DATA WAREHOUSING
CS614

© Copyright Virtual University of Pakistan

CONTENTS

Lecture 1: Introduction to Data Ware Housing–Part I 10
Learning Goals 10
1.1 Why Data Warehousing? 10
1.2 The Need for a Data Warehouse 10
1.3 Historical Overview 11
1.4 Crisis of Credibility 14
Lecture 2: Introduction to Data Ware Housing – Part II 15
Learning Goals 15
2.1 Why a Data Warehouse (DWH)? 15
2.2 What is a DWH? 18
2.3 Another view of a DWH 19
Lecture 3: Introduction to Data Ware Housing–Part III 21
Learning Goals 21
3.1 What is a Data Warehouse? 21
3.2 How is it different? 22
3.3 Why keep historical data? 24
3.4 Deviation from the PURIST approach 26
Lecture 4: Introduction to Data Ware Housing–Part IV 28
Leaning Goals 28
4.1 Typical Queries 29
4.2 Putting the pieces together 31
4.3 Why this is hard? 31
4.4 High level implementation steps 32
Lecture 5: Introduction to Data Ware Housing–Part V 34
Learning Goals 34
5.1 Types and Typical Applications of DWH 34
5.2 Telecommunications Data Warehouse 35
5.3 Typical Applications of DWH 36
5.4 Major Issues 40
Lecture 6: Normalization 41
Learning Goals 41
6.1 Normalization 41
6.2 Normalization: 1NF 43
6.3 Normalization: 2NF 44
Normalization: 3NF 46
6.5 Conclusions 48

© Copyright Virtual University of Pakistan

Lecture 07: De-Normalization 49
Leaning Goals 49
7.1 What is De-Normalization? 49
7.2 Why De-Normalization in DSS? 50
7.3 How De-Normalization improves performance? 50
7.4 Four Guidelines for De-normalization 51
7.5 Areas for Applying De-Normalization Techniques 52
7.6 Five principal De-Normalization techniques 52
Lecture 08: De-Normalization Techniques 54
Learning Goals 54
8.1 Splitting Tables 54
8.2 Pre-Joining 57
8.3 Adding Redundant Columns 58
8.4 Derived Attributes 60
Lecture 09: Issues of De-Normalization 62
Learning Goals 62
9.1 Storage Issues: Pre-joining 62
9.2 Performance Issues: Pre-joining 63
9.3 Other Issues: Adding redundant columns 65
Lecture 10: Online Analytical Processing (OLAP) 69
Learning Goals 69
10.1 DWH & OLAP 69
10.2 Analysis of last example 71
10.3 Challenges 72
10.4 OLAP: Facts & Dimensions 74
10.5 Where Does OLAP Fit In? 74
10.6 Difference between OLTP and OLAP 75
10.7 OLAP FASMI Test 76
Lecture 11: Multidimensional OLAP (MOLAP) 78
Learning Goals 78
11.1 OLAP Implementations 78
11.2 MOLAP Implementations 78
11.3 Cube Operations 80
11.4 MOLAP Evaluation 83
11.5 MOLAP Implementation Issues 84
11.6 Partitioned Cubes 84
11.7 Virtual Cubes 86

© Copyright Virtual University of Pakistan

Lecture 12: Relational OLAP (ROLAP) 87
Learning Goals 87
12.1 Why ROLAP? 87
12.2 ROLAP as a "Cube" 87
12.3 ROLAP and Space Requirement 90
12.4 ROLAP Issues 92
12.5 How to reduce summary tables? 94
12.6 Performance vs. Space Trade-off 95
12.7 HOLAP 96
12.9 DOLAP 97
Lecture 13: Dimensional Modeling (DM) 98
Learning Goals 98
13.1 The need for ER modeling? 98
13.2 Why ER Modeling has been so successful? 99
13.3 Need for DM: Un-answered Qs 99
13.4 Need for DM: The Paradox 101
13.5 How to simplify an ER data model? 103
13.6 What is DM? 103
13.7 Schemas 104
13.8 Quantifying space requirement 107
Lecture 14: Process of Dimensional Modeling 109
Learning Goals 109
14.1 The Process of Dimensional Modeling 109
14.2 Step-1: Choose the Business Process 109
14.3 Step-2: Choosing the Grain 111
14.4 The case FOR data aggregation 112
14.5 The case AGAINST data aggregation 112
14.6 Step 3: Choose Facts 115
14.7 Step 4: Choose Dimensions 115
Lecture 15: Issues of Dimensional Modeling 119
Learning Goals 119
15.1 Step-3: Classification of Aggregation Functions 119
15.2 Step-3: A Fact-less Fact Table 121
15.3 Step-4: Handling Multi-valued Dimensions? 121
15.4 Step-4: DWH Dilemma: Slowly Changing Dimensions 123
15.5 Creating a Current Value Field 126
15.6 Step-4: Pros and Cons of Handling 126

© Copyright Virtual University of Pakistan

15.7 Step-4: Junk Dimension 127
Lecture 16: Extract Transform Load (ETL) 129
Learning Goals 129
16.1 The ETL Cycle 130
16.2 Overview of Data Extraction 132
16.3 Types of Data Extraction 132
16.4 Logical Data Extraction 133
16.5 Physical Data Extraction… 134
16.6 Data Transformation 135
16.7 Significance of Data Loading Strategies 138
Three Loading Strategies 139
Lecture 17: Issues of ETL 140
Learning Goals 140
17.1 Why ETL Issues? 140
17.2 Complexity of required transformations 144
17.3 Web scrapping 146
17.4 ETL vs. ELT 147
Lecture 18: ETL Detail: Data Extraction & Transformation 149
Learning Goals 149
18.1 Extracting Changed Data 149
18.2 CDC in Modern Systems 150
18.3 CDC in Legacy Systems 151
18.4 CDC Advantages 151
18.5 Major Transformation Types 152
Lecture 19: ETL Detail: Data Cleansing 158
Learning Goals 158
19.1 Lighter Side of Dirty Data 158
19.2 3 Classes of Anomalies 160
19.3 Syntactically Dirty Data 160
19.4 Handling missing data 162
19.5 Key Based Classification of Problems 162
19.6 Automatic Data Cleansing 164
Lecture 20: Data Duplication Elimination & BSN Method 165
Learning Goals 165
20.1 Why data duplicated? 165
20.2 Problems due to data duplication 165
20.3 Formal definition & Nomenclature 168

20.4 Overview of the Basic Concept 169
20.5 Basic Sorted Neighborhood (BSN) Method 170
20.6 Limitations of BSN Method 176
Lecture 21: Introduction to Data Quality Management (DQM) 179
Learning Goals 179
21.1 What is Quality? Informally 179
21.2 What is Quality? Formally 179
21.3 Orr's Laws of Data Quality 181
21.4 Total Quality Management (TQM) 182
21.5 Cost of Data Quality Defects 183
Lecture 22: DQM: Quantifying Data Quality 186
Learning Goals 186
22.1 Data Quality Assessment Techniques 186
22.2 Data Quality Validation Techniques 189
Lecture 23: Total DQM 193
Learning Goals 193
23.1 TDQM in a DWH 193
23.2 The House of Quality 194
23.3 The House of Quality Data Model 195
23.4 How to improve Data Quality? 196
23.5 Misconceptions on Data Quality 198
Lecture 24: Need for Speed: Parallelism 201
Learning Goals 201
24.1 When to parallelize? 201
24.2 Speed-Up & Amdahl's Law 204
24.3 Parallelization OLTP Vs. DSS 205
24.4 Brief Intro to Parallel Processing 206
24.5 NUMA 206
24.6 Shared disk Vs. Shared Nothing RDBMS 210
24.7 Shared Nothing RDBMS & Partitioning 211
Lecture 25: Need for Speed: Hardware Techniques 212
Learning Goals 212
25.1 Data Parallelism: Concept 212
25.2 Data Parallelism: Ensuring Speed-UP 213
25.3 Pipelining: Speed-Up Calculation 214
25.4 Partitioning & Queries 217
25.5 Skew in Partitioning 218

Lecture 26: Conventional Indexing Techniques 220
Learning Goals 220
26.1 Need for Indexing: Speed 220
26.2 Indexing Concept 221
26.3 Conventional indexes 222
26.4 B-tree Indexing 224
26.5 Hash Based Indexing 227
Lecture 27: Need for Speed: Special Indexing Techniques 231
Learning Goals 231
27.1 Inverted index: Concept 232
27.2 Bitmap Indexes: Concept 233
27.3 Cluster Index: Concept 236
Lecture 28: Join Techniques 239
Leaning Goals 239
28.1 About Nested-Loop Join 239
28.2 Nested-Loop Join: Code 240
28.3 Nested-Loop Join: Cost Formula 242
28.4 Sort-Merge Join 243
28.5 Hash-Based Join: Working 245
Lecture 29: A Brief Introduction to Data mining (DM) 248
Learning Goals 248
29.1 What is Data Mining?: Informal 248
29.2 What is Data Mining?: Slightly Informal 249
29.3 What is Data Mining?: Formal 249
29.4 Why Data Mining? 251
29.5 Claude Shannon's info. theory 252
29.6 Data Mining is HOT! 254
29.7 How Data Mining is different? 254
29.8 Data Mining Vs. Statistics 256
Lecture 30: What Can Data Mining Do 259
Learning Goals 259
30.1 CLASSIFICATION 259
30.2 ESTIMATION 260
30.3 PREDICTION 260
30.4 MARKET BASKET ANALYSIS 261
30.5 CLUSTERING 263
30.6 DESCRIPTION 266

Lecture 31: Supervised Vs. Unsupervised Learning 269
Learning Goals 269
31.1 Data Structure in Data Mining 269
31.2 Main types of DATA MINING 270
31.3 Clustering: Min-Max Distance 271
31.4 How Clustering works? 272
31.5 Classification 275
31.6 Clustering vs. Cluster Detection 279
31.7 The K-Means Clustering 280
Lecture 32: DWH Lifecycle: Methodologies 283
Learning Goals 283
32.1 Layout the Project 283
32.2 Implementation Strategies 283
32.3 Development Methodologies 284
32.4 WHERE DO YOU START? 286
Lecture 33: DWH Implementation: Goal Driven Approach 289
Learning Goals 289
33.1 Business Dimensional Lifecycle: The Road Map Ralph Kimball's Approach 289
33.2 DWH Lifecycle: Key steps 290
33.3 DWH Lifecycle- Step 1: Project Planning 291
33.4 DWH Lifecycle- Step: 2 Requirements Definition 294
Lecture 34: DWH Implementation: Goal Driven Approach 299
Learning Goals 299
34.1 Technical Architecture Design 299
34.2 DWH Lifecycle- Step 3.1: Technology Track 301
34.3 DWH Lifecycle- Step 3.3: Analytic Applications Track 306
34.4 DW Lifecycle- Step 4: Deployment 308
34.5 DW Lifecycle- Step 5: Maintenance and Growth 309
Lecture 35: DWH Life Cycle: Pitfalls, Mistakes, Tips 311
Learning Goals 311
35.1 Five Signs of trouble 311
35.2 Eleven Possible Pitfalls 312
35.3 Top 10-Common Mistakes to Avoid 316
35.4 Top 7-Key Steps for a smooth DWH implementation… 318
35.5 Conclusions 320
Lecture 36: Course Project 321

