COURSE NAME:
DATA WAREHOUSING & DATA MINING

LECTURE 7
TOPICS TO BE COVERED:

- Data warehouse implementation
- Computation of data cubes
- Modelling OLAP data

- Data warehouses contain huge volumes of data. OLAP servers demand that decision support queries be answered in the order of seconds. Therefore, it is crucial for data warehouse systems to support highly efficient cube computation techniques, access methods and query processing techniques.

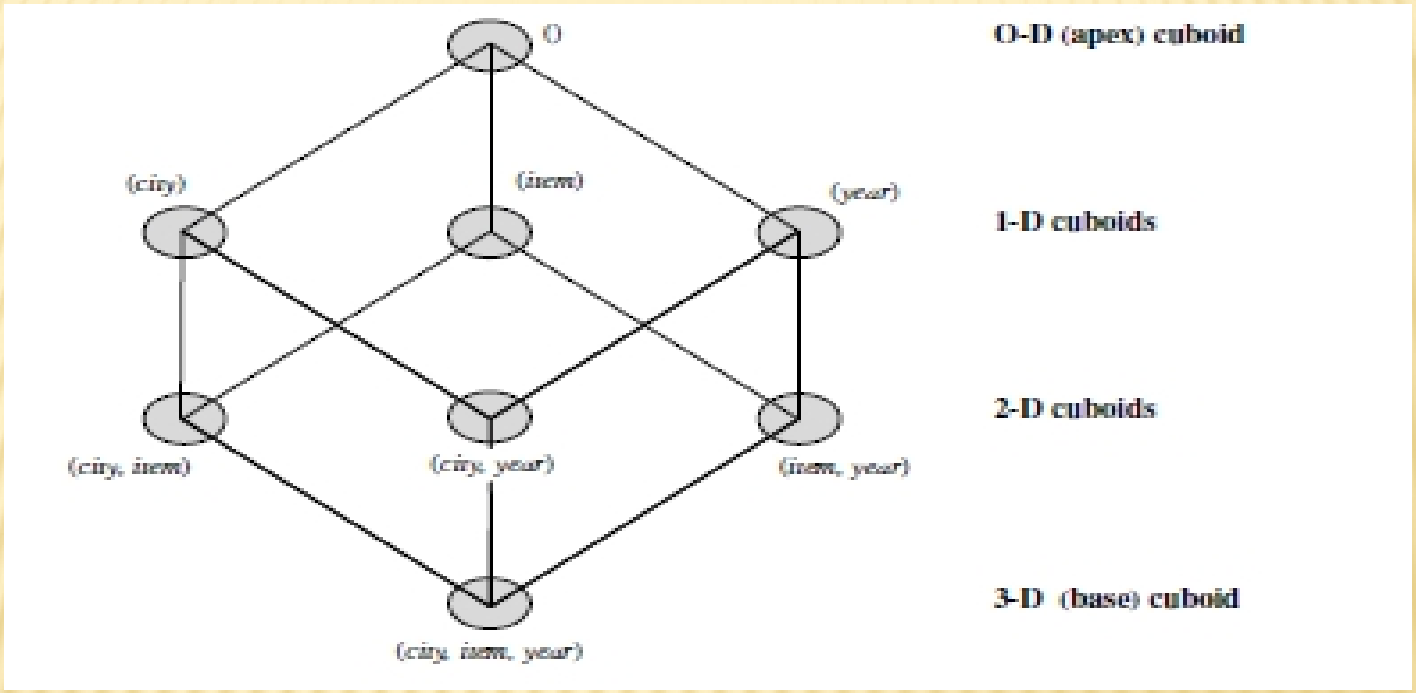
EFFICIENT COMPUTATION OF DATA CUBES

- At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions.
- In SQL terms, these aggregations are referred to as

  group-by's. Each group-by can be represented by a *cuboid, where the set of group-by's* forms a lattice of cuboids defining a data cube.

