Data Warehousing

Introduction to Databases

CompSci 316 Fall 2016

DUKE
COMPUTER SCIENCE

Announcements (Thu., Dec. 8)

- Homework #4 last Gradiance problem due today
  - Sample solution to be posted by this weekend
- Project demos to start tomorrow
  - Check your email for schedule
  - Submit report/code before demo (you have until next Thursday to update it)
- Final exam Thur. Dec. 15 7-10pm
  - **Different room: LSRC B101**
  - Open-book, open-notes
  - Comprehensive, but with strong emphasis on the second half of the course
  - Sample final + solution posted on Sakai

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—mediated or federated systems

OLTP versus OLAP

OLTP

- Mostly updates
- Short, simple transactions
- Clerical users
- Goal: transaction throughput

OLAP

- Mostly reads
- Long, complex queries
- Analysts, decision makers
- Goal: fast queries

Implications on database design and optimization?

OLAP databases do not care much about redundancy

- "Denormalize" tables
- Many, many indexes
- Precomputed query results

Eager versus lazy integration

Eager (warehousing)

- In advance: before queries
- Copy data from sources

☞Answer could be stale

☞Need to maintain consistency

☞Query processing is local to the warehouse

  - Faster
  - Can operate when sources are unavailable

Lazy

- On demand: at query time
- Leave data at sources

☞Answer is more up-to-date

☞No need to maintain consistency

☞Sources participate in query processing

  - Slower
  - Interferes with local processing
  - Still has consistency issues

Maintaining a data warehouse

- The "ETL" process
  - Extract relevant data and/or changes from sources
  - Transform data to match the warehouse schema
  - Load/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

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	08/23/2015	c3	p1	s1	1	12
102	09/12/2015	c3	p2	s1	2	17
105	09/24/2015	c5	p1	s3	5	13
…	…	…	…	…	…	…

Fact table

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

Customer

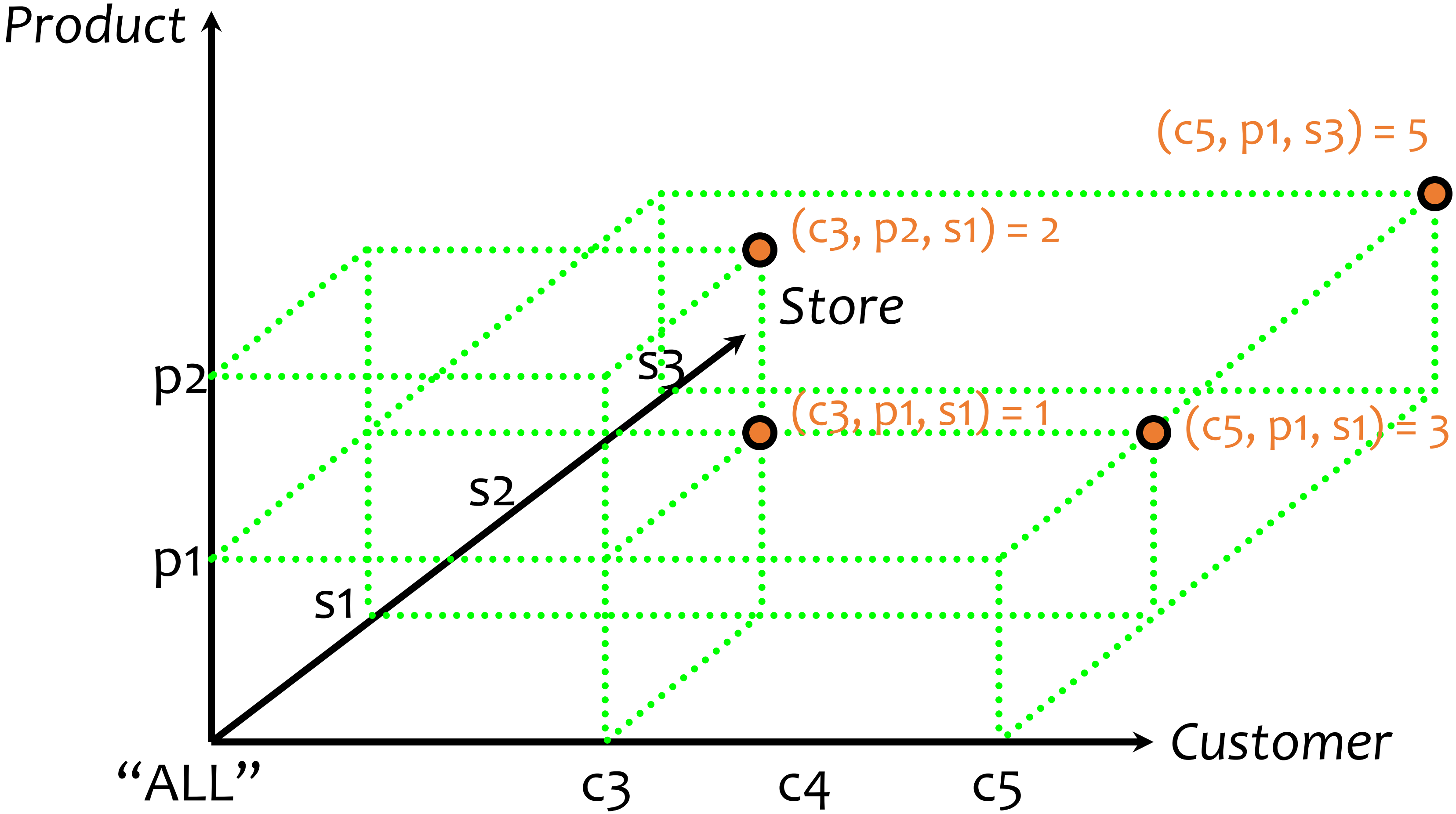
CID	name	address	city
c3	Amy	100 Main St.	Durham
c4	Ben	102 Main St.	Durham
c5	Coy	800 Eighth St.	Durham
…	…	…	…