Learning Goals 321
36.1 Part-I 321
36.2 Part-II: Implementation Life-Cycle 322
36.3 Contents of Project Reports 326
Lecture 37: Case Study: Agri-Data Warehouse 330
Learning Goals 330
37.2 Major Players 331
37.3 Economic Threshold Level ETL_A 332
37.4 The need: IT in Agriculture 333
37.5 A simplified ERD of the Agri System 337
Lecture 38: Case Study: Agri-Data Warehouse 340
Learning Goals 340
38.1 Step 6: Data Acquisition & Cleansing 340
38.2 Step-6: Why the issues? 341
38.3 Decision Support using Agri-DWH 344
38.4 Conclusions & lessons learnt 347
Lecture 39. Web Warehousing: An introduction 348
Leaning Goals 348
39.1 Reasons for web warehousing 350
39.2 Web searching 350
39.3 Drawbacks of traditional web searches 351
39.4 How Client Server interactions take place? 353
39.5 Low level web traffic Analysis 355
39.6 High level web traffic analysis 355
39.7 Web traffic record: Log files 357
Lecture 40: Web Warehousing: Issues 360
Learning Goals 360
40.1 Web Warehouse Dimensions 361
40.2 Clickstream 362
40.3 Proxy servers 368
40.4 Some New Challenges 370
Lab Lecture-1: Data Transfer Service (DTS) 372
Lab Lecture-2: Lab Data Set 396
Lab Lecture-3: Extracting Data Using Wizard 410
Lab Lecture-4: Data Profiling 436
Lab Lecture-5: Data Transformation & Standardization 456

Lecture 1: Introduction to Data Ware Housing–Part I

A data warehouse is a type of data management system

Learning Goals

- The world is changing (actually changed), either change or be left behind.
- Missing the opportunities or going in the wrong direction has prevented us from growing.
- What is the right direction?
- Harnessing the data, in a knowledge driven economy.

1.1 Why Data Warehousing?

The world economy has moved form the industrial age into information driven knowledge economy. The information age is characterized by the computer technology, modern communication technology and Internet technology; all are popular in the world today. Governments around the globe have realized potential of information, as a "multi-factor" in the development of their economy, which not only creates wealth for the society, but also affects the future of the country. Thus, many countries in the world have placed the modern information technology into their strategic plans. They regard it as the most important strategic resource for the development their society, and are trying their best to reach and occupy the peak of the modern information driven knowledge economy.

What is the right direction?

Ever since the IT revolution that happened more than a decade ago every government has been trying and tried to increase our software exports. But have persistently failed to get the desired results. I happened to meet a gentleman who got venture capital of several million US dollars and I asked him why our software export has not gone up? His answer was simple, "we have been investing in outgoing or outdated tools and technologies". We have also been just following India, without thinking for a moment, what India is today, started maybe a decade ago. So my next question was "what should we be doing today?" His answer was "we have captured and stored data for a long time, now it is time to explore and make use of that data". There is a saying that "a fool and his money are soon parted", since that gentleman was rich and is still rich, hence he does qualify to be a wise man, and his words of wisdom to be paid attention to.

1.2 The Need for a Data Warehouse

"Drowning in data and starving for information"

"Knowledge is power, Intelligence is absolute power!"

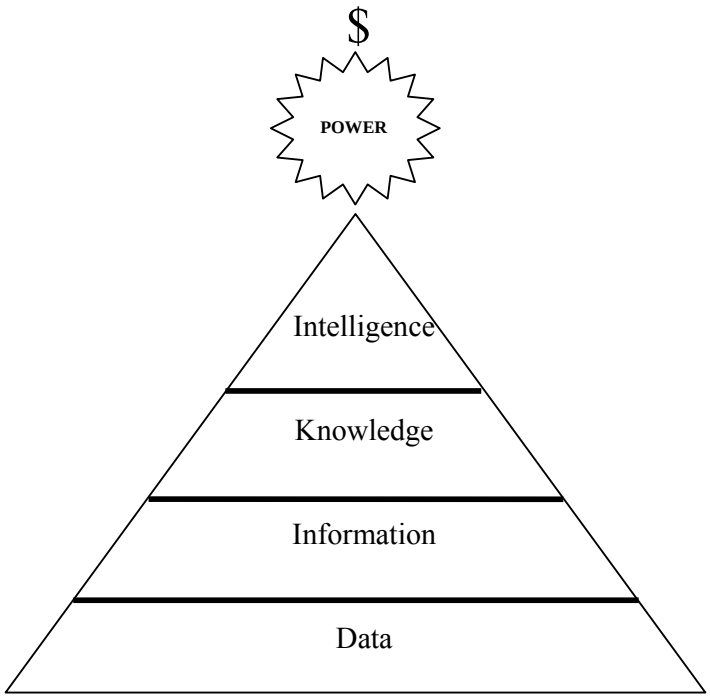


Figure-1.1: Relationship between Data, Information, Knowledge & Intelligence

Data is defined as numerical or other facts represented or recorded in a form suitable for processing by computers. Data is often the record or result of a transaction or an operation that involves modification of the contents of a database or insertion of rows in tables. Information in its simplest form is processed data that is meaningful. By processing, summarizing or analyzing data, organizations create information. For example the current balance, items sold, money made etc. This information should be designed to increase the knowledge of the individual, therefore, ultimately being tailored to the needs of the recipient. Information is processed data so that it becomes useful and provides answers to questions such as "who", "what", "where", and "when". Knowledge, on the other hand is an application of information and data, and gives an insight by answering the "how" questions. Knowledge is also the understanding gained through experience or study. Intelligence is appreciation of "why", and finally wisdom (not shown in the figure-1.1) is the application of intelligence and experience toward the attainment of common goals, and wise people are powerful. Remember knowledge is power.

1.3 Historical Overview

It is interesting to note that DSS (Decision Support System) processing as we know it today has reached this point after a long and complex evolution, and yet it continues to evolve. The origin of DSS goes back to the very early days of computers.

Figure-1.2 shows the historical overview or the evolution of data processing from the early 1960s up to 1980s. In the early 1960s, the world of computation consisted of exclusive applications that were executed on master files. The applications featured reports and programs, using languages like COBOL and punched cards i.e. the COBOLian era. The master files were stored on magnetic tapes, which were good for storing a large volume of data cheaply, but had the drawback of needing to be accessed sequentially, and being very unreliable (ask your system administrator even today about

tape backup reliability). Experience showed that for a single pass of a magnetic tape that scanned 100% of the records, only 5% of the records, sometimes even less were actually required. In addition, reading an entire tape could take anywhere from 20-30 minutes, depending on the data and the processing required.

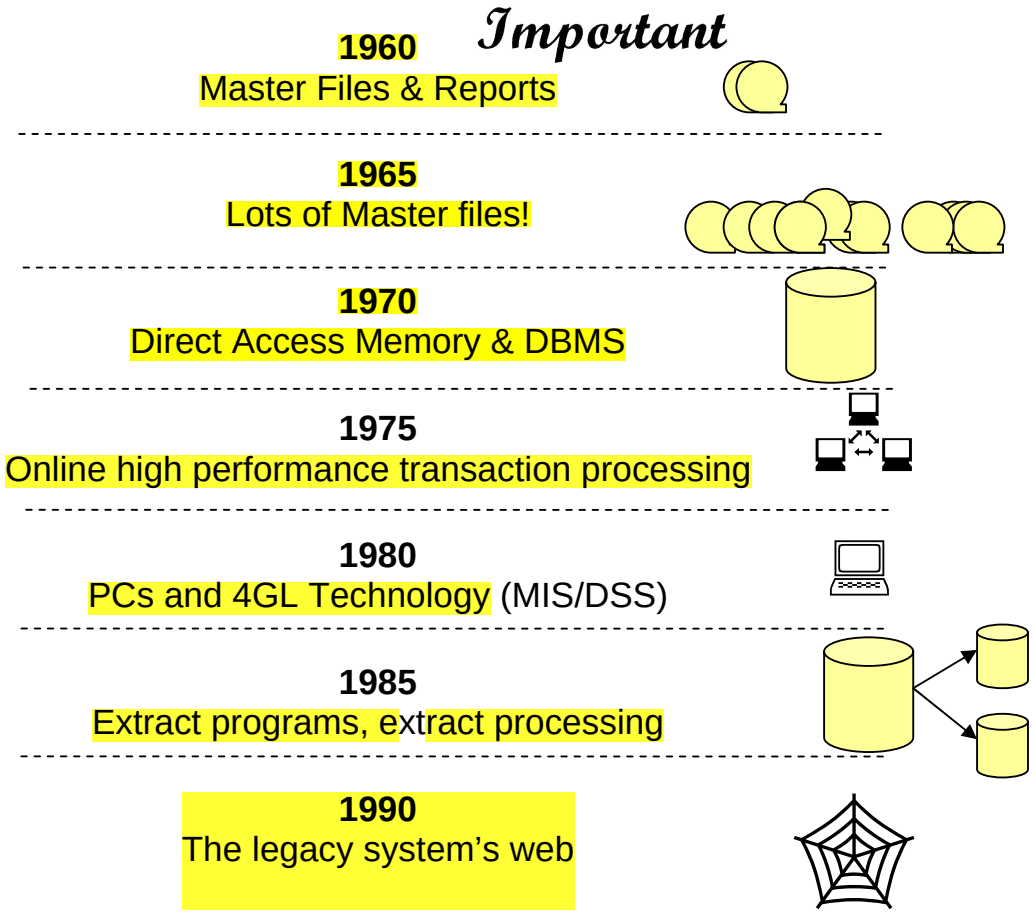


Figure-1.2: Historical Overview of use of Computers for Data Processing

Around the mid-1960s, the growth of master files and magnetic tapes exploded. Soon master files were used at every computer installation. This growth in usage of master files, resulted in huge amounts of redundant data. The spreading of master files and massive redundancy of data presented some very serious problems, such as:

- Data coherency i.e. the need to synchronize data upon update.
- Program maintenance complexity.
- Program development complexity.
- Requirement of additional hardware to support many tapes.

In a nut-shell, the inherent problems of master files because of the limitations of the medium used started to become a bottleneck. If we had continued to use only the magnetic tapes, we may not have had an Information revolution! Consequently, there would have never been large, fast MIS (Management Information Systems) systems, ATM systems, Airline Flight reservation systems, maybe not even Internet as we know it. As one of my teachers very rightly said, "every problem is an opportunity" therefore, the ability to store and manage data on diverse media (other than magnetic tapes) opened up

the way for a very different and more powerful type of processing i.e. bringing the IT and the business user together as never before.

The advent of DASD

By 1970s, a new technology for the storage and access of data had had been introduced. The 1970s saw the advent of disk storage, or DASD (Direct Access Storage Device). Disk storage was fundamentally different from magnetic tape storage in the sense that data could be accessed directly on DASD i.e. non-sequentially. There was no need to go all the way through records 1, 2, 3, . . . k so as to reach the record k + 1. Once the address of record k + 1 was known, it was a simple matter to go to record k + 1 directly. Furthermore, the time required to go to record k + 1 was significantly less than the time required to scan a magnetic tape. Actually it took milliseconds to locate a record on a DASD i.e. orders of magnitude better performance than the magnetic tape.

