# Data Warehousing

CPS 216
Advanced Database Systems

## Review

Data warehousing: integrating data for OLAP

- OLAP versus OLTP
- Warehousing versus mediation
- Warehouse maintenance
  - Warehouse data as materialized views
  - Recomputation versus incremental maintenance
  - Self-maintenance



## Today

- Star, snowflake, and cube
- ROLAP and MOLAP algorithms

# Star schema

Product — Dimension table

| PID | name | cost |
|---|---|---|
| p1 | beer | 10 |
| p2 | diaper | 16 |
| … | … | … |

Store — Dimension table

| SID | city |
|---|---|
| s1 | Durham |
| s2 | Chapel Hill |
| s3 | RTP |
| … | … |

Sale

| OID | date | CID | PID | SID | qty | price |
|---|---|---|---|---|---|---|
| 100 | 11/23/2001 | c3 | p1 | s1 | 1 | 12 |
| 102 | 12/12/2001 | c3 | p2 | s1 | 2 | 17 |
| 105 | 12/24/2001 | c5 | p1 | s3 | 5 | 13 |
| … | … | … | … | … | … | … |

Fact table
- Big
- Constantly growing
- Stores measures (often aggregated in queries)

Customer

| CID | name | address | city |
|---|---|---|---|
| c3 | Amy | 100 Main St. | Durham |
| c4 | Ben | 102 Main St. | Durham |
| c5 | Carl | 800 Eighth St. | Durham |
| … | … | … | … |

Dimension table
- Small
- Updated infrequently

4

# Dimension hierarchies

Product — Brand

Product — Product type — Product category

| brandID | name |
|---|---|
| b1 | Huggies |
| b2 | Budwiser |
| … | … |

| PID | brandID | typeID |
|---|---|---|
| p1 | b1 | t1 |
| p2 | b2 | t2 |
| p3 | b1 | t4 |
| … | … | … |

| typeID | name | catID |
|---|---|---|
| t1 | diaper | cat7 |
| t2 | beer | cat9 |
| t3 | juice | cat9 |
| t4 | pacifier | cat7 |
| … | … | … |

| catID | name |
|---|---|
| cat7 | babies |
| cat9 | beverages |
| … | … |

+ star schema = snowflake schema

5

# Star join indexes

- Queries frequently join fact table with dimension tables
  - » Materialize the join result to speed up queries
- For each combination of dimension attribute values, store the list of tuple ID's in the fact table
  - Brand name, store city, customer city → sales records; Product type, store city → sales records; etc.
  - Conceptually, multi-attribute indexes on the join result
- One index to support each combination of selection conditions on attributes?
  - Too many indexes!

6

# Bitmap join indexes

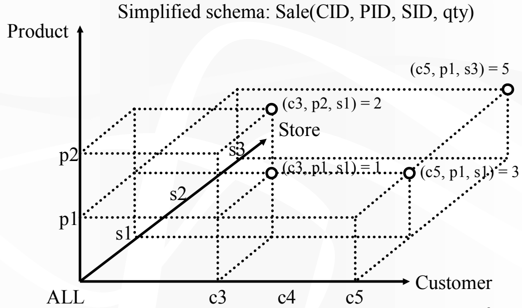
» O'Neil & Quass, SIGMOD 1997

- Bitmap and projection indexes for each dimension attribute
  - Value of the dimension attribute ↔ tuple ID's in the fact table
- To process an arbitrary combination of selection conditions, use bitmap indexes
  - Bitmaps can be combined efficiently
- To retrieve attribute values for output, use projection indexes

7

# Data cube

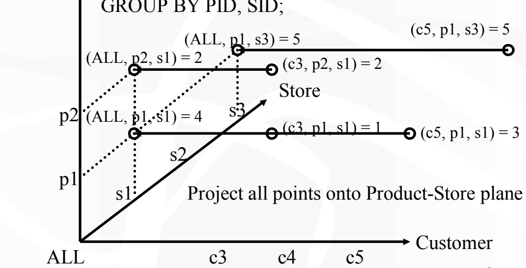
Simplified schema: Sale(CID, PID, SID, qty)




8

# Completing the cube (slide 1)

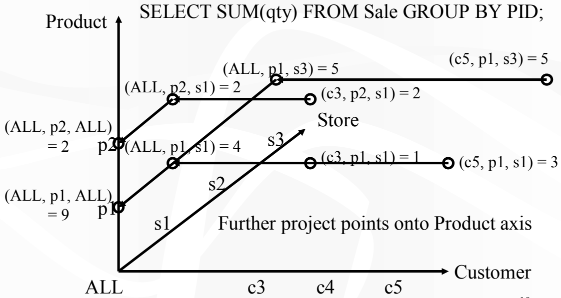
Total quantity of sales for each product in each store
SELECT SUM(qty) FROM Sale
GROUP BY PID, SID;




