The Snowflake Schema



Typical OLAP Operations



Pivoting (cross-tabulation)

- Rotate data cube to show a different orientation of axes

Roll-up

- Move up concept hierarchy, grouping into larger units along a dimension with more generalization

Drill-down

- Disaggregate to a finer-grained view to show more details

Slice and dice

- Perform projection operations on the dimensions

Other operations, such as arithmetic (to get derived values), sorting, selection…

Data Warehouse Performance

DW queries are typically more complex and require extraction from multi-dimensional data models accessing multiple entities over large volumes of data

trans_ts	trans_id	cc_num	gender	state	job	category	merchant	amt
2023-03-18 19:15:43	845989900032	4855352458051797	M	MI	Biomedical engineer	entertainment	Franklin-Stephens	24.719999
2023-03-26 21:39:39	961320600020	4075450219348819318	F	OH	Embryologist, clinical	health_fitness	Henson, Raymond and Taylor	12.670000
2023-03-26 21:39:41	961321200087	4175536993406036	F	FL	Nurse, mental health	health_fitness	Brown PLC	13.690000
2023-03-18 19:15:36	845987900054	349787212051694	F	FL	Embryologist, clinical	food_dining	Wallace, Graves and Roberts	65.720001
2023-03-26 21:39:35	961318700023	3564220071822445	F	NC	Chiropractor	health_fitness	Arnold PLC	76.580002
2023-03-26 21:39:34	961318200059	2261515007621169	F	MO	Town planner	health_fitness	Sosa, Moore and Galloway	19.049999
2023-03-26 21:39:45	961322800051	3574265318793345	F	OH	Educational psychologist	health_fitness	Hoffman Inc	65.529999
2023-03-26 21:39:38	961320100097	213151562643977	M	CT	Occupational psychologist	food_dining	Perez, Taylor and Walker	79.949997
2023-03-26 21:39:38	961319800003	60426963421	M	NY	Scientist, physiological	food_dining	Jimenez, Sullivan and Hamilton	43.500000
2023-03-18 19:15:36	845987900051	349787212051694	F	FL	Embryologist, clinical	food_dining	Wallace, Graves and Roberts	65.720001
2023-03-26 21:39:34	961318200002	2261515007621169	F	MO	Town planner	health_fitness	Sosa, Moore and Galloway	19.049999
2023-03-18 19:15:38	845988500058	4652236635361807574	F	AK	Speech and language therapist	entertainment	Moore-Garcia	30.990000
2023-03-26 21:39:38	961319900044	180035179624760	F	GA	Biochemist, clinical	health_fitness	Mann Ltd	52.990002
2023-03-26 21:39:35	961318900087	4089785118156191	F	TX	Microbiologist	health_fitness	Jimenez, Gay and Johnson	174.460007
2023-03-26 21:39:39	961320500095	3589939377734985	M	GA	Lecturer, higher education	food_dining	Martinez Ltd	16.850000
2023-03-18 19:16:00	845995400079	213177161964548	F	CA	Ergonomist	food_dining	Martin Ltd	37.029999
2023-03-18 19:15:43	845989900007	4855352458051797	M	MI	Biomedical engineer	entertainment	Franklin-Stephens	24.719999
2023-03-18 19:16:00	845995400017	213177161964548	F	CA	Ergonomist	food_dining	Martin Ltd	37.029999
2023-03-26 21:39:43	961322200044	3590488398874666	F	CA	Seismic interpreter	health_fitness	Williams, Ford and Fisher	91.099998
2023-03-18 19:15:40	845989300052	4215676907446113937	M	CA	Operational researcher	entertainment	Williams-Jones	18.510000
2023-03-26 21:39:35	961318700075	3564220071822445	F	NC	Chiropractor	health_fitness	Arnold PLC	76.580002
2023-03-18 19:15:46	845990900076	213124215186694	M	CA	Economist	entertainment	Garcia, Flores and Brooks	36.080002
2023-03-26 21:39:36	961319100085	4609870180373	F	FL	Lobbyist	health_fitness	Gomez, Garcia and Holmes	75.000000
2023-03-18 19:15:49	845992000009	30244932206764	F	PA	Social research officer, government	food_dining	Bates-Carter	102.940002
2023-03-26 21:39:45	961322900012	4955429710170375388	F	TX	Maintenance engineer	health_fitness	Patterson, Lee and Hunter	191.830002