With DASD came a new type of system software known as a DBMS (Data Base Management System). The purpose of the DBMS was to facilitate the programmer to store and access data on DASD. In addition, the DBMS took care of such tasks as storing data on DASD, indexing data, accessing it etc. With the winning combination of DASD and DBMS came a technological solution to the problems of magnetic tape based master files. When we look back at the mess that was created by master files and the mountains of redundant data aggregated on them, it is no wonder that database is defined as a single source of data for all processing and a prelude to a data warehouse i.e. “a single source of truth”.

PC & 4GL

By the 1980s, more and new hardware/software, such as PCs and 4GLs (4th Generation Languages) began to come out. The end user began to take up roles previously unimagined i.e. directly controlling data and systems, outside the domain of the classical data center. With PCs and 4GL technology the notion dawned that more could be done with data than just servicing high-performance online transaction processing i.e. MIS (Management Information Systems) could be developed to run individual database applications for managerial decision making i.e. forefathers of today’s DSS. Previously, data and IT were used exclusively to direct detailed operational decisions. The combination of PC and 4GL introduced the notion of a new paradigm i.e. a single database that could serve both operational high performance transaction processing and (limited) DSS, analytical processing, all at the same time.

The extract program

Shortly after the advent of massive online high-performance transactions, an innocent looking program called "extract" processing, began to show up.

The extract program was the simplest of all programs of its time. It scanned a file or database, used some criteria for selection, and, upon finding qualified data, transported the data into another file or database. Soon the extract program became very attractive, and flooded the information processing environment.

The spider web

Figure 1.2 shows that a "spider web" of extract processing programs began to form. First, there were extracts. Then there were extracts of extracts, then extracts of extracts of extracts, and it went on. It was common for large companies to be doing tens of

thousands of extracts per day.

This pattern of extract processing across the organization soon became a routine activity, and even a name was coined for it. Extract processing gone out of control produced what was called the "naturally evolving architecture". Such architectures occurred when an organization had a relaxed approach to handling the whole process of hardware and software architecture. The larger and more mature the organization; the worse was the problems of the naturally evolving architecture.

Taken jointly, the extract programs or naturally evolving systems formed a spider web, also called "legacy systems" architecture.

1.4 Crisis of Credibility

What is the financial health of my company?

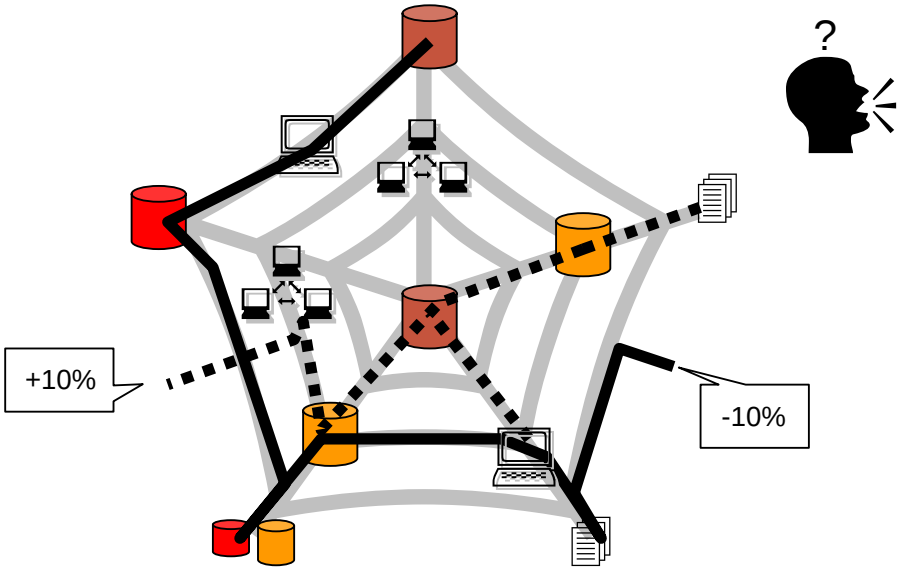


Figure-1.3: Crisis of Credibility: Who is right?

Consider the CEO of an organization who is interested in the financial health of his company. He asks the relevant departments to work on it and present the results. The organization is maintaining different legacy systems, employs different extract programs and uses different external data sources. As a consequence, Department-A which uses a different set of data sources, external reports etc. as compared to Department-B (as shown in Figure-1.3) comes with a different answer (say) sales up by 10%, as compared to the Department-B i.e. sales down by 10%. Because Department-B used another set of operational systems, data bases and external data sources. When CEO receives the two reports, he does not know what to do. CEO is faced with the option of making decisions based on politics and personalities i.e. very subjective and non-scientific. This is a typical example of the crisis in credibility in the naturally evolving architecture. The question is which group is right? Going with either of the findings could spell disaster, if the finding turns about to be incorrect. Hence the second important question, result of which group is credible? This is very hard to judge, since neither had malicious intensions but both got a different view of the business using different sources.

Masters

Lecture 2: Introduction to Data Ware Housing – Part II

Learning Goals

- Data recording and storage is growing.
- History is excellent predictor of the future.
- Gives total view of the organization.
- Data recording and storage is growing.
- Intelligent decision-support is required for decision-making.

2.1 Why a Data Warehouse (DWH)?

Moore's law on increase in performance of CPUs and decrease in cost has been surpassed by the increase in storage space and decrease in cost. Meaning, it is true that the cost of CPUs is going down and the performance is going up, but this is applicable at a higher rate to storage space and cost i.e. more and more cheap storage space is becoming available as compared to fast CPUs.

As you would have experienced, when you (or your father's) briefcase seems to be small as compared to the contents carried in it, it seems a good idea to buy a new and larger briefcase. However, after sometime the new briefcase too seems to be small for the contents carried. On the practical side, it has been noted that the amount of data recorded in an organization doubles every year and this is an exponential increase.

Reason-1: Data Sets are growing

MCQs

How Much Data is that?		
1 MB	2^{20} or 10^6 bytes	Small novel – 31/2 Disk
1 GB	2^{30} or 10^9 bytes	Paper rims that could fill the back of a pickup van
1 TB	2^{40} or 10^{12} bytes	50,000 trees chopped and converted into paper and printed
2 PB	1 PB = 2^{50} or 10^{15} bytes	Academic research libraries across the U.S.
5 EB	1 EB = 2^{60} or 10^{18} bytes	All words ever spoken by human beings

Table-2.1: Quantifying size of data

- Size of Data Sets are going up ↑.
- Cost of data storage is coming down ↓.
  - Total hardware and software cost to store and manage 1 Mbyte of data
    - 1990: ~ $15
    - 2002: ~ ¢15 (Down 100 times)
    - By 2007: < ¢1 (Down 150 times)

- A Few Examples
  - WalMart: 24 TB (Tera Byte)
  - France Telecom: ~ 100 TB
  - CERN: Up to 20 PB by 2006 (Peta Byte)
  - Stanford Linear Accelerator Center (SLAC): 500TB

A Ware House of Data is NOT a Data Warehouse

Someone says I have a data set of size 1 GB so I have a DWH can you beat this?
Someone else says, I have a data set of size 100 GB, can you beat this?
Someone else says, I have a 1 TB data set, who can beat this?

Who has a data warehouse? Not enough information, it is much more than just the size, it is a whole concept, it is NOT a shrink wrapped solution, it evolves. A company may have a TB of data and not have a data warehouse; while on the other hand, a company may have 500 GB of data and have a fully functional data warehouse.

Size is NOT Everything

History is excellent predictor of the future
Secondly as I mentioned earlier the data warehouse has the historical data. And one thing that we have learned by using information is that, "past is the best predictor of the future". You use historical data, because it gives you an insight into how the environment is changing. Also you must have heard that "history repeats itself", however this repetition of history is not likely to be constant for all businesses or all events. Note that you just can't use the historical data to predict the future; you have to have to bring your own insight and experience to interpret how the environment is changing in order to predict the future accurately and meaningfully.

Gives total view of the organization
So why would you want data warehouse in your organization? First of all a data warehouse gives a total view of an organization. If you look at the operational system i.e. the databases in most environments, the databases are designed around different lines of business. Consider the case of a Bank; a bank will typically have current accounts and savings accounts, foreign currency account etc. The bank will have an MIS system for leasing, and another system for managing credit cards and another system for every different kind of business they are in. However, nowhere they have the total view of the environment from the customer's perspective. The reason being, transaction processing systems are typically designed around functional areas, within a business environment. For good decision making you should be able to integrate the data across the organization so as to cross the LoB (Line of Business). So the idea here is to give the total view of the organization especially from a customer's perspective within the data warehouse, as shown in Figure-2.1

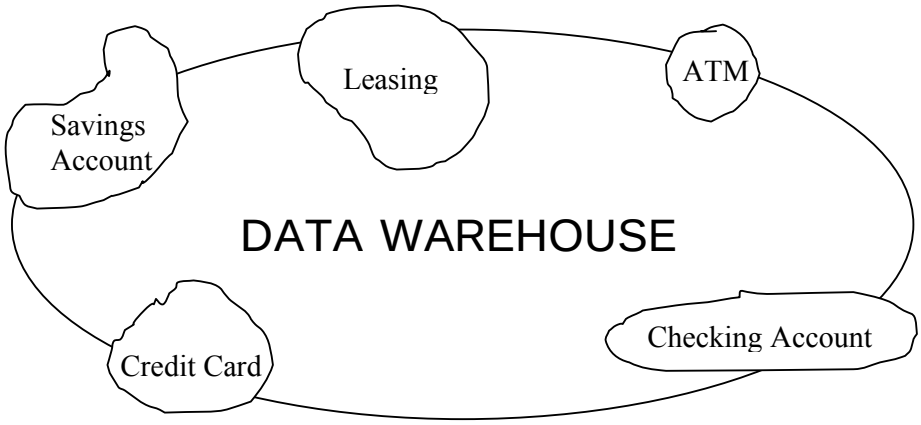


Figure-2.1: A Data Warehouse crosses the LoB

Intelligent decision-support is required for decision-making

Consider a bank which is losing customers, for reasons not known. However, one thing is for sure that the bank is losing business because of lost customers. Therefore, it is important, actually critical to understand which customers have left and why they have left. This will give you the ability to predict going forward (in time), to identify which customers will leave you (i.e. the bank). We are going to talk about this in the course using data mining algorithms, like clustering, classification, regression analysis etc. However, this being another example of using historical data to predict the future. So I can predict today, which customers will leave me in the next 3 months before they even leave. There can be, and there are whole courses on data mining, but we will just have an applied overview of data mining in this course.

Reason-2: Businesses demand intelligence

- Complex questions from integrated data.
- "Intelligent Enterprise"

DBMS Approach	Intelligent Enterprise
List of all items that were sold last month? List of all items purchased by Khizar? The total sales of the last month grouped by branch? How many sales transactions occurred during the month of January?	Which items sell together? Which items to stock? Where and how to place the items? What discounts to offer? How best to target customers to increase sales at a branch? Which customers are most likely to respond to my next promotional campaign, and why?

Table-2.2: Comparison of queries

Let's take a close look at the typical queries for a DBMS. They are either about listing the contents of tables or running aggregates of values i.e. rather simple and straightforward

queries and fairly easy to program. The queries follow rather pre-defined paths into the database and are unlikely to come up with something new or abnormal.