THE COMPUTE CUBE OPERATOR

- The compute cube operator computes aggregates over all subsets of the dimensions specified in the operation.
- This can require excessive storage space, especially for large numbers of dimensions.
- E.g.
  - *“Compute the sum of sales, grouping by city and item.”*
  - *“Compute the sum of sales, grouping by city.”*
  - *“Compute the sum of sales, grouping by item.”*



Lattice of cuboids, making up a 3-D data cube. Each cuboid represents a different group-by. The base cuboid contains the three dimensions *city, item*, and *year*.

- Data cube can be viewed as a lattice of cuboids
  - The bottom-most cuboid is the base cuboid
  - The top-most cuboid (apex) contains only one cell
  - How many cuboids in an n-dimensional cube with L levels?

$$T = \prod_{i=1}^{n} (L_i + 1)$$

- Where L_i is the number of levels associated with dimension i.

MATERIALIZATION OF DATA CUBE

There are three choices for data cube materialization given a base cuboid:

- **1. No materialization:** Do not precompute any of the “nonbase” cuboids. This leads to computing expensive multidimensional aggregates on the fly, which can be extremely slow.
- **2. Full materialization:** Precompute all of the cuboids. The resulting lattice of computed cuboids is referred to as the full cube. This choice typically requires huge amounts of memory space in order to store all of the precomputed cuboids.
- **3. Partial materialization:** Selectively compute a proper subset of the whole set of possible Cuboids Partial materialization represents an interesting trade-off between storage space and response time.

    - Selection of which cuboids to materialize
        - Based on size, sharing, access frequency, etc.

INDEXING OLAP DATA

- To facilitate efficient data accessing, most data warehouse systems support index structures
  and materialized views (using cuboids).
- To index OLAP data by *bitmap indexing and join indexing.*

INDEXING OLAP DATA: BITMAP INDEX

- The bitmap indexing method is popular in OLAP products because it allows quick searching in data cubes. The bitmap index is an alternative representation of the *record ID (RID) list.*
- *In the bitmap index for a given attribute, there is a distinct bit* vector, *Bv, for each value v in the domain of the attribute. If the domain of a given* attribute consists of *n values, then n bits are needed for each entry in the bitmap index*
- (i.e., there are *n bit vectors). If the attribute has the value v for a given row in the data* table, then the bit representing that value is set to 1 in the corresponding row of the bitmap index. All other bits for that row are set to 0.

INDEXING OLAP DATA: BITMAP INDEX

- Index on a particular column
- Each value in the column has a bit vector: bit-op is fast
- 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
- not suitable for high cardinality domains

