Virtual University of Pakistan

Data Warehousing

Lecture-9

Issues of De-normalization

Ahsan Abdullah
Assoc. Prof. & Head
Center for Agro-Informatics Research
www.nu.edu.pk/cairindex.asp
National University of Computers & Emerging Sciences, Islamabad
Email: ahsan@cluxing.com

Issues of De-normalization

Why Issues?

Issues of Denormalization

- Storage
- Performance
- Ease-of-use
- Maintenance

Industry Characteristics Master:Detail Ratios

- Health care 1:2 ratio
- Video Rental 1:3 ratio
- Retail 1:30 ratio

Storage Issues: Pre-joining Facts

- Assume 1:2 record count ratio between claim master and detail for health-care application.
- Assume 10 million members (20 million records in claim detail).
- Assume 10 byte member_ID.
- Assume 40 byte header for master and 60 byte header for detail tables.

Storage Issues: Pre-joining (Calculations)

With normalization:

Total space used = 10 x 40 + 20 x 60 = 1.6 GB

After denormalization:

Total space used = (60 + 40 – 10) x 20 = 1.8 GB

Net result is 12.5% additional space required in raw data table size for the database.

Performance Issues: Pre-joining

Consider the query “How many members were paid claims during last year?”

With normalization:

Simply count the number of records in the master table.

After denormalization:

The member_ID would be repeated, hence need a count distinct. This will cause sorting on a larger table and degraded performance.

Why Performance Issues: Pre-joining

Depending on the query, the performance actually deteriorates with denormalization! This is due to the following three reasons:

- **Forcing a sort due to count distinct.**
- **Using a table with 1.5 times header size.**
- **Using a table which is 2 times larger.**
- **Resulting in 3 times degradation in performance.**

Bottom Line: Other than 0.2 GB additional space, also keep the 0.4 GB master table.

Performance Issues: Adding redundant columns

Continuing with the previous Health-Care example, assuming a 60 byte detail table and 10 byte Sale_Person.

- Copying the Sale_Person to the detail table results in all scans taking 16% longer than previously.
- Justifiable only if significant portion of queries get benefit by accessing the denormalized detail table.
- Need to look at the cost-benefit trade-off for each denormalization decision.

Other Issues: Adding redundant columns

Other issues include, increase in table size, maintenance and loss of information:

- The size of the (largest table i.e.) transaction table increases by the size of the Sale_Person key.
  - For the example being considered, the detail table size increases from 1.2 GB to 1.32 GB.
- If the Sale_Person key changes (e.g. new 12 digit NID), then updates to be reflected all the way to transaction table.
- In the absence of 1:M relationship, column movement will actually result in loss of data.

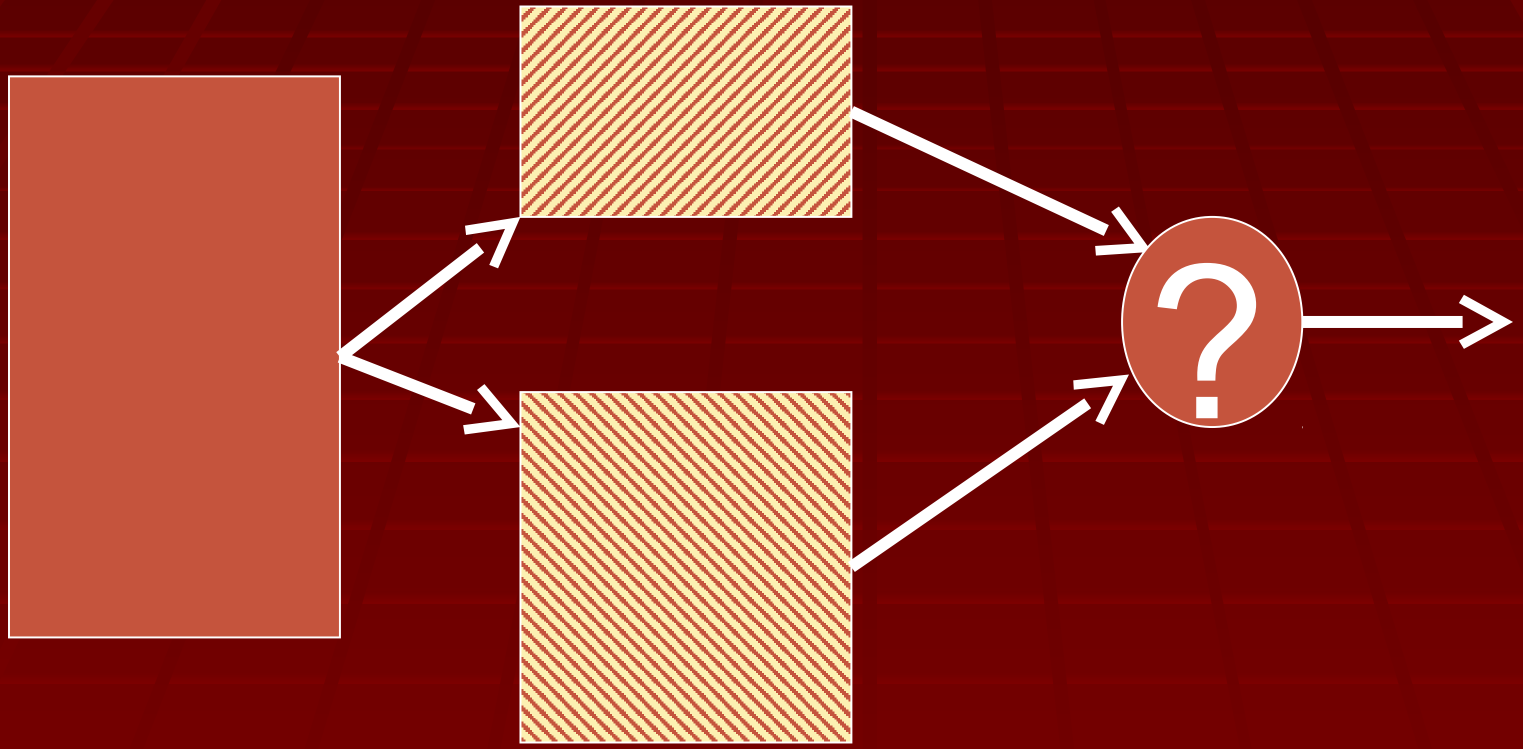
Ease of use Issues: Horizontal Splitting