# of columns

10
50
100
500
1,000

# of rows

1,000
100,000
1,000,000
1,000,000,000
100,000,000,000

Optimizing Query Performance

Revision

Approaches used in Column Oriented Databases (e.g. Clickhouse)

- Sparse Primary Index
- Data Skipping Indexes
- Materialized Views
- Projections
- Parallelism

Bitmap Indexing

Join Indexing

Bitmap Indexing

Index on a particular column

- Each value in the column has a bit vector
- The length of the bit vector: # of records in the base table
- The *i*-th bit is set if the *i*-th row of the base table has the value for the indexed column

What are the advantages of Bitmap Index?

Base table

Cust	Region	Type
C1	Asia	Retail
C2	Europe	Dealer
C3	Asia	Dealer
C4	America	Retail
C5	Europe	Dealer

Index on Region

RecID	Asia	Europe	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

Index on Type

RecID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Example of Bitmap Indexing

```
SELECT        *
FROM    Customer
WHERE Region = 'Europe'  AND
              Type= 'Dealer';
```

Bitwise AND:

Europe: 01001

Dealer: 01101

Result: 01001

Customer

	Cust	Region	Type
0	C1	Asia	Retail
1	C2	Europe	Dealer
0	C3	Asia	Dealer
0	C4	America	Retail
1	C5	Europe	Dealer

Index on Region

RecID	Asia	Europe	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

Index on Type

RecID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Bitmap Indexing

Advantages

- Significant reduction in space and I/O
  - ✓E.g., 100 records, varchar(20), 7 unique values: 100*20*8 bits vs 100*7 bits
- Reduce query processing time
  - ✓*Comparison, join and aggregation* operations can be reduced to bit operation (How?)
  - ✓Bit operations are very fast
    - Region = “Asia” AND “Type =“Retail”? 10 operations
    - Bitmap AND for 8 records 1 operation

Disadvantages

- Not suitable for ***high cardinality*** domains Maintaining a bitmap index takes a lot of resources, therefore, bitmap indexes are particularly good for the ***read-only*** tables or tables that have infrequently updates i.e Data Warehouses (otherwise see B-Tree).

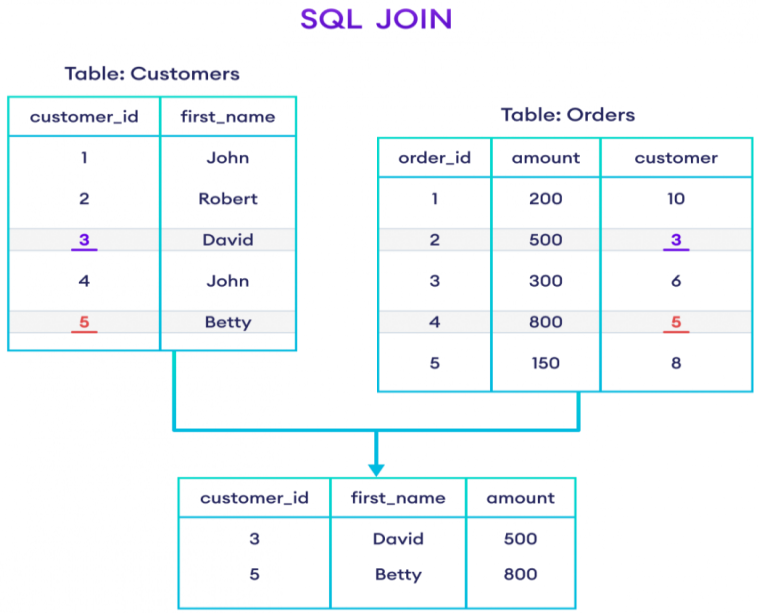
Join Indexing (JI)