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

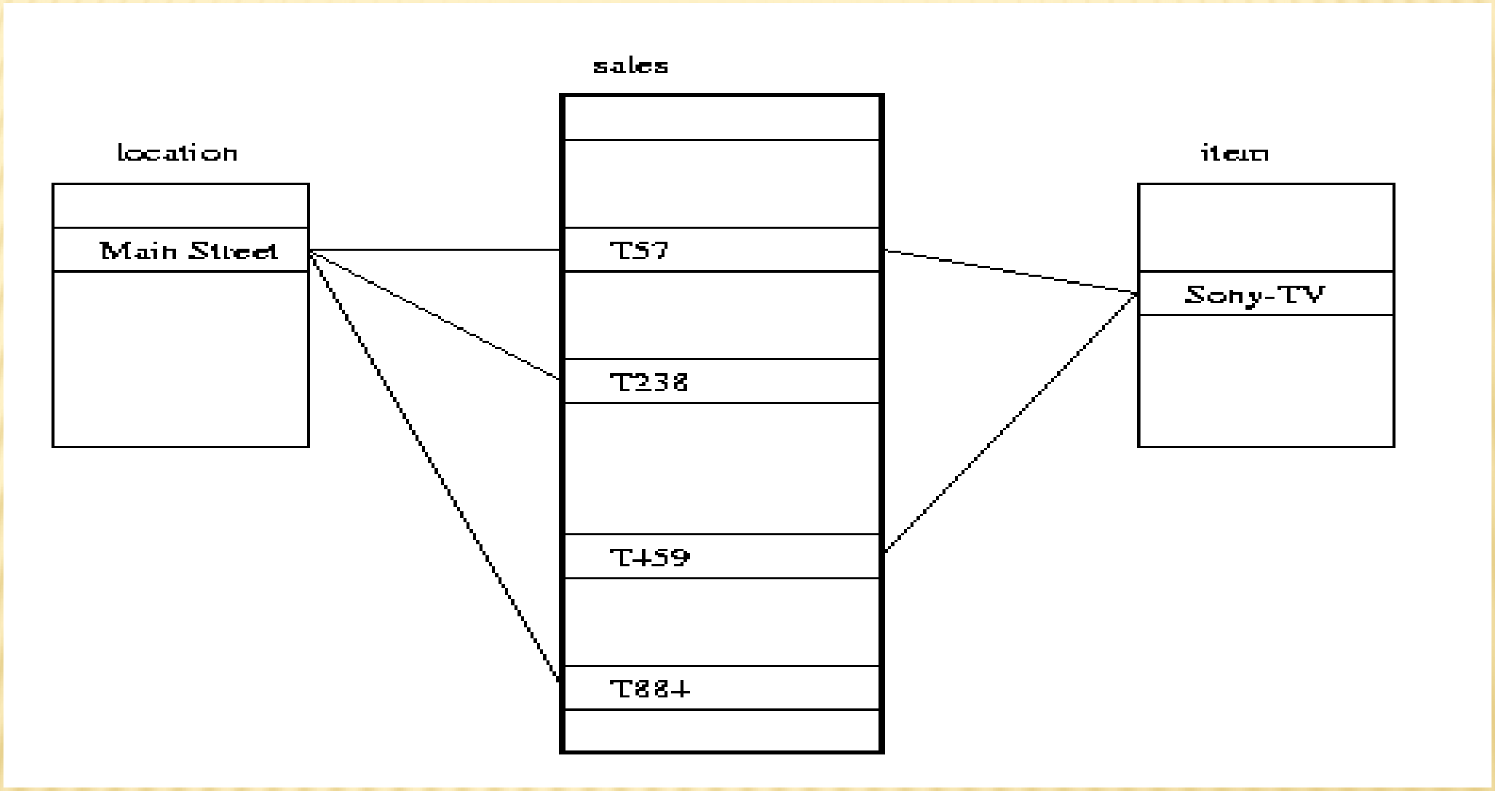
RecID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

INDEXING OLAP DATA: JOIN INDICES

- Join index: JI(R-id, S-id) where R (R-id, …) $\triangleright\triangleleft$ S (S-id, …)
- Traditional indices map the values to a list of record ids
  - It materializes relational join in JI file and speeds up relational join
- In data warehouses, join index relates the values of the dimensions of a start 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 the city
  - Join indices can span multiple dimensions