Dimension table

- Small
- Updated infrequently

Data cube

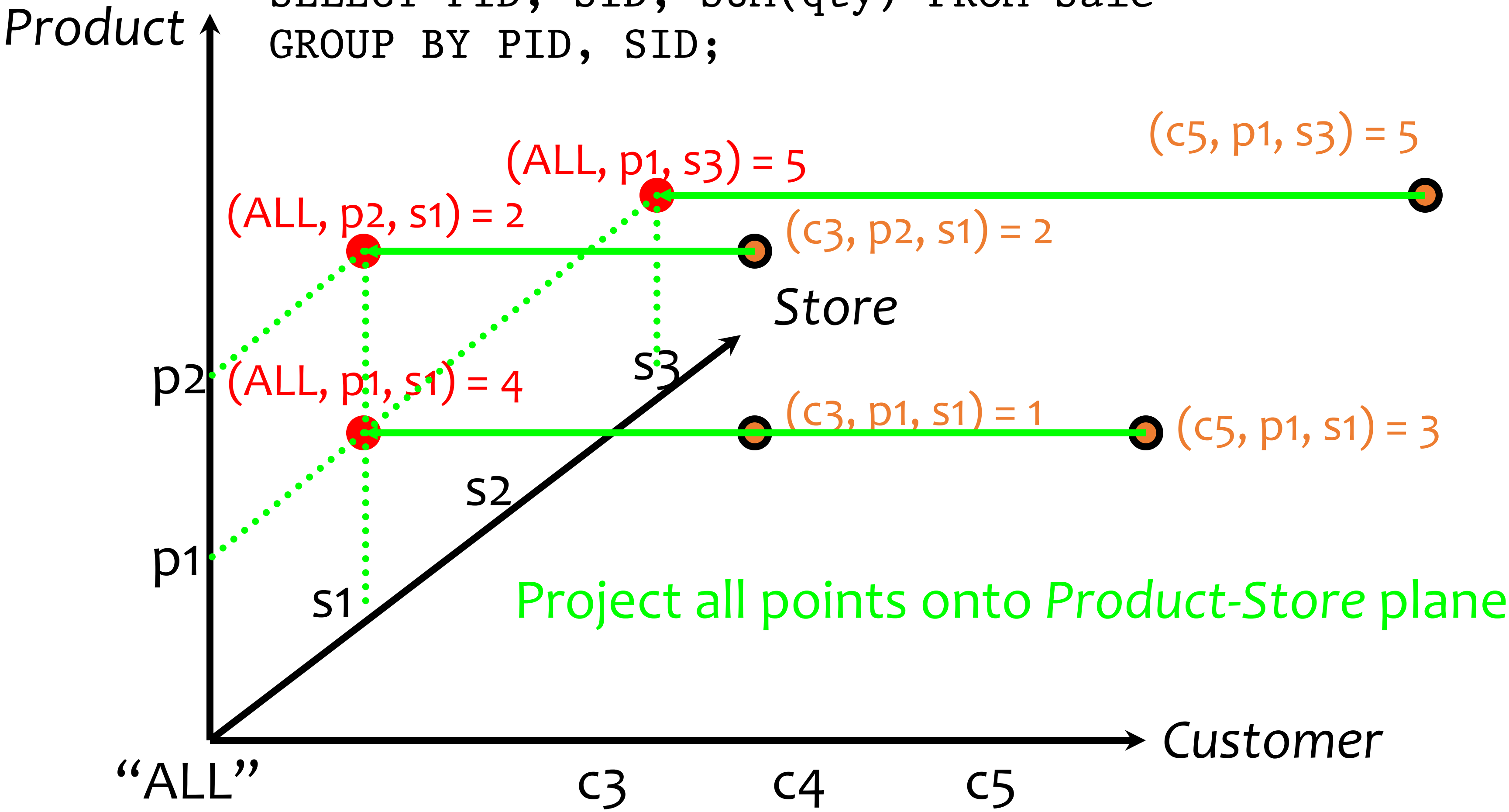
Simplified schema: *Sale* (*CID*, *PID*, *SID*, *qty*)