Reason-3: Businesses want much more…

1. What happened?
2. Why it happened?
3. What will happen?
4. What is happening?
5. What do you want to happen?

These questions primarily point to what is called as the different stages of a Data Warehouse i.e. starting from the first stage, and going all the way to stage 5. The first stage is not actually a data warehouse, but a pure batch processing system. Note that as the stages evolve the amount of batching processing decreases, this being maximum in the first stage and minimum in the last or 5th stage. At the same time the amount of ad-hoc query processing increases. Finally in the most developed stage there is a high level of event based triggering. As the system moves from stage-1 to stage-5 it becomes what is called as an active data warehouse.

2.2 What is a DWH?

A complete repository of historical corporate data extracted from transaction systems that is available for ad-hoc access by knowledge workers

The other key points in this standard definition that I have also underlined and listed below are:

Complete repository

- All the data is present from all the branches/outlets of the business.
- Even the archived data may be brought online.
- Data from arcane and old systems is also brought online.

Transaction System

- Management Information System (MIS)
- Could be typed sheets (NOT transaction system)

Ad-Hoc access

- Does not have a certain predefined database access pattern.
- Queries not known in advance.
- Difficult to write SQL in advance.

Knowledge workers

- Typically NOT IT literate (Executives, Analysts, Managers).
- NOT clerical workers.
- Decision makers.

The users of data warehouse are knowledge workers in other words they are decision makers in the organization. They are not the clerical people entering the data or

overseeing the transactions etc or doing programming or performing system design/analysis. These are really decision makers in the organization like General Manager Marketing, or Executive Director or CEO (Chief Operating Officer). Typically those decision makers are people in areas like marketing, finance and strategic planning etc.

Completeness: There is a misnomer here, about completeness. As per the standard definition a data warehouse is a complete repository of corporate data. The reality is that it can never be complete. We will discuss this in detail very shortly.

Transaction System: Unlike databases where data is directly entered, the input to the data warehouse can come from OLTP or transactional systems or other third party databases. This is not a rule, the data could come from typed or even hand filled sheets, as was the case for the census data warehouse.

Ad-Hoc access: It dose not have a certain repeatable pattern and it's not known in advance. Consider financial transactions like a bank deposit, you know exactly what records will be inserted deleted or updated. That's in OLTP system and in ERP system. But in a data warehouse there are really no fixed patterns. Say the marketing person, just sits down and thinks about what questions he/she has about customers and there behaviors and so on and they are typically using some tool to generate SQL dynamically and then that SQL gets executed and that you don't know in advance.

Although there may be some patterns of queries, but they are really not very predictable and the query patterns may change over time. Hence there are no predefined access paths into the database. That's why relational databases are so important for the data warehouse, because relational databases allow you to navigate the data in any direction that is appropriate using the primary, foreign key structure within the data model. Meaning, using a data warehouse, does not implies that we just forget about databases.

2.3 Another view of a DWH

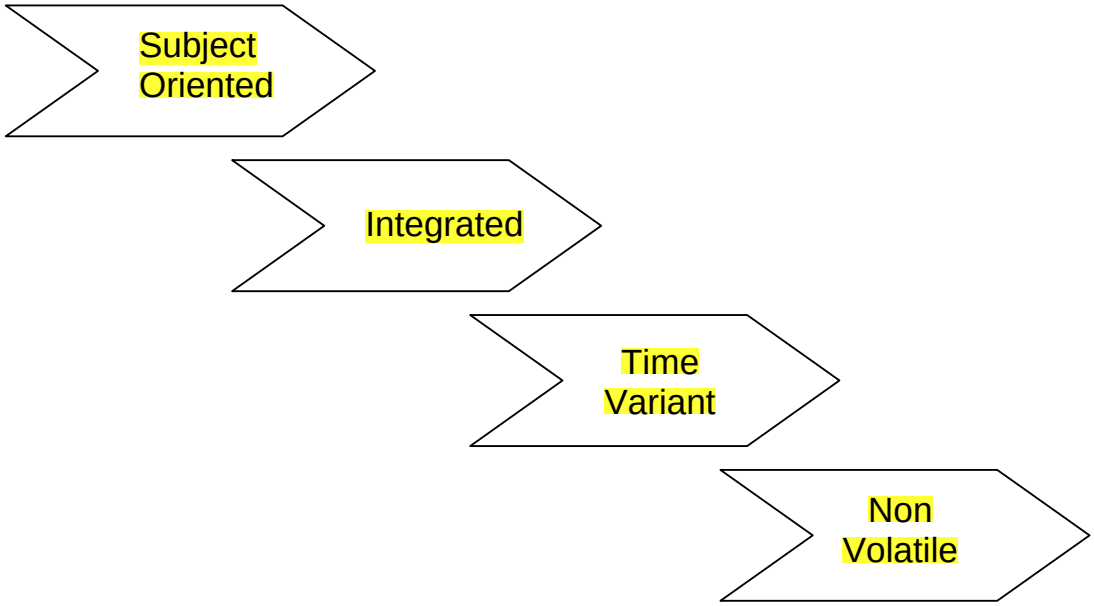


Figure-2.2: Another view of a Data Warehouse

Subject oriented: The goal of data in the data warehouse is to improve decision making, planning, and control of the major subjects of enterprises such as customer, products, regions, in contrast to OLTP applications that are organized around the work-flows of the company.

Integrated: The data in the data warehouse is loaded from different sources that store the data in different formats and focus on different aspects of the subject. The data has to be checked, cleansed and transformed into a unified format to allow easy and fast access.

Time variant: Time variant records are records that are created as of some moment in time. Every record in the data warehouse has some form of time variancy associated with it. In an OLTP system, the contents change with time i.e. updated such as bank account balance or mobile phone balance, but in a warehouse as the data is loaded; the moment usually becomes its time stamp.

Non-volatile: Unlike OLTP systems, after inserting data in the data warehouse it is neither changed nor removed. The only exceptions are when *false* or incorrect data gets inserted erroneously or the capacity of the data warehouse exceeded and archiving becomes necessary.

Masters

Lecture No. 03
Lecture 3: Introduction to Data Ware Housing–Part III

Learning Goals

It is a blend of many technologies, the basic concept being:

- Take all data from different operational systems.
- If necessary, add relevant data from industry.
- Transform all data and bring into a uniform format.
- Integrate all data as a single entity.
- Store data in a format supporting easy access for decision support.
- Create performance enhancing indices.
- Implement performance enhancement joins.
- Run ad-hoc queries with low selectivity.

3.1 What is a Data Warehouse?

A Data Warehouse is not something shrink-wrapped i.e. you take a set of CDs and install into a box and soon you have a Data Warehouse up and running. A Data Warehouse evolves over time, you don't buy it. Basically it is about taking/collecting data from different heterogeneous sources. Heterogeneous means not only the operating system is different but so is the underlying file format, different databases, and even with same database systems different representations for the same entity. This could be anything from different columns names to different data types for the same entity.

Companies collect and record their own operational data, but at the same time they also use reference data obtained from external sources such as codes, prices etc. This is not the only external data, but customer lists with their contact information are also obtained from external sources. Therefore, all this external data is also added to the data warehouse.

As mentioned earlier, even the data collected and obtained from within the company is not standard for a host of different reasons. For example, different operational systems being used in the company were developed by different vendors over a period of time, and there is no or minimal evenness in data representation etc. When that is the state of affairs (and is normal) within a company, then there is no control on the quality of data obtained from external sources. Hence all the data has to be transformed into a uniform format, standardized and integrated before it can go into the data warehouse.

In a decision support environment, the end user i.e. the decision maker is interested in the big picture. Typical DSS queries do not involve using a primary key or asking questions about a particular customer or account. DSS queries deal with number of variables spanning across number of tables (i.e. join operations) and looking at lots of historical data. As a result large number of records are processed and retrieved. For such a case, specialized or different database architectures/topologies are required, such as the star schema. We will cover this in detail in the relevant lecture.

Recall that a B-Tree is a data structure that supports dictionary operations. In the context of a database, a B-Tree is used as an index that provides access to records without actually scanning the entire table. However, for very large databases the corresponding B-Trees becomes very large. Typically the node of a B-Tree is stored in a memory block, and traversing a B-Tree involves O(log n) page faults. This is highly undesirable, because by default the height of the B-Tree would be very large for very large data bases. Therefore, new and unique indexing techniques are required in the DWH or DSS environment, such as bitmapped indexes or cluster index etc. In some cases the designers want such powerful indexing techniques, that the queries are satisfied from the indexes without going to the fact tables.

In typical OLTP environments, the size of tables are relatively small, and the rows of interest are also very small, as the queries are confined to specifics. Hence traditional joins such as nested-loop join of quadratic time complexity does not hurt the performance i.e. time to get the answer. However, for very large databases when the table sizes are in millions and the rows of interest are also in hundreds of thousands, nested-loop join becomes a bottle neck and is hardly used. Therefore, new and unique forms of joins are required such as sort-merge join or hash based join etc.

Run Ad-Hoc queries with low Selectivity

Have already explained what is meant by ad-hoc queries. A little bit about selectivity is in order. Selectivity is the ratio between the number of unique values in a column divided by the total number of values in that column. For example the selectivity of the gender column is 50%, assuming gender of all customers is known. If there are N records in a table, then the selectivity of the primary key column is 1/N. Note that a query consists of retrieving records based on a a combination of different columns, hence the choice of columns determine the selectivity of the query i.e. the number of records retrieved divided by the total number of records present in the table.

In an OLTP (On-Line Transaction Processing) or MIS (Management Information System) environment, the queries are typically Primary Key (PK) based, hence the number of records retrieved is not more than a hundred rows. Hence the selectivity is very high. For a Data Warehouse (DWH) environment, we are interested in the big picture and have queries that are not very specific in nature and hardly use a PK. As a result hundreds of thousands of records (or rows) are retrieved from very large tables. Thus the ratio of records retrieved to the total number of records present is high, and hence the selectivity is low.

imp

record retrieved $\propto$ $\frac{\text{total no of record}}{\text{selectivity}}$