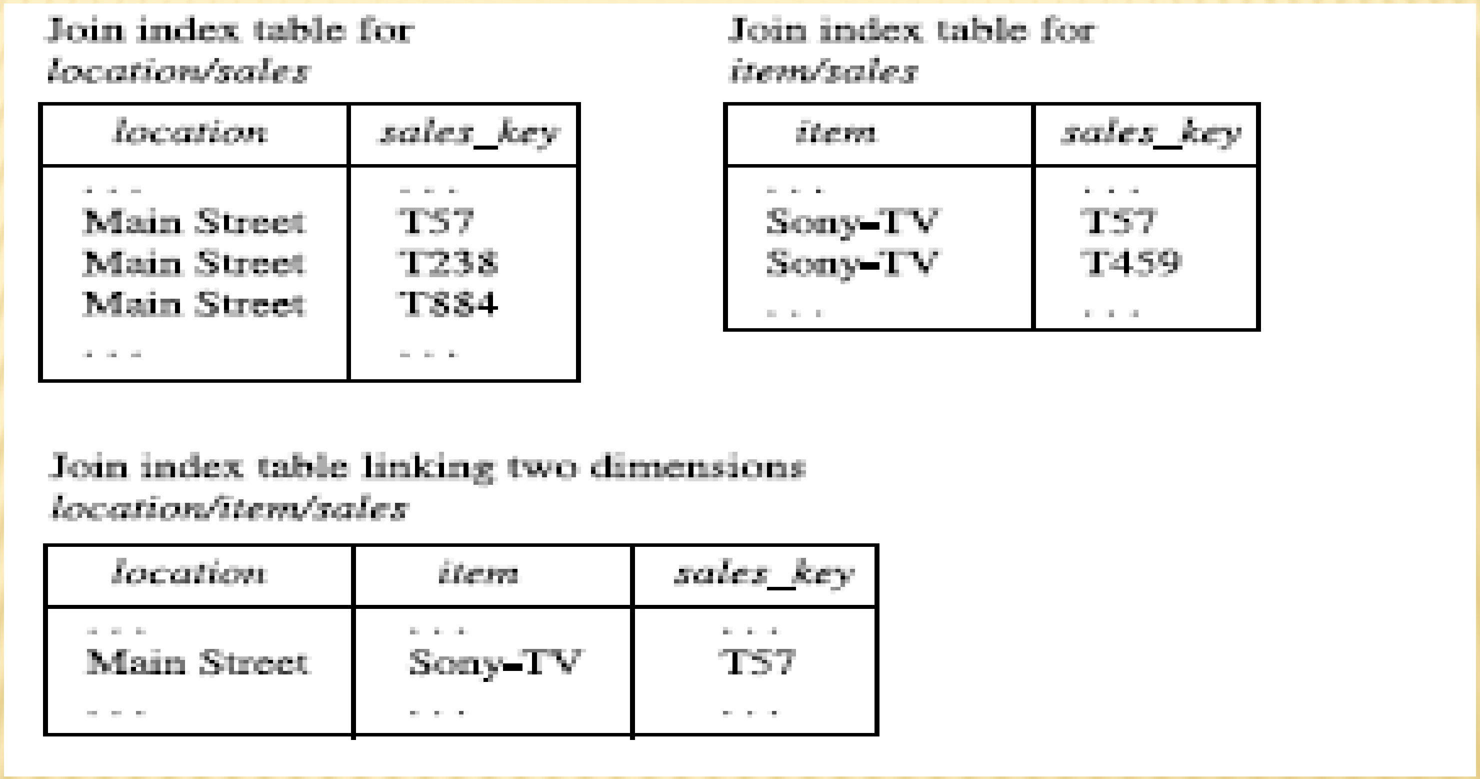
- Join indexing registers the joinable rows of two or more relations from a relational database, reducing the overall cost of OLAP join operations.

INDEXING OLAP DATA: JOIN INDEXING



Linkages between a *sales fact table and dimension tables for location and item.*

INDEXING OLAP DATA: JOIN INDEXING



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 two dimensions *location/item/sales*

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

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

EFFICIENT PROCESSING OLAP QUERIES

- **Determine which operations should be performed on the available cuboids**
  - Transform drill, roll, etc. into corresponding SQL and/or OLAP operations, e.g., dice = selection + projection
- **Determine which materialized cuboid(s) should be selected for OLAP op**.
  - Let the query to be processed be on {brand, province_or_state} with the condition "year = 2004", and there are 4 materialized cuboids available:

    1) {year, item_name, city}

    2) {year, brand, country}

    3) {year, brand, province_or_state}

    4) {item_name, province_or_state} where year = 2004

    Which should be selected to process the query?