9

# Completing the cube (slide 2)

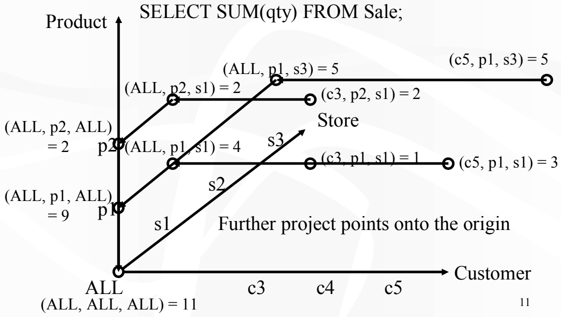
Total quantity of sales for each product
SELECT SUM(qty) FROM Sale GROUP BY PID;






# Completing the cube (slide 3)

Total quantity of sales
SELECT SUM(qty) FROM Sale;






# CUBE operator

» Gray et al., ICDE 1996
- Sale(CID, PID, SID, qty)
- Proposed SQL extension:
  SELECT SUM(qty) FROM Sale
  GROUP BY CUBE CID, PID, SID;
- Output contains:
  - Normal groups produced by GROUP BY
    - (c1, p1, s1, sum), (c1, p2, s3, sum), etc.
  - Groups with one or more ALL's
    - (ALL, p1, s1, sum), (c2, ALL, ALL, sum), (ALL, ALL, ALL, sum), etc.
- Can you write a CUBE query using only GROUP BY's?

## ROLLUP operator

- Sometimes CUBE is too much
  - (…, state, city, street, …, age, DOB, …)
  - CUBE state, city, street returns meaningless groups
    - (ALL, ALL, ’Main Street’): sales on any Main Street?
  - CUBE age, DOB returns useless groups
    - (ALL, DOB): DOB functionally determines age!
- Proposed SQL extension:
  GROUP BY ROLLUP state, city, street;
- Output contains groups with ALL’s only as suffix
  - (’NC’, ’Durham’, ’Main Street’), (’NC’, ’Durham’, ALL), (’NC’, ALL, ALL), (ALL, ALL, ALL)
  - But not (ALL, ALL, ’Main Street’) or (ALL, ’Durham’, ALL)

13

## Computing GROUP BY

- ROLAP (Relational OLAP)
  - Use standard relational engine
  - Sorting and clustering
  - Using indexes
  - Automatic summary tables
- MOLAP (Multidimensional OLAP)
  - Use a sparse multidimensional array

14

## Sorting and clustering

- Sort (or cluster, e.g., using hashing) tuples according to GROUP BY attributes
  - Tuples in the same group are processed together
  - Only one intermediate aggregate result needs to be kept—low memory requirement
- What if GROUP BY attributes ≠ sort attributes?
  - Still fine if GROUP BY attributes form a prefix of the sort order
  - Otherwise, need to keep intermediate aggregate results around

15

## More on sort order

- Sort by the order in which GROUP BY attributes appear?
  - Not necessary; e.g., GROUP BY PID, SID can be processed just as efficiently by sorting on SID, PID
- Sort by the order in which GROUP BY ROLLUP attributes appear?
  - Useful; e.g., GROUP BY ROLLUP state, city, street can be processed efficiently by sorting on state, city, street, but not by sorting on street, city, state

16

## Using bitmap join indexes

» O'Neil & Quass, SIGMOD 1997

- Use the bitmap join indexes on $GB_1$, $GB_2$, …, $GB_k$
- For each value $v_1$ of $GB_1$ in order:
  For each value $v_2$ of $GB_2$ in order: …
    For each value $v_k$ of $GB_k$ in order:
      Intersect bitmaps to locate tuples;
      Retrieve their measures;
      Calculate aggregate for group ($v_1$, $v_2$, …, $v_k$);
- Helps if data is sorted by $GB_1$, $GB_2$, …, $GB_k$
  - So measures in the same group are clustered

17

## Automatic summary tables

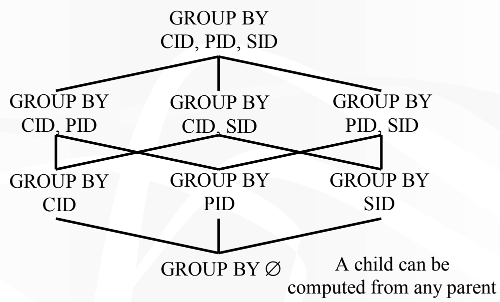
- Computing GROUP BY aggregates is expensive
- OLAP queries perform GROUP BY all the time

- Idea: precompute and store the aggregates!