Traditional Join index

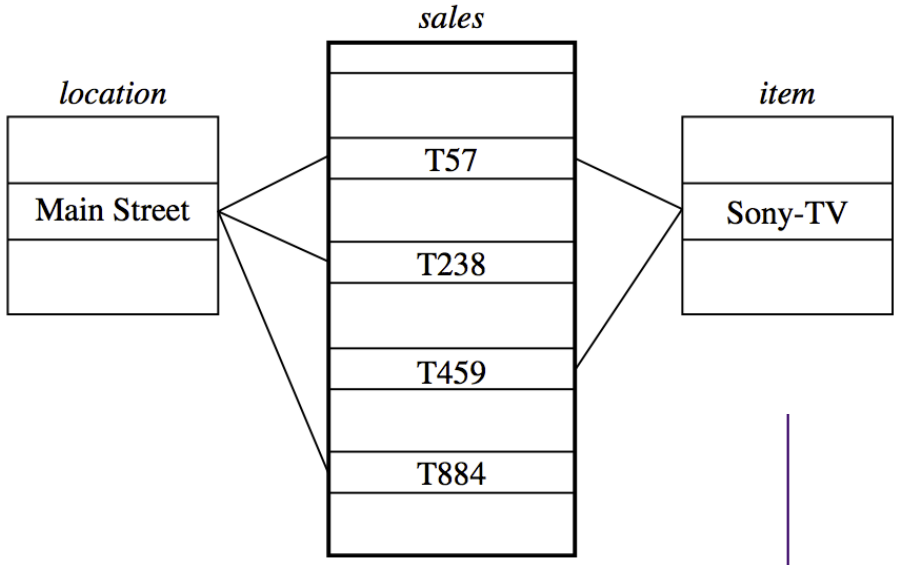
- JI(R-id, S-id) where *R(R-id, …)* ⋈ *S(S-id, …)*
- Map the values to a list of records' IDs
- Materialize relational join in *JI* file and speed up relational join

In a data warehouse, join index relates the values of the dimensions of a star schema to rows in the fact table

- E.g., fact table *Sales* and two dimensions *city* and *product*
  - ✓ A join index on *city* maintains for each distinct city a list of R-IDs of the tuples recording the *Sales* in that city
- Join indices can span multiple dimensions



Join Indexing (JI)



Join index table for *location/sales*

location	*sales_key*
. . .	. . .
Main Street	T57
Main Street	T238
Main Street	T884
. . .	. . .

Join index table for *item/sales*

item	*sales_key*
. . .	. . .
Sony-TV	T57
Sony-TV	T459
. . .	. . .

Join index table linking *location* and *item* to *sales*

location	*item*	*sales_key*
. . .	. . .	. . .
Main Street	Sony-TV	T57
. . .	. . .	. . .

Join index (JI) tables based on the linkages between the *sales* fact table and the *location* and *item* dimension tables

Topics

OLAP Operations

DW Performance for OLAP

- Models
- Indexes

DW Maintenance for OLAP

- View Materialisation
- Data Lineage

DW Maintenance – DW vs Views

- Data warehouses exist as persistent storage instead of being materialized on demand.
- Data warehouses are not just relational views; they are multidimensional views with levels of aggregation.
- Data warehouses can be indexed to optimize performance. Views cannot be indexed independent of the underlying databases.
- Data warehouses characteristically provide specific support of functionality; views cannot.
- Data warehouses provide large amounts of integrated and often temporal data, generally more than is contained in one database, whereas views are an extract of a database.
- Data warehouses bring in data from multiple sources via a complex ETL process that involves cleaning, pruning, and summarization, whereas views are an extract from a database through a predefined query.

View Materialization in DW

Advantages

- OLAP queries are typically aggregate queries, e.g., CUBE and ***GROUP BY***, Drill-Down, etc.
- Pre-aggregation is essential for interactive response time
  - ➢ **Pre-calculate** expensive joins
  - ➢ Speed up online OLAP queries

Disadvantages

- It increases storage cost, if dimensionality (d) is big, a large number **(2^d) of views** would need to be considered for materialization.
- The content of the materialized views must be maintained when the underlying transactional tables are modified.
- Need carefully designed maintenance strategy to trade-off between query performance and accessibility to up-to-date data.