3.2 How is it different?

Decision making is Ad-Hoc

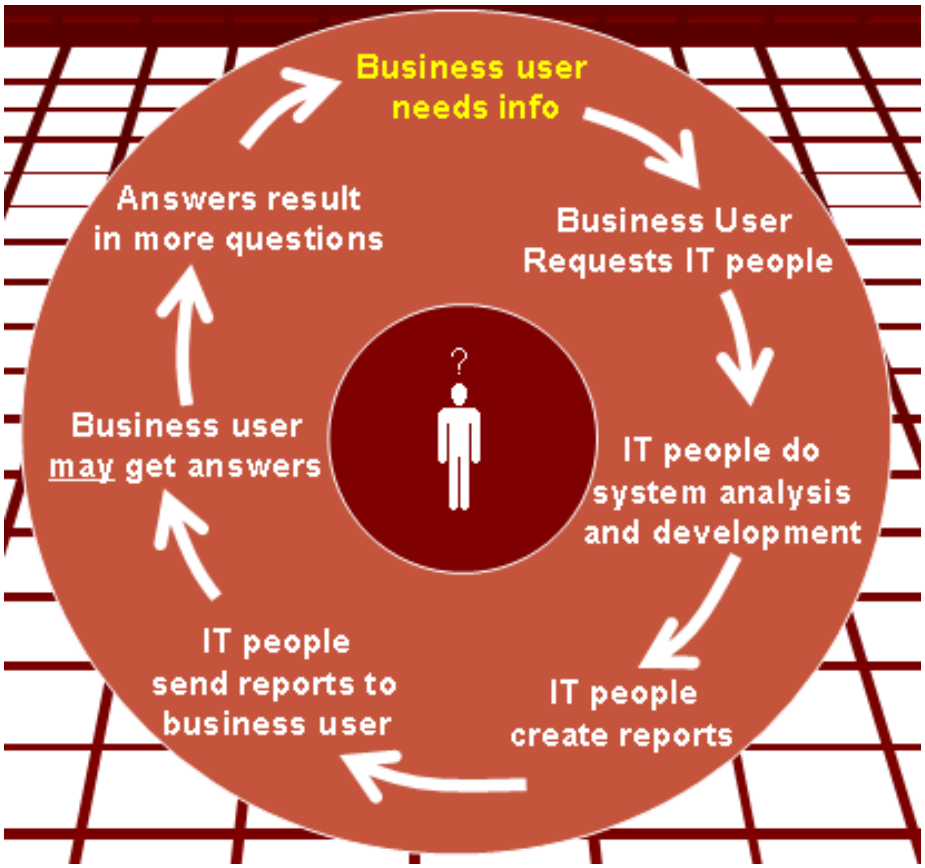


Figure-3.1: Running in circles

Consider a decision maker or a business user who wants some of his questions to be answered. He/she sets a meeting with the IT people, and explains the requirements. The IT people go over the cycle of system analysis and design, that takes anywhere from couple of weeks to couple of months and they finally design and develop the system. Happy and proud with their achievement the IT people go to the business user with the reporting system or MIS system. After a learning curve the business users spends some time with the brand new system, and may get some answers to the required questions. But then those answers results in more questions. The business user has no choice to meet the IT people with a new set of requirements. The business user is frustrated that his questions are not getting answered, while the IT people are frustrated that the business user always changes the requirements. Both are correct in their frustration.

Different patterns of hardware utilization

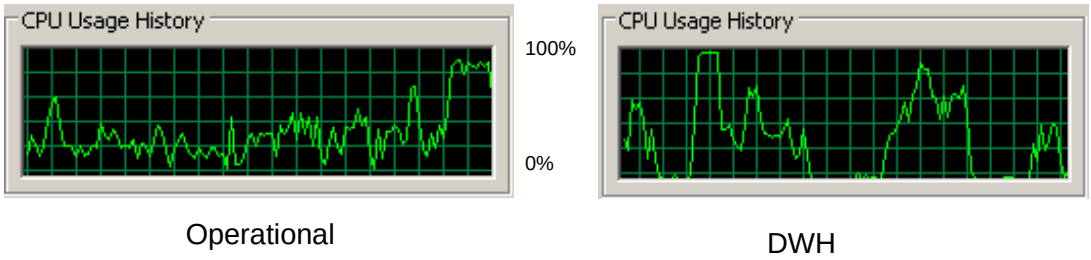


Figure-3.2: Different patterns of CPU Usage

Although there are peaks and valleys in the operational processing, but ultimately there is relatively static pattern of utilization. There is an essentially different pattern of hardware utilization in the data warehouse environment i.e. a binary pattern of utilization, either the hardware is utilized fully or not at all. Calculating a mean utilization for a DWH is not a meaningful activity. Therefore, trying to mix the two environments is a recipe for disaster. You can optimize the machine for the performance of one type of application, not for both.

Bus vs. Train Analogy

Consider the analogy of a bus and train. I believe you can find dozens of buses operating between Lahore and Rawalpindi almost every 30 minutes. As a consequence, literally there are buses moving between Lahore and Rawalpindi almost continuously through out the day. But how many times a dedicated train moves between the two cities? Only twice a day and carries a bulk of passengers and cargo. Binary operation i.e. either traveling or not. The train can NOT be optimized for every 30-min travel, it will never fill to capacity and run into loss. A bus can not be optimized to travel only twice, it will stand idle and passengers would take vans etc. Bottom line: Two modes of transportation, can not be interchanged.

Combines historical & Operational Data

- Don't do data entry into a DWH, OLTP or ERP are the source systems.
- OLTP systems don't keep history, cant get balance statement more than a year old.
- DWH keep historical data, even of bygone customers. Why?
- In the context of bank, want to know why the customer left?
- What were the events that led to his/her leaving? Why?
- Customer retention.

3.3 Why keep historical data?

The data warehouse is different because, again it's not a database you do data entry. You are actually collecting data from the operational systems and loading into the DWH. So the transactional processing systems like the ERP system are the source systems for the data warehouse. You feed the data into the data warehouse. And the data warehouse typically collects data over a period of time. So if you look at your transactional processing OLTP systems, normally such systems don't keep very much history. Normally if a customer leaves or expired, the OLTP systems typically purge the data associated with that customer and all the transactions off the database after some amount of time. So normally once a year most business will purge the database of all the old customers and old transactions. In the data warehouse we save the historical data. Because you don't need historical data to do business today, but you do need the historical data to understand patterns of business usage to do business tomorrow, such why a customer left?

How much History?

- Depends on:
  - Industry.
  - Cost of storing historical data.
  - Economic value of historical data.
- Industries and history
  - **Telecomm** calls are much much more as compared to bank transactions- 18 months of historical data.
  - **Retailers** interested in analyzing yearly seasonal patterns- 65 weeks of historical data.
  - **Insurance** companies want to do actuary analysis, use the historical data in order to predict risk- 7 years of historical data.

MCQs

Hence, a DWH NOT a complete repository of data

How back do you look historically? It really depends a lot on the industry. Typically it's an economic equation. How far back depends on how much dose it cost to store that extra years work of data and what is it's economic value? So for example in financial organizations, they typically store at least 3 years of data going backward. Again it's typical. It's not a hard and fast rule.

In a telecommunications company, for example, typically around 18 months of data is stored. Because there are a lot more call details records then there are deposits and withdrawals from a bank account so the storage period is less, as one can not afford to store as much of it typically. Another important point is, the further back in history you store the data, the less value it has normally. Most of the times, most of the access into the data is within that last 3 months to 6 months. That's the most predictive data.

In retail business, retailers typically store at least 65 weeks of data. Why do they do that? Because they want to be able to look at this season's selling history to last season's selling history. For example, if it is Eid buying season, I want to look at the transit-buying this Eid and compare it with the year ago. Which means I need 65 weeks in order to get year going back, actually more then a year? It's a year and a season. So 13 weeks are additionally added to do the analysis. So it really depends a lot on the industry. But normally you expect at least 13 months.

Economic value of data Vs. Storage cost & Data Warehouse a complete repository of data?

This raises an interesting question, do we decide about storage of historical data using only time, or consider space also, or both?

Usually (but not always) periodic or batch updates rather than real-time.

- The boundary is blurring for active data warehousing.
- For an ATM, if update not in real-time, then lot of real trouble.
- DWH is for strategic decision making based on historical data. Wont hurt if transactions of last one hour/day are absent.
- Rate of update depends on:
  - Volume of data,
  - Nature of business,
  - Cost of keeping historical data,
  - Benefit of keeping historical data.

Important

It's also true that in the traditional data warehouse the data acquisition is done on periodic or batch based, rather then in real time. So think again about ATM system, when I put my ATM card and make a withdrawal, the transactions are happening in real time, because if they don't the bank can get into trouble. Someone can withdraw more money then they had in their account! Obviously that is not acceptable. So in an online transaction processing (OLTP) system, the records are updated, deleted and inserted in real-time as the business events take place, as the data entry takes place, as the point of sales system at a super market captures the sales data and inserts into the database.

Imp

In a traditional data warehouse that is not true, because the traditional data warehouse is for strategic decision-making not for running day to day business. And for strategic decision making, I don't need to know the last hour's worth of ATM deposits. Because strategic decisions take the long term perspective. For this reason and for efficiency reasons normally what happens is that in the data warehouse you update on some predefined schedule basis. May be it's once a month, maybe it's once a weak, maybe it's even once every night. It depends on the volume of data you are working with, and how important the timings of the data are and so on.

3.4 Deviation from the PURIST approach

Let me first explain what/who a purist is. A purist is an idealist or traditionalist who wants everything to be done by the book or the old arcane ways (only he/she knows), in short he/she is not pragmatic or realist. Because the purist wants everything perfect, so he/she has good excuses of doing nothing, as it is not a perfect world. When automobiles were first invented, it was the purists who said that the automobiles will fail, as they scare the horses. As Iqbal very rightly said "*Aina no sa durna Tarzay Kuhan Pay Arna…*"

As data warehouses become main-stream and the corresponding technology also becomes mainstream technology, some traditional attributes are being deviated in order to meet the increasing demands of the user's. We have already discussed and reconciled with the fact that a data warehouse is NOT the complete repository of data. The other most noticeable deviations being time variance and non-volatility.

<u>Deviation from Time Variance and Nonvolatility</u>

As the size of data warehouse grows over time (e.g., in terabytes), reloading and appending data can become a very tedious and time consuming task. Furthermore, as business users get the "hang of it" they start demanding that more up-to-date data be

available in the data warehouse. Therefore, instead of sticking to the traditional data warehouse characteristic of keeping the data nonvolatile and time variant, new data is being added to the data warehouse on a daily basis, if not on a real-time basis and at the same time historical data removed to make room for the "fresh" data. Thus, new approaches are being made to tackle this task. Two possible methods are as follows:

- Perform hourly/daily batch updates from shadow tables or log files. Transformation rules are executed during the loading process. Thus, when the data reaches the target data warehouse database, it is already transformed, cleansed and summarized.

Perform real-time updates from shadow tables or log files. Again, transformation rules are executed during the loading process. Instead of batch updates, this takes place on a per transaction basis that meets certain business selection criteria.

Masters Lecture 4: Introduction to Data Ware Housing–Part IV

Leaning Goals

- Decision makers typically don't work 24 hrs a day and 7 days a week. An ATM system does.
- Once decision makers' start using the DWH, and start reaping the benefits, they start liking it…
- Start using the DWH more often, till want it available 100% of the time.
- For business across the globe, 50% of the world may be sleeping at any one time, but the businesses are up 100% of the time.
- 100% availability not a trivial task, need to take into loading strategies, refresh rates etc.

Starts with a 6x12 availability requirement ... but 7x24 usually becomes the goal.

If we look at the availability requirements for a data warehouse, normally you start out with 6 days a week, 12 hours a day. The decision makers are not going to come during the middle of a weekend to ask marketing questions. They don't normally do that. They don't come in midnight to ask these questions either. However, a transaction processing system, like the ATM systems, should be available all the time. I can go to an ATM at midnight if I want to, and withdraw money. That's not usually the requirement in the beginning of the data warehouse project, but as the data warehouse matures, availability becomes much more important. Because the decision makers start accessing and using the data more often. They start doing maybe interactive kind of analysis in the day and data mining kind of things over night. We will talk more about data mining later. So you can start out with 6×12 availability but will usually evolve in to 7×24 over a period of time. So one of the things we are going to talk about, although there will not be a major focus is about the tradeoffs that you make in availability. How do I design my data warehouse for 100% availability? So that it is always available for quires. And it turns out that there are very subtle interactions between availability and data loading. How do I make sure the data is available for querying while uploading the data? These issues turn out to have some tricky subtleties in that kind of environment.