Completing the cube—plane

Total quantity of sales for each product in each store

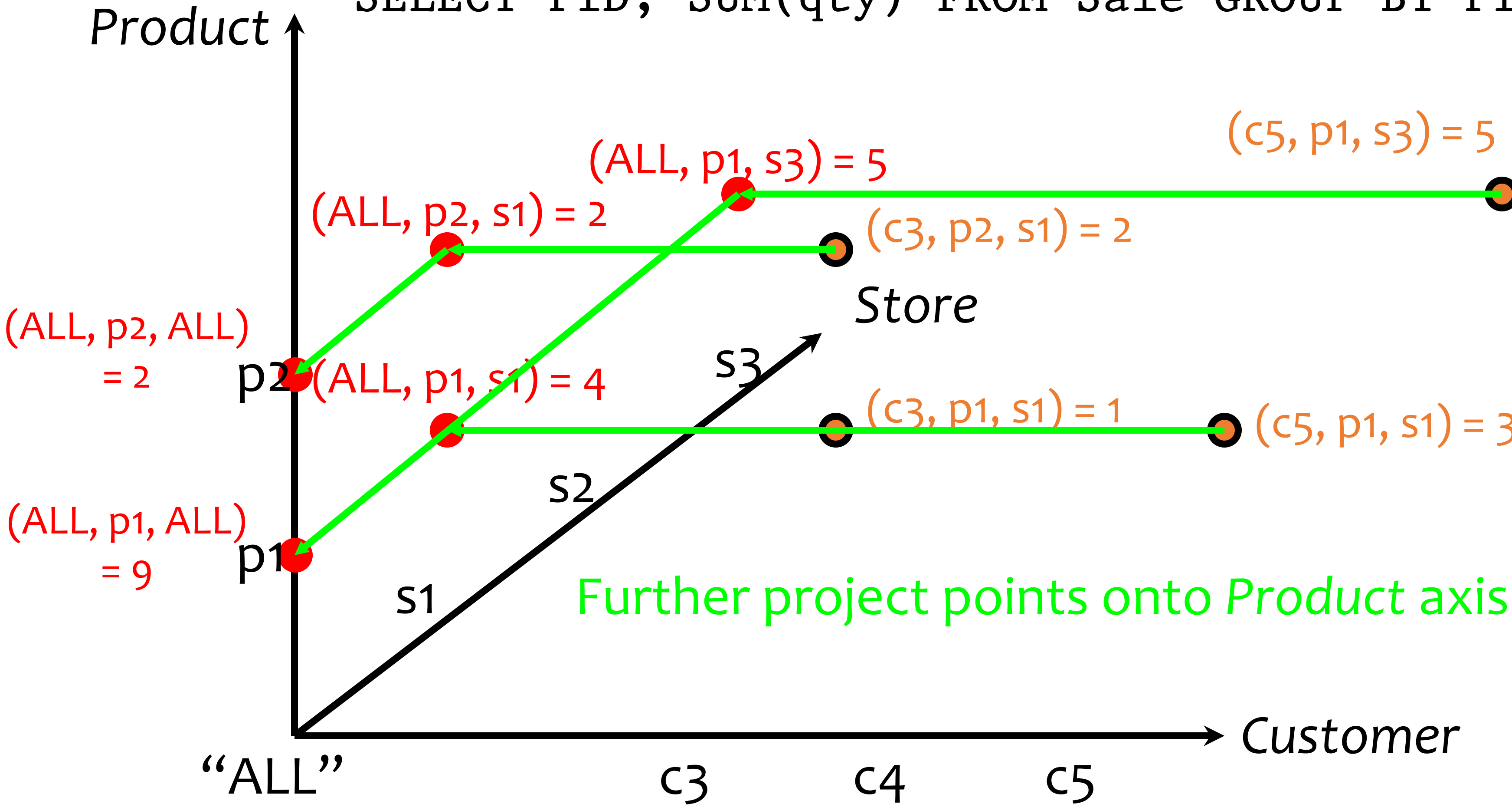
```
SELECT PID, SID, SUM(qty) FROM Sale
GROUP BY PID, SID;
```



