Data Warehousing & Mining

CPS 196.3

Introduction to Database Systems

Data integration

- ❖ Data resides in many distributed, heterogeneous OLTP (On-Line Transaction Processing) sources
  - Sales, inventory, customer, …
  - NC branch, NY branch, CA branch, …
- ❖ Need to support OLAP (On-Line Analytical Processing) over an integrated view of the data
- ❖ Possible approaches to integration
  - Eager: integrate in advance and store the integrated data at a central repository called the data warehouse
  - Lazy: integrate on demand; process queries over distributed sources

OLTP versus OLAP

OLTP	OLAP
❖ Mostly updates	❖ Mostly reads
❖ Short, simple transactions	❖ Long, complex queries
❖ Clerical users	❖ Analysts, decision makers
❖ Goal: ACID, transaction throughput	❖ Goal: fast queries

Eager versus lazy integration

Eager (warehousing)	Lazy
❖ In advance: before queries	❖ On demand: at query time
❖ Copy data from sources	❖ Leave data at sources
☞ Answer could be stale	☞ Answer is more up-to-date
☞ Need to maintain consistency	☞ No need to maintain consistency
☞ Query processing is local to the warehouse	☞ Sources participate in query processing
▪ Faster	▪ Slower
▪ Can operate when sources are unavailable	▪ Interferes with local processing

Maintaining a data warehouse

- ❖ The "ETL" process
  - Extraction: extract relevant data and/or changes from sources
  - Transformation: transform data to match the warehouse schema
  - Loading: integrate data/changes into the warehouse
- ❖ Approaches
  - Recomputation
    - Easy to implement; just take periodic dumps of the sources, say, every night
    - What if there is no "night," e.g., a global organization?
    - What if recomputation takes more than a day?
  - Incremental maintenance
    - Compute and apply only incremental changes; fast if changes are small
    - Not easy to do for complicated transformations
    - Need to detect incremental changes at the sources

"Star" schema of a data warehouse

Dimension table

Product

PID	*name*	*cost*
p1	beer	10
p2	diaper	16
...	...	...

Dimension table

Store

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

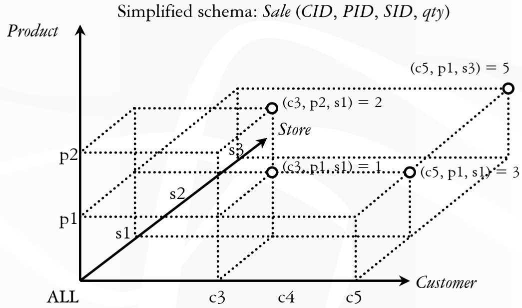
CID	*name*	*address*	*city*
c3	Amy	100 Main St.	Durham
c4	Ben	102 Main St.	Durham
c5	Coy	800 Eighth St.	Durham
...	...	...	...

Dimension table

- ❖ Small
- ❖ Updated infrequently

Data cube

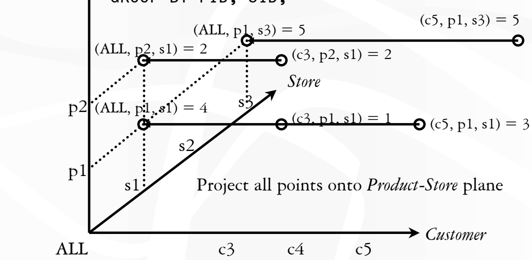
Simplified schema: *Sale* (*CID*, *PID*, *SID*, *qty*)



Completing the cube—plane

Total quantity of sales for each product in each store

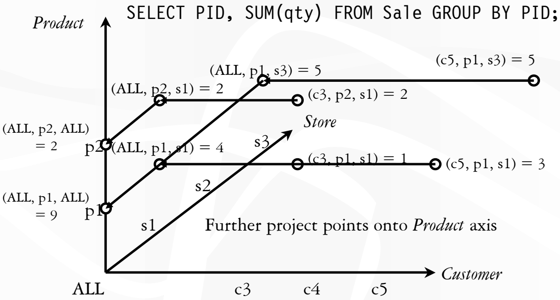
```
SELECT PID, SID, SUM(qty) FROM Sale
GROUP BY PID, SID;
```



Completing the cube—axis

Total quantity of sales for each product

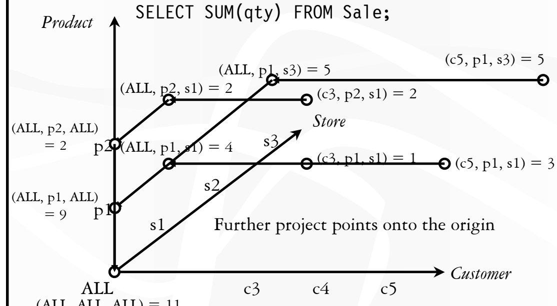
```
SELECT PID, SUM(qty) FROM Sale GROUP BY PID;
```



Completing the cube—origin

Total quantity of sales

```
SELECT SUM(qty) FROM Sale;
```



`CUBE` operator

- *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?