Does not follows the traditional development model

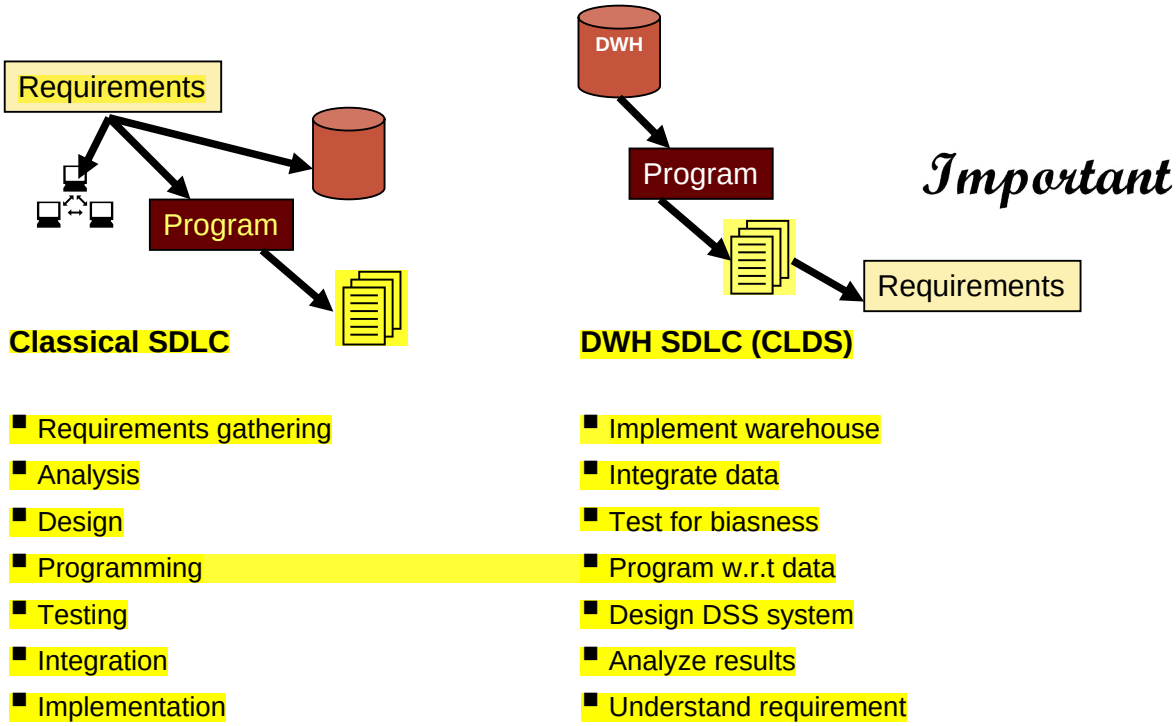


Figure-4.1: Comparison of SDLC & CLDS

Operational environment is created using the classical systems development life cycle. The DWH on the other hand operates under a very different life cycle, sometimes called CLDS i.e. the reverse of SDLC. The classical SDLC is requirement driven. In order to build the system, you must first understand the end user or business user requirements. Then you go into the stages of design and development typically taught in software engineering. The CLDS is almost exactly the reverse of SDLC. The CLDS starts with the data. Once the data is in hand, it is integrated, and then tested to see what bias there is, if any. Programs are then written against the data. The results are analyzed, and finally the requirements of the system are understood. Some Data Warehouse developers/vendors overuse this "natural" approach and add extensive charges for adding features or enhancing features already present once the end users get the "hang" of the system. The CLDS is a classical data driven development life cycle. Trying to apply inappropriate tools and techniques of development will only result in waste of effort/resources and confusion

4.1 Typical Queries

OLTP (On Line Transaction Processing) specific query

```
Select tx_date, balance from tx_table
Where account_ID = 23876;
```

DWH specific query *Important*

```
Select balance, age, sal, gender from customer_table and tx_table
Where age between (30 and 40) and
Education = 'graduate' and
```

CustID.customer_table = Customer_ID.tx_table;

Lets take a brief look a the two queries, the results of comparing them are summarized in table 4.1 below:

Important

OLTP	DWH
Primary key used	Primary key NOT used
No concept of Primary Index	Primary index used
May use a single table	Uses multiple tables
Few rows returned	Many rows returned
High selectivity of query	Low selectivity of query
Indexing on primary key (unique)	Indexing on primary index (non-unique)

Table-4.1: Comparison of OLTP and DWH for given queries

	Data Warehouse	OLTP
Scope	* Application –Neutral * Single source of "truth" * Evolves over time * How to improve business	* Application specific * Multiple databases with repetition * Off the shelf application * Runs the business
Data Perspective	* Historical, detailed data * Some summary * Lightly denormalized	* Operational data * No summary * Fully normalized
Queries	* Hardly uses PK * Number of results returned in thousands	* Based on PK * Number of results returned in hundreds
Time factor	* Minutes to hours * Typical availability 6x12	* Sub seconds to seconds * Typical availability 24x7

Table-4.2: Detailed comparison of OLTP and DWH

Comparisons of response times

- On-line analytical processing (OLAP) queries must be executed in a small number of seconds.
- Often requires denormalization and/or sampling.
- Complex query scripts and large list selections can generally be executed in a small number of minutes.
- Sophisticated clustering algorithms (e.g., data mining) can generally be executed in a small number of hours (even for hundreds of thousands of customers).

So there is one class of decision support environment called OLAP (Online Analytical Processing). The idea is that I am doing iterative decision making, using point and clicking, drilling down into data, and refining my decision model and I should be able to "execute" queries in an OLAP environment in a small number of seconds. How is it possible to execute queries in a small numbers of seconds? And potentially working with

billions of rows of data? I have to be a bit clever here. I have to consider special techniques like sampling, like de-normalization, special indexing techniques and other smart kinds of techniques. We will talk about those techniques when we go to the relevant parts. And how do I design a database which has these characteristics?

If I have got complex combination of tables and a large list selection and if it takes small number of minutes that's acceptable. If I got a lot of iterative analysis to do by pulling a list of names and customers and some information about them to send a direct mail to those customers, then it is acceptable to assume that that such queries don't have to be executed in few seconds. Of course as long as I am going to find them, it's all right.

4.2 Putting the pieces together

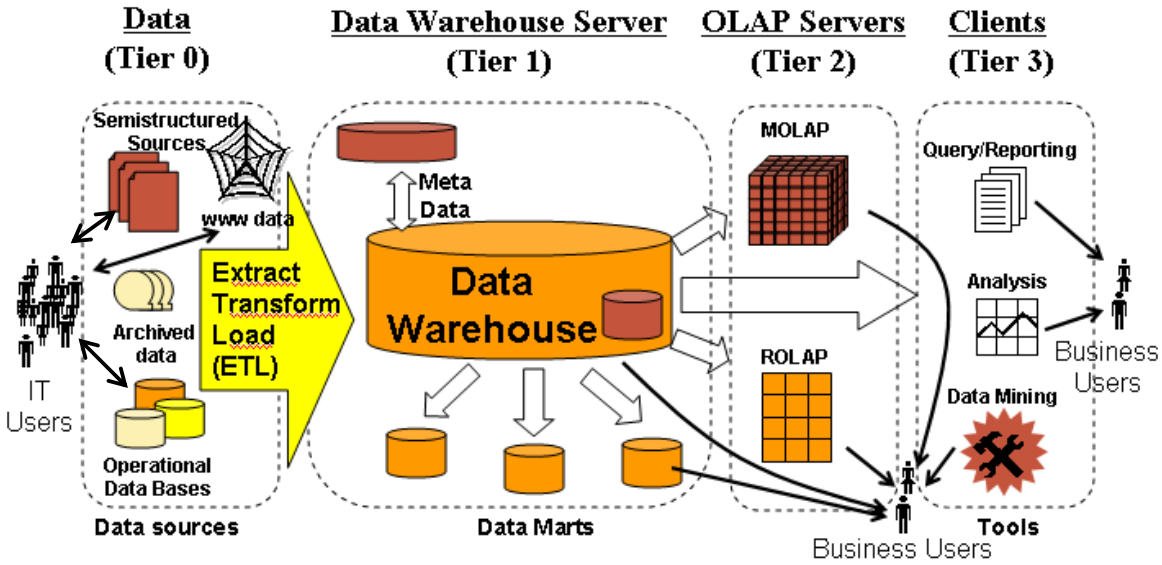


Figure-4.2: Putting the pieces together

Figure 4.2 gives a complete picture of how different "pieces" of a data warehouse fit together. We start with extracting data from different sources, transform and clean that data and then load it into the data warehouse. The data warehouse itself has to have specialized schemas to give quick answers to typical macroscopic queries. Data marts are created out of the data warehouse to service the needs of different departments such as marketing, sales etc such that they don't have to work with the heavy load of the complete data warehouse. Once the data warehouse is in place (as briefly discussed before) data cubes are created for OLAP based drill-down "all" possible queries. And we move further and use smart tools such as data mining clustering and classification techniques.

4.3 Why this is hard?

- Data sources are unstructured & heterogeneous.
- Requirements are always changing.
- Most computer scientist trained on OTLP systems, those concepts not valid for VLDB & DSS.
- The scale factor in VLDB implementations is difficult to comprehend.

- Performance impacts are often non-linear O(n) Vs. O(nlog_n) e.g. scanning vs. indexing.
- Complex computer/database architectures.
- Rapidly changing product characteristics.
- And so on...

In early days of data warehousing you would never even conceive giving business end user access to the data. They would always have to go through a data center, some kind of reporting or MIS system, that's really not true any more. The hard part of the problem now is architecting, designing and building these systems. And it is particularly hard at the high end. Because in a data warehouse environment, there are no stable requirements-change is constant. This is one of the main reasons why SDLC system dose not work. By the time you collect all the requirements following the SDLC approach and start designing the system the requirements may have completely changed. They are completely meaningless. Because any question that was interesting 6 months ago is definitely not interesting today.

The other issue is with very large databases scale dose strange things in the environment. And this is very important for computer scientist to understand because at small scale (i.e. hundreds of rows) an O(n log n) algorithm and an $O(n^2)$ algorithms are not much different in terms of performance. It dose not matter if you are using bubble sort or a heap sort for small amount of data; nobody cares as the difference in performance is not noticeable. However, when you get millions of records or a billion records, then it starts to hurt. So when you get very large data bases, and manipulating large amount of data because we are retaining history, this becomes a big problem and a proper design is required to be in place. Else the scale will kill you. The difference between O(n log n) and an $O(n^2)$ is huge when you get to billions of records.

4.4 High level implementation steps

Phase-I

1. Determine Users' Needs
2. Determine DBMS Server Platform
3. Determine Hardware Platform
4. Information & Data Modeling
5. Construct Metadata Repository

Phase-II

6. Data Acquisition & Cleansing
7. Data Transform, Transport & Populate
8. Determine Middleware Connectivity
9. Prototyping, Querying & Reporting
10. Data Mining
11. On Line Analytical Processing (OLAP)

Phase-III

12. Deployment & System Management

If you look at the high level implementations steps for a data warehouse it is important that you are driving the design of the data warehouse in the context of the data warehouse by the business requirements, and not driven by the technology.