Issues in View Materialization

For 2^d possible views of a d-dimension fact table, what views should we materialize?

For $k \leq 2^d$ number of materialised views, given a query, can we use *k* materialized views to answer all GROUP BY queries?

How frequently should we refresh the materialized views to make them consistent with the underlying tables? (And how can we do this incrementally?)

Cuboid

Recall:

- A ***fact table T*** has *d* dimension attributes $A_1,\ldots,A_d$, and a numeric value attribute *B*
  - We assume the dimensions have no hierarchies
- A GROUP BY query is parameterized with a subset dimensions $G \subseteq \{A_1,\ldots, A_d\}$. Its result, denoted as $\boldsymbol{T_G}$, is referred to as a *cuboid*

Definition: A Cuboid is a denotation of one of the 2^d summarization (GROUP BY) views, which can be used for materialization.

- The *data cube* of a fact table *T* is the set of all results of the 2^d GROUP BY queries (i.e., there are $\boldsymbol{2^d}$ **cuboids** in maximum).

Running Example of a Fact Table *T*

We will use the following fact table as our running example – 8 possible **GROUP BY** queries with **d=3** dimensions:

Example 1.

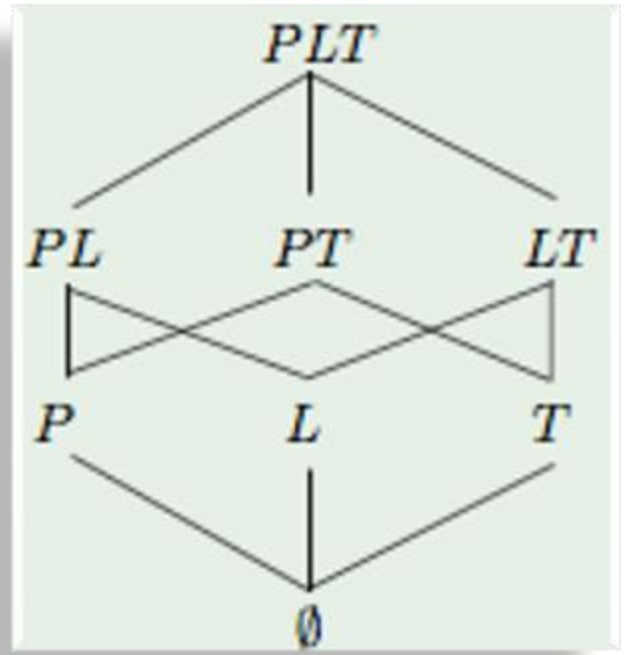
product	location	time	sales
tv	HK	Jan	5
tv	NY	Jan	6
tv	HK	Feb	4
tv	SH	Feb	8
tv	SH	Mar	2
dvd	NY	Jan	3
dvd	SH	Jan	7
dvd	SH	Feb	1
laptop	NY	Feb	4
laptop	SH	Feb	9

Lattice Structured Cuboid

The subset relationships of all dimensions $\{A_1,...,A_d\}$ form up a *lattice structure L,* where a subset *G* is a parent of *G',* if and only if $G' \subset G$ and $|G|=|G'|+1$

```
COMPUTE CUBE sales_cube
AS
SELECT      SUM(*)
FROM      Sales
CUBE BY product, location, time
```

sales_cube (2^3)



PLT (all dimensions):

```
SELECT SUM(sales)
FROM     Sales
GROUP BY product, location, time
```

Sales

Product	Location	Time	sales

Query on Materialized Views (2-1)

How are queries answered from a set of materialized cuboids? What is the cost?

Q1: If there are no materialized views, how to answer a GROUP BY query with dimension set G = {product}?