Horizontal splitting is a Divide&Conquer technique that exploits parallelism. The conquer part of the technique is about combining the results.

Lets see how it works for hash based splitting/partitioning.

- Assuming uniform hashing, hash splitting supports even data distribution across all partitions in a pre-defined manner.
- However, hash based splitting is not easily reversible to eliminate the split.

Ease of use Issues: Horizontal Splitting



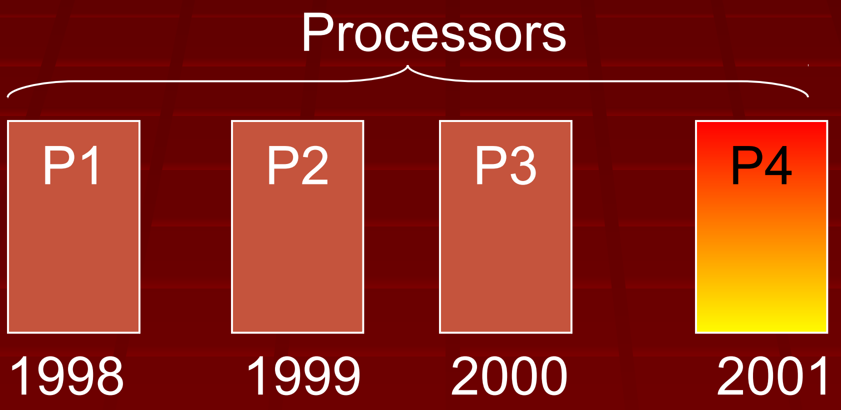
Ease of use Issues: Horizontal Splitting

- Round robin and random splitting:
  - Guarantee good data distribution.
  - Almost impossible to reverse (or undo).
  - Not pre-defined.

Ease of use Issues: Horizontal Splitting

- Range and expression splitting:
  - Can facilitate partition elimination with a smart optimizer.
  - Generally lead to "hot spots” (uneven distribution of data).

Performance Issues: Horizontal Splitting



Dramatic cancellation of airline reservations after 9/11, resulting in "hot spot"

Splitting based on year

Performance issues: Vertical Splitting Facts

Example: Consider a 100 byte header for the member table such that 20 bytes provide complete coverage for 90% of the queries.

Split the member table into two parts as follows:

1. Frequently accessed portion of table (20 bytes), and

2. Infrequently accessed portion of table (80+ bytes). Why 80+?

Note that primary key (member_id) must be present in both tables for eliminating the split.

Performance issues: Vertical Splitting Good vs. Bad

Scanning the claim table for most frequently used queries will be 500% faster with vertical splitting

Ironically, for the “infrequently” accessed queries the performance will be inferior as compared to the un-split table because of the join overhead.