So you start with the business requirements and say ok what problems I am trying to solve here. Am I going to do fraud detection or do customer retention analysis? What are the specifications of the problems that we have discussed? And identify the key source of these problems so you can understand what is going to be the cost and time required to fix them. But make sure this is done in the context of a logical data model that expresses the underlying business relationship of the data.

The data warehouse should be able to support multiple applications, multiple workloads against a single source of truth for the organization. So by identifying the business opportunity, you identify the information required, and then in the next stage, how to schedule those deliverables that are most important to the business. Ask yourself, what can I do in a 90 days time period to deliver values to the business? And then based on what I have decided here do the software and hardware selection. Remember that software is hardware, and hardware is easy ware. Because the software is much more complex to scale (algorithm complexity) then the hardware. Hardware is getting cheaper over time. So you drive typically from the software not the hardware.

Masters

Lecture 5: Introduction to Data Ware Housing–Part V

Learning Goals

Types of Data Warehouses

- Financial
- Telecommunication
- Insurance
- Human Resource

5.1 Types and Typical Applications of DWH

Financial DWH

- First data warehouse that an organization builds. This is appealing because:
  - Nerve center, easy to get attention.
  - In most organizations, smallest data set.
  - Touches all aspects of an organization, with a common denomination i.e. money.
  - Inherent structure of data directly influenced by the day-to-day activities of financial processing.
  - Financial data warehouses suffer from an anomaly of inability to match balanced accounts, due to many legitimate reasons.

Financial data warehouses are often the first data warehouse that an organization builds. This is appealing because:

Financial data is ALWAYS at the nerve center of the organization. Therefore, getting attention drawn to a well-built financial data warehouse is a very easy thing to do. In most organizations (but not all), financial data represents one of the smallest volumes of data that exist. Finance cuts across all of the aspects of the corporate data and has a common denominator i.e. money. Financial data inherently has a structure of data directly influenced by the day-to-day activities of financial processing and the list goes on. For these reasons, finance becomes a very good target for beginning to build the corporate data warehouse.

Financial data warehouses have some severe drawbacks that are not found elsewhere. When purists use a financial data warehouse, they are overjoyed to find that it is almost impossible to reconcile down to the rupee. Again and again they will report that they saw a report yesterday that stated the balance to be Rs. 12,345,678 but when did the same report today the using our accounting MIS system the balance was Rs. 13,245,678. Hence one can not trust the new system☺.

Note that it is unlikely that the financial data warehouse would balance to the last rupee, such a notion is completely wrong for a number of reasons. The accounting periods may

be different in different operational systems, but in a data warehouse, it is the corporate calendar. The classifications of regions may change. In the operational system Northern Pakistan may consist of Murree, Abbotabad etc. but in the national data warehouse, Northern Pakistan consists of every major city north of Lahore. In the operational system (for example) sale or oranges is recorded in dozen in one part of Pakistan, while in other they are sold by weight and in another part by crates. However, in the national data warehouse, they are accounted for based on dozen and the list goes on.

Telecommunication

Dominated by sheer volume of data

Many ways to accommodate call level detail:

- Only a few months of call level detail,
- Storing lots of call level detail scattered over different storage media,
- Storing only selective call level detail, etc.
- Unfortunately, for many kinds of processing, working at an aggregate level is simply not possible.

5.2 Telecommunications Data Warehouse

The telecommunications data warehouse is dominated by the sheer volume of data generated at the call level subject area. There are many other types of data in this environment as well. No other data warehouse environment is dominated by the size of one of the subject areas as call level detail dominates a telecommunications data warehouse.

There are many ways that call level detail can be accommodated:

- Only a few months of call level detail,
- Storing lots of call level detail scattered over different storage media,
- Summarizing or aggregating call level detail,
- Storing only selective call level detail, and so forth.

Unfortunately, for many kinds of processing, the only way it can be accommodated is by working at the call level detail. Working at a summary level or an aggregate level simply is not a possibility.

Insurance

Insurance data warehouses are similar to other data warehouses BUT with a few exceptions.

- Stored data that is very, very old, used for actuarial processing.

- Typical business may change dramatically over last 40-50 years, but not insurance. Therefore, if someone says that the insurance business has not changed much over the last 40-50 years would be telling the truth.

- In retailing or telecomm there are a few important dates, but in the insurance environment there are many dates of many kinds. In the retail sales environment there are a few dates such as: sale date, stocking date, and perhaps manufacturing date. In the banking environment there is the transaction date. However, in the insurance environment there are dates everywhere of every kind.

- Long operational business cycles, in years. Thus the operating speed is different. Consider a bank, you insert the ATM card and draw the money instantly, or deposit a check and it is cashed at most in a week. You go to store and purchase an item, and instantly make a telephone call. However, in an insurance environment a claim is made and it may take several years before it is settled.

- Transactions are not gathered and processed, but are in kind of "frozen" state.

- Thus a very unique approach of design & implementation.

Differences between different types of data warehouses *Important*

- Financial data warehouses, often the first a corporation builds. However, it will not reconcile down to the last rupee with the existing operational financial environment.

- Insurance data warehouses are similar to other data warehouses with a few exceptions: such as the length of time that insurance data warehouses exists, in terms of the dates found in the business, and in terms of the operational business cycle.

- The human resources data warehouse is different from others because this without doubt has only one major subject area (yes you guessed it right!).

- The telecommunications data warehouse is dominated by the sheer volume of data generated at one subject area.

Indeed there are many types of data warehouses, each with their own peculiarities and all of them can not be discussed here.

5.3 Typical Applications of DWH *Important*

There are, and there can be many many applications of a data warehouse. It is not possible to discuss all of them. Some representative applications are listed to be discussed as follows:

- Fraud detection.

- Profitability analysis.
- Direct mail/database marketing.
- Credit risk prediction.
- Customer retention modeling.
- Yield management.
- Inventory management.

ROI on any one of these applications can justify hardware/software and consultancy costs in most organizations.

Fraud detection *Important*

- By observing data usage patterns.
- People have typical purchase patterns.
- Deviation from patterns.
- Certain cities notorious for fraud.
- Certain items bought by stolen cards.
- Similar behavior for stolen phone cards.

You can look at patterns of data usage, and behavior of customer and detect fraud very easily. Lets suppose you are a Credit Card Company. You know that Danyal as a customer has certain behaviors. He tends to buy airplane tickets, make hotel reservations, rents cars using his credit card. When he goes shopping, it is usually only on a weekend and only on certain stores that he goes shopping and buys sports goods and action clothing. And then all of sudden you see very different behaviors coming in, such as credit card transactions in Karachi. Karachi by the way has very high stolen credit card rate. And the patterns of buying behavior are stereos, and CDs and TVs i.e. electronic goods that are typically the things that people buy on stolen credit cards. You can notice and say "look, this is not airplane or hotel reservations, this does not look like Danyal". It is not in the city where Danyal lives or goes to very often. Although he was there a week ago, so maybe this is how you can tell that when and where his credit card was stolen. And the purchases were made after he left the city. Hence "high probability of fraud".

So I can predict based on the behavior of the individuals, when things out of the typical routine happen. Then I can stop the fraud happening even before it has spread. That's the example in credit card but you can do the same thing with telephone cards. I am sure you have used these telephone cards. You punch in your number and you get access to telephone network. You find people at airports or railway stations who make there living by stealing the telephone card codes. They stay near the phones at the air port, they act like they are talking on the phone, but they are really looking over your shoulder when you are punching in the code. And then they memorize the code and then go sell it on the street to people who make long distance phone calls to all their friends in Brazil and Australia all these expensive places. You can look at the calling patterns and say look "Danyal makes regular calls to areas like say Rawalpindi and to friends in Lahore" and all of sudden we see all these phone calls to Toba Taik Singh or Bhawalpur, meaning there is some thing wrong here.

Profitability Analysis *Important*

- Banks know if they are profitable or not.

- Don't know which customers are profitable.
- Typically more than 50% are NOT profitable.
- Don't know which one?
- Balance is not enough, transactional behavior is the key.
- Restructure products and pricing strategies.
- Life-time profitability models (next 3-5 years).

It is another example. If you go to an average bank, to Pakistan or any where in the world, the bank will know if they are profitable or not. But they don't know which customers are profitable or not, and at most banks this is true, especially true for retail banks. These banks do business with the consumers, such that more then 50 % of the customers are unprofitable. In other word the bank is loosing money on 50% of their customers. But they don't know which 50%? That's the problem. So the idea behind profitability analysis is to do the analysis to figure out which customers are profitable and which customers are not profitable. And then based on that they can restructure their product offering and there pricing strategies to do more profitable business. I used a banking example but same is true for Telecommunications Company or any other consumer oriented business that requires access to detail data. It is not sufficient to just know the account balance, but also transactional behavior and so on to do profitability analysis. And once I know profitability retrospectively, I can also build predictive models to understand what it's going to be prospectively. So I can build what's called lifetime value models to using the historical data to predict what the future profitability will be for the next 3 or 5 years.

Direct mail marketing *Important*

- Targeted marketing.
- Offering high bandwidth package NOT to all users.
- Know from call detail records of web surfing.
- Saves marketing expense, saving pennies.
- Knowing your customers better.

Assume you are a telecommunications company, and you have an offering that you can make such as to give value added Internet access through mobile phones. You can send a letter or an SMS to every customer in the country and tell them about this value added Internet access service. But 95% of that mail will be completely wasted. Because a small percentage of customers will be interested in such as service. So the question here is how you your message to the right people in a meaningful way. If you have the call detailed records, you can tell very very easily which of your customer's access the Internet and which of them just make missed calls. Because the patters of the call detail records of Internet access are completely different from making voice calls and making missed calls. They have different durations; they have different data bandwidth requirement and so on. So by having access to detail data, you can target those people that are better to get a second account, or better to get a value added connection/service.

Credit risk prediction *Important*

- Who should get a loan?
- Customer segregation i.e. stable vs. rolling.
- Qualitative decision making NOT subjective.
- Different interest rates for different customers.

- Do not subsidize bad customer on the basis of good.

Credit risk prediction is another area where data warehouse is used quite frequently. Let's say I and my friend both apply for home loan to construct a house. How would the bank determine which one of those should get a loan? So typically they will say, ok I have been employed for 5 years, have good salary, married, have children etc etc and then based on the context of application then they decide yes we think you are a low risk we will give you the loan. Or say no you are a high risk because you had a job only for one year, your salary is not high, and you have lots of change of jobs recently, so you know we don't feel so good about the risk we are taking, so we won't give you the loan.

So typically today it is based on what is called qualitative decision-making. Basically in the offices these are just references say OK Saeed is a nice person or is related to me or is my neighbor so we give him the loan. And I don't really know Waleed over here so we don't give him a loan. This is not the best way to give fair loan to people and is not the best way to manage risk in the business. So with credit risk, the idea is to build quantitative models, which predict risk based on historical data. And not only I use the historical data to predict the risk, if I get more sophisticated, what I can do is called risk based pricing. So today lets say you offer me (Saeed) a loan, and also to the other applicant i.e. Waleed also loan, and both are given the loan at the same interest rates. However, if Waleed is a lower risk then Saeed then why should both pay the same interest rates? If Waleed is a low risk, then he should get a lower interest rate other wise the bank is subsidizing the higher risk loan. So what's happening is that you are using very precise credit prediction models in order to adjust the interest to give lower interest to people who are at lower risk of default. And again it should use some data mining algorithms to predict the risk, which is called the risk index on every individual consumer. So that every consumer gets a loan at an interest that is appropriate to their risk level.