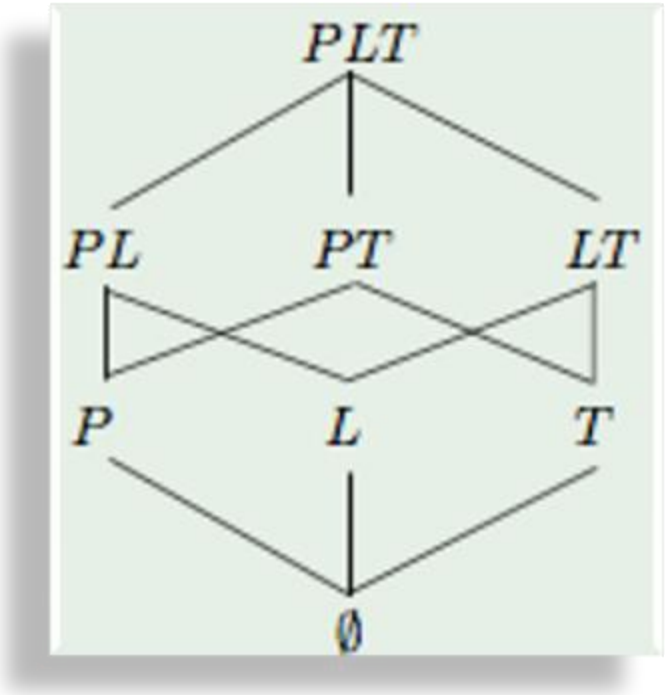
- Sort **fact table *T*** by *G = {product}*
- All records in **fact table T** will be scanned.

Suppose that we have pre-computed the cuboid T_{G1} of G1 = {product, location}. We can now answer a GROUP BY query with dimension set G = {product} directly from T_{G1}, instead of from the **fact table *T***

Q2: What other GROUP BY queries can benefit from a pre-computed T_{G1}?

Query on Materialized Views (2-2)

A materialized view with *G*-cuboid T_G can be used to answer a query with GROUP BY dimension set *Q* **only if *Q*⊆*G***, namely, *G* is an ancestor of *Q* in *Lattice L*



Some Complications

Question: What if the dimensions have hierarchies?

- **Product**: *item* < *brand*
- **Location**: *street* < *city* < *state* < *country*
- **Time**: *day* < *month* < *quarter* < *year*
- ***Query: Q = {brand, state}***
  - Cuboid 1: G_1={year, item, city}?
  - χ Cuboid 2: G_2={year, brand, country}?
  - Cuboid 3: G_3={year, brand, state}?
  - Cuboid 4: G_4={item, state}?

cuboid	gi	qj
1	Item <	brand
	City <	state
2	brand <=	brand
	country >	**state**
3	…	

With hierarchical dimensions, if a query $Q=\{q_1, q_2, q_j, \ldots\}$, can be answered by a cuboid $G=\{g_1, g_2, g_i, \ldots\}$, where g_j and q_i are dimensions, if and only if, $q_j >= g_i$ for all hierarchical dimensions. For example, *Q={brand, state}, G={year, item, city}*, where we have *brand > item*; *state>city*.

Computing Query Cost (2-1)

Let *cost(Q,G)* be the cost of answering a GROUP BY query with dimension set *Q*, on the materialized view of *G*-cuboid T_G, while $Q \subseteq G$.

Let us define *cost(Q,G)* as follows:

$$\textbf{cost(Q,G)=SORT}(T_G) \quad \text{if } Q \subseteq G$$

where $SORT(T_G)$ is the cost of **sorting the materialized view** of *cuboid* T_G.

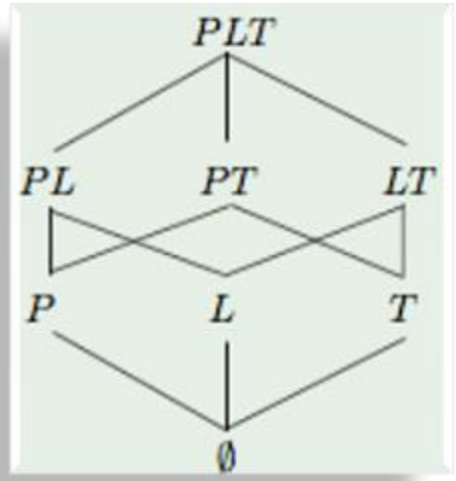
Note that:

For any Q_1 and Q_2 , where $Q_1 \neq Q_2$,

we have:

$cost(Q_1,G)=SORT(T_G)$ if $Q_1 \subseteq G$

$cost(Q_2,G)=SORT(T_G)$ if $Q_2 \subseteq G$



Computing Query Cost (2-2)

Now suppose that we have materialized a set S of cuboids $\{T_{G1}, ..., T_{Gk}\}$, $k \leq 2^d$.