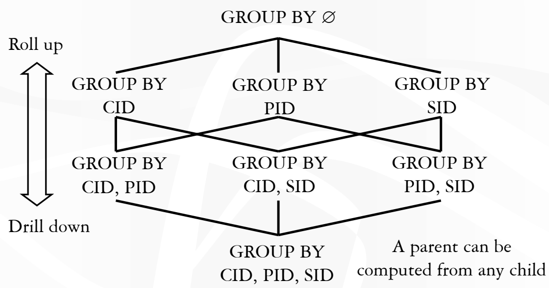
Gray et al., "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total." *ICDE* 1996

Automatic summary tables

- Computing `GROUP BY` and `CUBE` aggregates is expensive
- OLAP queries perform these operations over and over again

☞ Idea: precompute and store the aggregates as automatic summary tables (a DB2 term)
  - Maintained automatically as base data changes
  - Same as materialized views

Aggregation view lattice



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., "Implementing Data Cubes Efficiently." *SIGMOD* 1996

Data mining

- Data → knowledge
- DBMS meets AI and statistics
- Clustering, prediction (classification and regression), association analysis, outlier analysis, evolution analysis, etc.
  - Usually complex statistical "queries" that are difficult to answer → often specialized algorithms outside DBMS

- We will focus on frequent itemset mining

Mining frequent itemsets

- Given: a large database of transactions, each containing a set of items
  - Example: market baskets
- Find all frequent itemsets
  - A set of items X is frequent if no less than $s_{\min}\%$ of all transactions contain X
  - Examples: {diaper, beer}, {scanner, color printer}

TID	*items*
T001	diaper, milk, candy
T002	milk, egg
T003	milk, beer
T004	diaper, milk, egg
T005	diaper, beer
T006	milk, beer
T007	diaper, beer
T008	diaper, milk, beer, candy
T009	diaper, milk, beer
...	...

First try

- A naïve algorithm
  - Keep a running count for each possible itemset
  - For each transaction T, and for each itemset X, if T contains X then increment the count for X
  - Return itemsets with large enough counts
- Problem: The number of itemsets is huge!
  - 2^n, where n is the number of items
- Think: How do we prune the search space?

The Apriori property

- All subsets of a frequent itemset must also be frequent
  - Because any transaction that contains X must also contains subsets of X

☞ If we have already verified that X is infrequent, there is no need to count X's supersets because they must be infrequent too

The Apriori algorithm

Multiple passes over the transactions

- Pass k finds all frequent k-itemsets (itemset of size k)
- Use the set of frequent k-itemsets found in pass k to construct candidate $(k+1)$-itemsets to be counted in pass $(k+1)$
  - A $(k+1)$-itemset is a candidate only if all its subsets of size k are frequent

Example: pass 1

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{\min}\% = 20\%$

Frequent 1-itemsets

itemset	*count*
{A}	6
{B}	7
{C}	6
{D}	2
{E}	2

(Itemset {F} is infrequent)

Example: pass 2

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{\min}\% = 20\%$

Generate candidates

Scan and count

Check min. support

itemset	*count*
{A}	6
{B}	7
{C}	6
{D}	2
{E}	2

Frequent 1-itemsets

itemset	*count*
{A,B}	4
{A,C}	4
{A,D}	1
{A,E}	2
{B,C}	4
{B,D}	2
{B,E}	2
{C,D}	0
{C,E}	1
{D,E}	0

Candidate 2-itemsets

itemset	*count*
{A,B}	4
{A,C}	4
{A,E}	2
{B,C}	4
{B,D}	2
{B,E}	2

Frequent 2-itemsets

Example: pass 3

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{\min}\% = 20\%$

Generate candidates

Scan and count

Check min. support

itemset	*count*
{A,B}	4
{A,C}	4
{A,E}	2
{B,C}	4
{B,D}	2
{B,E}	2

Frequent 2-itemsets

itemset	*count*
{A,B,C}	2
{A,B,E}	2

Candidate 3-itemsets

itemset	*count*
{A,B,C}	2
{A,B,E}	2

Frequent 3-itemsets

Example: pass 4

TID	*items*
T001	A, B, E
T002	B, D
T003	B, C
T004	A, B, D
T005	A, C
T006	B, C
T007	A, C
T008	A, B, C, E
T009	A, B, C
T010	F

Transactions

$s_{\min}\% = 20\%$

Generate candidates

itemset	*count*
{A,B,C}	2
{A,B,E}	2

Frequent 3-itemsets

itemset	*count*

Candidate 4-itemsets

No more itemsets to count!

Example: final answer

itemset	*count*
{A}	6
{B}	7
{C}	6
{D}	2
{E}	2

Frequent 1-itemsets

itemset	*count*
{A,B}	4
{A,C}	4
{A,E}	2
{B,C}	4
{B,D}	2
{B,E}	2

Frequent 2-itemsets

itemset	*count*
{A,B,C}	2
{A,B,E}	2

Frequent 3-itemsets