Completing the cube—axis

Total quantity of sales for each product

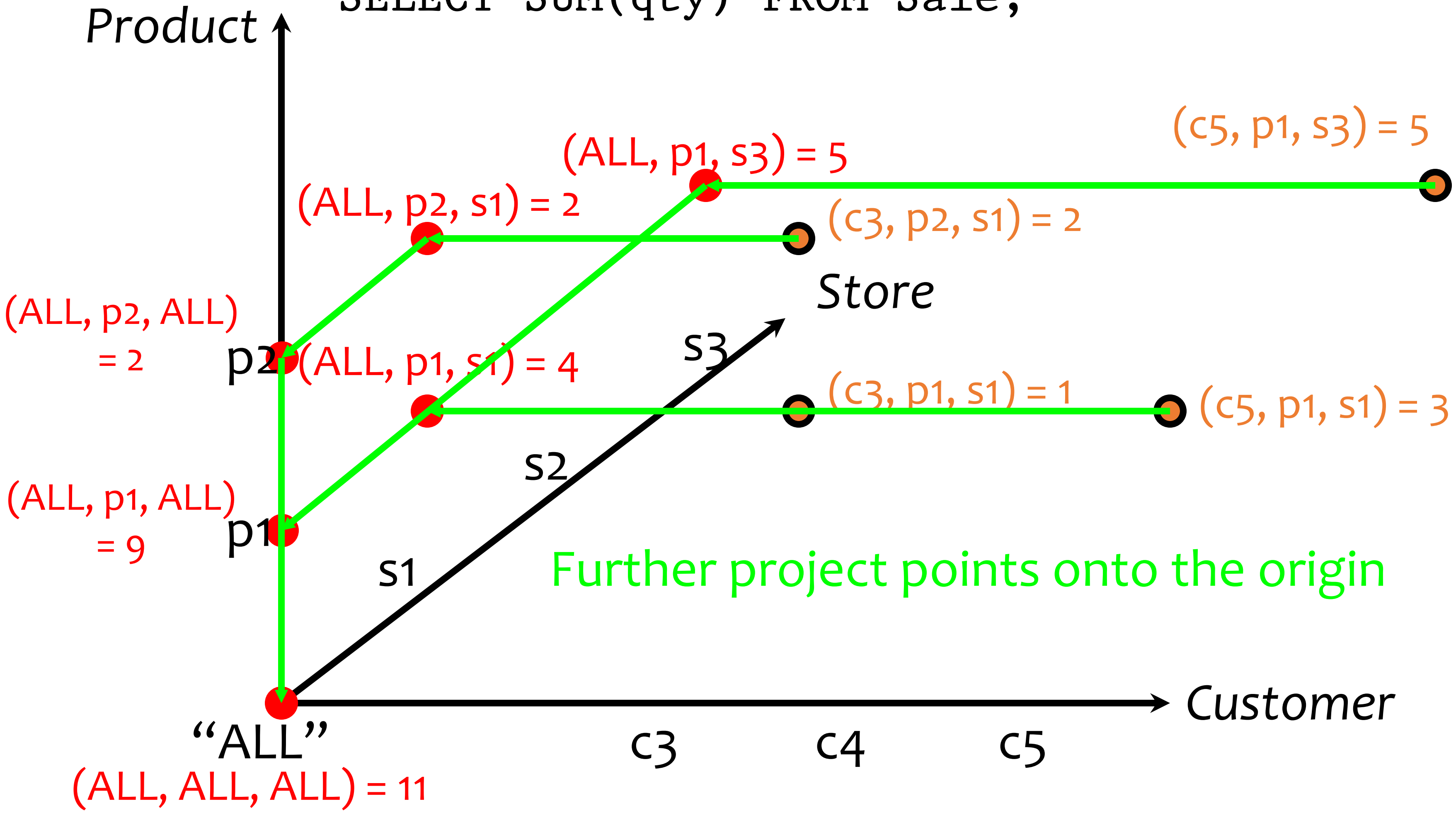
```
SELECT PID, SUM(qty) FROM Sale GROUP BY PID;
```