- Assume that S always includes the fact table T

To answer a GROUP BY query with dimension set Q, we should use the cuboid (i.e., chosen from the ***Lattice***) as follows:

From all the $T_G \in S$ satisfying $Q \subseteq G$, choose the one

from the lattice with the smallest SORT (T_G), denoted as $cost_S(Q, G)$.

If T_G is the selected cuboid, we define:

$$cost_S(Q, G) = SORT(T_G); \quad Q \subseteq G.$$

In other words, ***$cost_S(Q, G)$*** is the lowest cost of answering the query Q from S, $S = \{T_{G1}, ..., T_{Gk}\}$, $k \leq 2^d$.

Computing Benefit of Materialized Views

Assume all GROUP BY queries are issued with the same frequency

The total cost of answering **all possible GROUP BY queries** from a **set *S* of materialized cuboids** is thus:

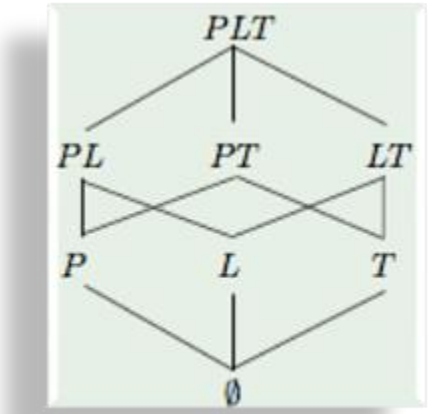
When |S| = 1, for t dimensionality, the total cost:

$$\begin{cases} \text{allcost(S)} = \sum_{\forall\, Q \subseteq G = \{A_1,\ldots,A_t\}} cost_S(Q,G) \\ \text{cost(Q, G)} = \text{SORT}(T_G) \end{cases}$$

Recall that any *S* will have to include **fact table *T***

$$\text{allcost}(\{T\}) = 2^t\, \text{SORT}(T)$$

$$benefit(S) = allcost(\{T\}) - allcost(S)$$



Strategies of Materializing Views

Target of View Materialization

- Minimize the total cost for all possible GROUP BY queries.

A naive solution

- Pre-compute and materialize all possible (i.e., 2^d) cuboids

What would be the problem?

- **Curse of dimensionality** (exponential increment of the cuboids)
  - *d* in practice can be very large
  - Cardinality of a dimension can be large
  - Dimensions may have hierarchies

Which cuboids should we choose to materialize?

Problem of Choosing Views to Materialize

The Problem of View Materialization

- Given the **fact table T**, the T_G={T_{G1}, T_{G2}, …, T_{Gn}}, an integer *k*, k ≤ n, **find a set *S* of *k* cuboids** (S ⊆ T_G) to be materialized **with the largest *benefit(S).***

This problem is known to be NP-hard. We will therefore turn to good approximate solutions

- Greedy algorithm
- Approximation ratio: *benefit(S) / benefit(S*)* = 1 − 1/e ≈ 0.632

Review

OLAP requires the creation of (typically) large, multi-dimensional **Data Warehouses (DW).**

OLAP operations (queries) include Roll-up, Drill-down, Pivoting, Slicing and Dicing, Cube, etc.

Widely-used indexing techniques to improve performance of data warehouse operations are: **bitmap index** and **join index.**

When implementing data warehouse, we need to consider the trade-off between **storage cost** and **query efficiency**.

GROUP BY query on materialized views will reduce the cost and bring benefit for computations.

Course Components