» Automatic summary tables
  - Maintained automatically as base data changes
  - Just another index/materialized view

18

## Aggregation view lattice




19

## Selecting views to materialize

- Factors in deciding what to materialize
  - What is its storage cost?
  - What is its update cost?
  - Which queries can benefit from it?
  - How much can a query benefit from it?
- Example
  - GROUP BY ∅ is small, but not useful to most queries
  - GROUP BY CID, PID, SID is useful to any query, but too large to be beneficial

» Harinarayan et al., SIGMOD 1996; Gupta & Mumick, ICDE 1999

20

## Interlude: TPC-D, -H, and -R

- TPC-D: standard OLAP benchmark until 1999
  - With aggressive use of precomputation techniques (materialized views, automatic summary tables), vendors were able to “cheat” and achieve amazing performance
- Now, TPC-D has been replaced by
  - TPC-H: ad hoc OLAP queries
    - Cannot use materialized views
  - TPC-R: business-reporting OLAP queries
    - Can use materialized views

» http://www.tpc.org/

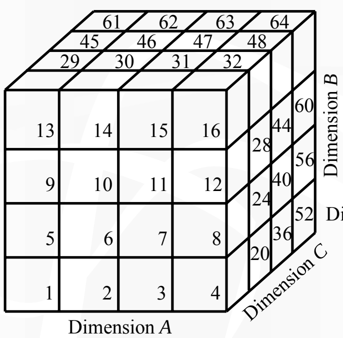
21

## From tables to arrays

» Zhao et al., SIGMOD 1997
- "Chunk" an *n*-dimensional cube into *n*-dimensional subcubes
  - For a dense chunk (>40% full), store it as is
  - For a sparse chunk (<40% full), compress it using <*coordinate*, *value*> pairs
- To convert a table into chunks
  - Pass 1: Partition table into memory-size partitions, each of which contains a number of chunks
  - Pass 2: Read partitions back in one at a time, and chunk each partition in memory



## Dimension order



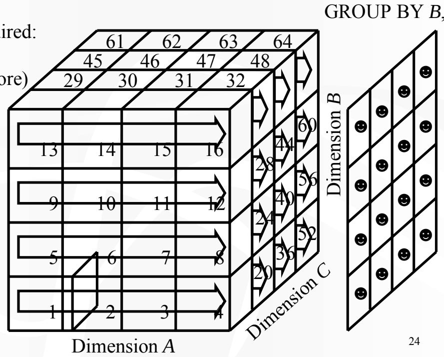

Dimension order: *A*, *B*, *C*
= Sort order: *C*, *B*, *A*



## Basic array cubing

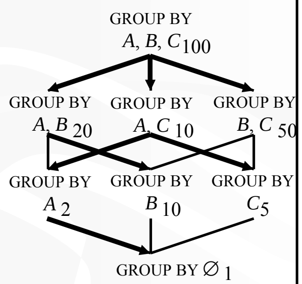
Memory required: 1 chunk (could be more)

GROUP BY *B*, *C*



## Minimal spanning tree

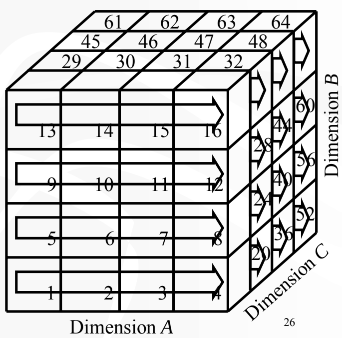
- Recall the aggregation lattice
- MST of the lattice: parent is always chosen to be the one with minimum size
- Compute each node from its parent in the MST




25

## Multiway array cubing

- Goal: compute all aggregates at the same time in a single pass over the array, using minimum amount of memory
- GROUP BY *B*, *C* requires 1 chunk
- GROUP BY *A*, *C* requires 4 chunks
- GROUP BY *A*, *B* requires 16 chunks




26

## Memory requirement

- Dimension order is $D_1, D_2, \ldots, D_n$
- Aggregate to compute projects out $D_p$ (i.e., GROUP BY $D_1, \ldots, D_{p-1}, D_{p+1}, \ldots, D_n$)
- The memory required is roughly $| D_1 | \cdot | D_1 | \cdot \ldots \cdot | D_{p-1} |$ chunks
  - Where $| D_i |$ denotes the number of chunks along $D_i$

» It is harder to aggregate away dimensions that come later in the dimension order

27

## Minimum-memory spanning tree

- MMST of the aggregation lattice
  - Parent is always chosen to be the one that makes the child require the minimum memory to compute
  - Note that results are produced in dimension order too, so computation of the entire MMST can be pipelined
- Choose an optimal dimension order to minimize the total amount of memory required by MMST
  - It turns out that this optimal order is $D_1, D_2, \ldots, D_n$, where $|D_1| \leq |D_2| \leq \ldots \leq |D_n|$

28

## ROLAP versus MOLAP

- Multiway array cubing algorithm (MOLAP) beats sorting-based ROLAP algorithms
  - Compressed array representation is more compact than table representation
  - Sorting-based ROLAP spends too much time on comparing and copying
  - In MOLAP, order is implied by the array positions

» An alternative ROLAP techinque
  - Convert table to array
  - Do MOLAP processing
  - Dump the result cube to a table

29

## Next time

Data mining

30