Completing the cube—origin

Total quantity of sales

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

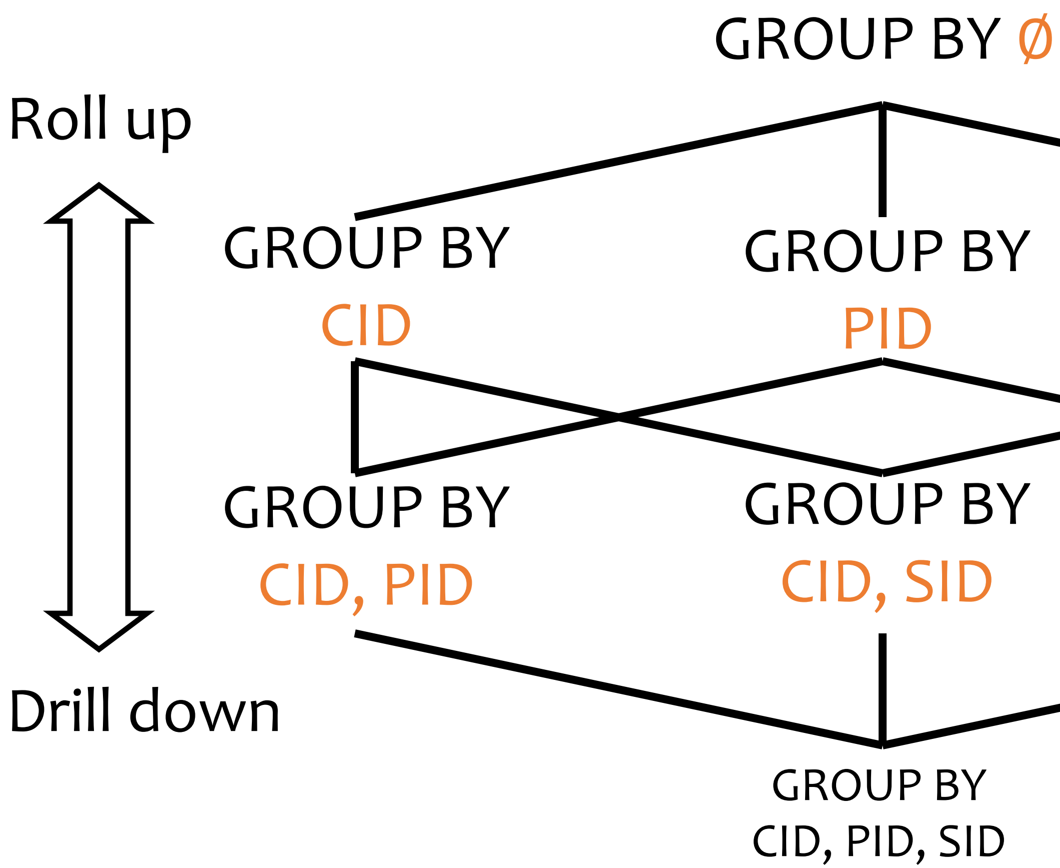


`CUBE` operator

- *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

Aggregation lattice



A parent can be computed from any child

Materialized views

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

☞Idea: precompute and store the aggregates as materialized views

- Maintained automatically as base data changes
- No. 1 user-requested feature in PostgreSQL!

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

Other OLAP extensions

- Besides extended grouping capabilities (e.g., `CUBE`), window operations have also been added to SQL
- A "window" specifies an ordered list of rows related to the "current row"
- A window function computes a value from this list and the "current row"
  - Standard aggregates: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`
  - New functions: `RANK`, `PERCENT_RANK`, `LAG`, `LEAD`, …

RANK window function example

sid	pid	cid	qty
Durham	beer	Alice	10
Durham	beer	Bob	2
Durham	chips	Bob	3
Durham	diaper	Alice	5
Raleigh	beer	Alice	2
Raleigh	diaper	Bob	100

```
SELECT SID, PID, SUM(qty),
       RANK() OVER w
FROM Sale GROUP BY SID, PID
WINDOW w AS
(PARTITION BY SID
 ORDER BY SUM(qty) DESC);
```

⇩ GROUP BY

sid	pid	cid	qty
Durham	beer	Alice Bob	10 2
Durham	chips	Bob	3
Durham	diaper	Alice	5
Raleigh	beer	Alice	2
Raleigh	diaper	Bob	100

Apply WINDOW after processing FROM, WHERE, GROUP BY, HAVING

- PARTITION defines the related set and ORDER BY orders it

E.g., for the following "row,"

Durham	beer	Alice Bob	10 2

the related list is:

Durham	beer	Alice Bob	10 2
Durham	diaper	Alice	5
Durham	chips	Bob	3

RANK example (cont'd)

sid	pid	cid	qty
Durham	beer	Alice Bob	10 2
Durham	chips	Bob	3
Durham	diaper	Alice	5
Raleigh	beer	Alice	2
Raleigh	diaper	Bob	100

```
SELECT SID, PID, SUM(qty),
       RANK() OVER w
FROM Sale GROUP BY SID, PID
WINDOW w AS
(PARTITION BY SID
 ORDER BY SUM(qty) DESC);
```

E.g., for the following "row,"

Durham	beer	Alice Bob	10 2

the related list is:

Durham	beer	Alice Bob	10 2
Durham	diaper	Alice	5
Durham	chips	Bob	3

Then, for each "row" and its related list, evaluate SELECT and return:

sid	pid	sum	rank
Durham	beer	12	1
Durham	diaper	5	2
Durham	chips	3	3
Raleigh	diaper	100	1
Raleigh	beer	2	2

Multiple windows

sid	pid	cid	qty
Durham	beer	Alice	10
		Bob	2
Durham	chips	Bob	3
Durham	diaper	Alice	5
Raleigh	beer	Alice	2
Raleigh	diaper	Bob	100

```
SELECT SID, PID, SUM(qty),
  RANK() OVER w,
  RANK() OVER w1 AS rank1
FROM Sale GROUP BY SID, PID
WINDOW w AS
  (PARTITION BY SID
   ORDER BY SUM(qty) DESC),
w1 AS
  (ORDER BY SUM(qty) DESC)
ORDER BY SID, rank;
```

No `PARTITION` means all "rows" are related to the current one

So `rank1` is the "global" rank:

sid	pid	sum	rank	rank1
Durham	beer	12	1	2
Durham	diaper	5	2	3
Durham	chips	3	3	4
Raleigh	diaper	100	1	1
Raleigh	beer	2	2	5

Summary

- Eagerly integrate data from operational sources and store a redundant copy to support OLAP
- OLAP vs. OLTP: different workload → different degree of redundancy
- SQL extensions: grouping and windowing