Yield Management *Important*

- Works for fixed inventory businesses.
- The price of item suddenly goes to zero.
- Item prices vary for varying customers.
- Example: Air Lines, Hotels, Car rentals etc.
- Price of (say) Air Ticket depends on:
    - How much in advance ticket was bought?
    - How many vacant seats were present?
    - How profitable is the customer?
    - Ticket is one-way or return?

Yield management works, for example for sophisticated airlines which use a variable pricing mechanism for the seats. Say you are flying on a plane; 90% of the time the person sitting next to you pays a different price for the ticket as compared to yours. The seats are basically same, yet you pay a different price for the ticket, why is it that? Because the airline used a sophisticated method to see how much in advance the person bought the ticket, and how many seats were available on the plane at the time the person bought the ticket. Or did they buy a round ticket or one-way ticket; there are all kinds of sophisticated ways to do the pricing. Yield management helps decide what should be the price at any time for that airline seat.

Yield management is important for almost any type of fixed inventory business. For example, once the plane leaves, the price of the seat if it is empty or full means nothing any more. So the goal of the airline is to fill every seat with the maximum possible price for that seat. The same is true for hotels. For example a three star hotel had an empty room last night. If they can get Rs. 1500 for that night it is better then having nothing at all. So someone who booked the room for example as a group, would have paid lower price per room as compared to someone who is booking only one room. The price is different, because the hotels use yield management to fill out their hotel rooms as profitably as they can. This is applicable to car rentals also.

Agriculture Systems *Important*

- Agri and related data collected for decades.
- Metrological data consists of 50+ attributes.
- Decision making based on expert judgment.
- Lack of integration results in underutilization.
- What is required, in which amount and when?

Each year different government departments and agencies in Pakistan create tens of thousands of digital and non digital files from thousands of pest-scouting surveys, yield surveys, metrological data collection, river flows etc. This data collection has been going on for decades. The data collected has never been compiled, standardized and integrated to give a complete picture. Thus the lack of data integration and standardization contributes to an under-utilization of historical data, and inevitably results in an inability to perform any scientific predictive analysis for effective decision support and policy making. An Agriculture data warehouse is the answer, as processing 100+ variables by experts for large historical data is not possible. This has repeatedly resulted in tragic outcomes.

Although the benefits are many, but are not without some major issues or problems, some of them are as follows:

5.4 Major Issues

- Unavailability of data and in an illegible format and form.
- Most of the detail data is not digitized; therefore it can not be readily used by computers.
- Dispersion of data across multiple and geographically displaced organizations.
- Even though the said organizations are required to share the data with common citizens, no official data sharing mechanism is currently in place.

Until the requisite training is provided, government employees and farmers are not equipped with enough technical and literate skills to use latest information technology tools to use the data warehouse

Masters
Subscribes to Masters

Masters

Lecture 6: Normalization

Learning Goals

- What is normalization
- Anomalies
- Normalization: 1NF
- Normalization: 2NF
- Normalization: 3NF

What is normalization?

What are the goals of normalization?

- Eliminate redundant data.
- Ensure data dependencies make sense.

What is the result of normalization?

What are the levels of normalization?

Always follow purist's approach of normalization?
NO

6.1 Normalization

Before we begin our discussion about the normal forms, it's important to understand that what we will discuss are guidelines and guidelines only i.e. not a dejure standard. Sometimes (especially in a DSS environment), it becomes necessary to drift from this purist approach to meet practical business requirements of performance. However, when deviation take place, it's extremely important to evaluate any possible consequences these deviations would cause and the corresponding inconsistencies and anomalies this is what de-normalization is about.

Important

Normalization is the process of efficiently organizing data in a database by decomposing (splitting) a relational table into smaller tables by projection. There are basically two goals of normalization as follows:

1. Eliminate redundant data (for example, storing the same data in more than one table)

2. Ensuring data dependencies make sense (only storing related data in a table).

Both of these are worthy goals, as they reduce the amount of space a database consumes, and ensure that data is logically stored and is in third normal form (3NF). The database community has developed a series of guidelines or benchmarks for ensuring that databases are indeed normalized. These goals are referred to as normal forms and are numbered from one (the lowest form of normalization, referred to as first normal form or 1NF) through five (fifth normal form or 5NF). The first two normal forms are

intermediate steps to achieve the goal of having all tables in 3NF. Note that in order to be correct, decomposition must be lossless i.e. the new tables can be recombined by a natural join to recreate the original table without creating any false or redundant data and without any loss of any data/information.

Consider a student database system to be developed for a multi-campus university, such that it specializes in one degree program at a campus i.e. BS, MS or PhD.

SID	Degree	Campus	Course	Marks
1	BS	Islamabad	CS-101	30
1	BS	Islamabad	CS-102	20
1	BS	Islamabad	CS-103	40
1	BS	Islamabad	CS-104	20
1	BS	Islamabad	CS-105	10
1	BS	Islamabad	CS-106	10
2	MS	Lahore	CS-101	30
2	MS	Lahore	CS-102	40
3	MS	Lahore	CS-102	20
4	BS	Islamabad	CS-102	20
4	BS	Islamabad	CS-104	30
4	BS	Islamabad	CS-105	40

SID: Student ID

Degree: Registered as BS or MS student

Campus: City where campus is located

Course: Course taken

Marks: Score out of max of 50

Figure-6.1: Part of example student database system

We consider the case of development of a student DBMS for a multi-campus university i.e. with campuses in many cities. There is a head office of the University (say) in Islamabad which would like to get an overall picture of all the students at all the campuses of the university. A typical detail table is shown with a brief description of the fields. These are in no way the only fields, but sufficient to briefly review the basic forms of normalization.

In order to uniquely associate marks with course and the student, a composite primary key composed of SID and campus is used.

Database system in 1NF

SID	Degree	Campus	Course	Marks
1	BS	Islamabad	CS-101	30
1	BS	Islamabad	CS-102	20
1	BS	Islamabad	CS-103	40
1	BS	Islamabad	CS-104	20
1	BS	Islamabad	CS-105	10
1	BS	Islamabad	CS-106	10
2	MS	Lahore	CS-101	30
2	MS	Lahore	CS-102	40
3	MS	Lahore	CS-102	20
4	BS	Islamabad	CS-102	20
4	BS	Islamabad	CS-104	30
4	BS	Islamabad	CS-105	40

Table FIRST

Figure-6.2: Part of example student database system in 1NF

Although the table is shown in 1NF i.e. it contains atomic values, there are no repeating values, there is no aggregation, yet it contains redundant data. For example, information about the student's degree, and campus location is repeated for every course taken. Redundancy causes what are called *update anomalies*. Update anomalies are problems that arise when records are inserted, deleted, or updated in the database. For example, the following anomalies can occur:

6.2 Normalization: 1NF
Update Anomalies

INSERT. Certain student with SID 5 got admission in a different campus (say) Karachi cannot be added until the student registers for a course.

DELETE. If student graduates and his/her corresponding record is deleted, then all information about that student is lost.

UPDATE. If student migrates from Islamabad campus to Lahore campus (say) SID = 1, then six rows would have to be updated with this new information.

INSERT. The fact that a certain student with SID 5 got admission in a different campus (say) Karachi cannot be added until the student registers for a course.

DELETE. If student graduates and his/her corresponding record is deleted, then all information about that student is lost.
UPDATE. If student migrates from Islamabad campus to Lahore campus (say) SID = 1, then six rows would have to be updated with this new information.

6.3 Normalization: 2NF

Every non-key column is fully dependent on the PK.

FIRST is in 1NF but not in 2NF because degree and campus are functionally dependent upon only on the column SID of the composite key (SID, course). This can be illustrated by listing the functional dependencies in the table:

SID —> campus, degree

campus —> degree

(SID, Course) —> Marks

} SID and Campus are NOT unique

To transform the table FIRST into 2NF we move the columns SID, Degree and Campus to a new table called REGISTRATION. The column SID becomes the primary key of this new table.

The definition of second normal form states that only tables with composite primary keys can be in 1NF but not in 2NF.

A relational table is in second normal form 2NF if it is in 1NF and every non-key column is fully dependent upon the primary key. That is, every non-key column must be dependent upon the entire primary key. The table in the last slide is in 1NF but not in 2NF because degree, course and even marks are functionally dependent upon the SID and the campus.

FIRST is in 1NF but not in 2NF because degree and campus are functionally dependent upon only on the column SID of the composite key (SID, course). This can be illustrated by listing the functional dependencies in the table:

SID —> campus, status
campus —> degree
(SID, Course) —> Marks

PERFORMANCE

SID	Course	Marks
1	CS-101	30
1	CS-102	20
1	CS-103	40
1	CS-104	20
1	CS-105	10
1	CS-106	10
2	CS-101	30
2	CS-102	40
3	CS-102	20
4	CS-102	20
4	CS-104	30
4	CS-105	40

SID is now a PK

REGISTRATION

SID	Degree	Campus
1	BS	Islamabad
2	MS	Lahore
3	MS	Lahore
4	BS	Islamabad
5	PhD	Peshawar

PERFORMANCE in 2NF as (SID, Course) uniquely identify Marks

Figure-6.3: Part of example student database system in 2NF

To transform the table FIRST into 2NF we move the columns SID, Degree and city to a new table called REGISTRATION. The column SID becomes the primary key of this new table.

Normalization: 2NF

Presence of modification anomalies for tables in 2NF. For the table REGISTRATION, they are:

- **INSERT:** Until a student gets registered in a degree program, that program cannot be offered!
- **DELETE:** Deleting any row from REGISTRATION destroys all other facts in the table.

Why there are anomalies?

The table is in 2NF but NOT in 3NF

Tables in 2NF but not in 3NF still contain modification anomalies. In the example of REGISTRATION, they are:

- INSERT. The fact that a particular campus has a certain degree (Peshawar campus decides to run a PhD program) cannot be inserted until there is a student registered for PhD in the campus.

- DELETE. Deleting any row from REGISTRATION table destroys the degree information about the campus as well as the association between student and campus.

Normalization: 3NF

All columns must be dependent only on the primary key.

Table PERFORMANCE is already in 3NF. The non-key column, marks, is fully dependent upon the primary key (SID, degree).

REGISTRATION is in 2NF but not in 3NF because it contains a transitive dependency.

A transitive dependency occurs when a non-key column that is a determinant of the primary key is the determinate of other columns.

The concept of a transitive dependency can be illustrated by showing the functional dependencies:

REGISTRATION.SID —> REGISTRATION.Degree
REGISTRATION.SID —> REGISTRATION.Campus
REGISTRATION.Campus —> REGISTRATION.Degree

Note that REGISTRATION. Degree is determined both by the primary key SID and the non-key column campus.

For a relational table to be in third normal form (3NF) all columns must be dependent only upon the primary key. More formally, a relational table is in 3NF if it is already in 2NF and every non-key column is non transitively dependent upon its primary key. In other words, all non-key attributes are functionally dependent only upon the primary key. Or put another way, all attributes must be directly dependent on the primary key without implied dependencies through other attributes of the relation.

Table PERFORMANCE is already in 3NF. The non-key column marks is fully dependent and identified using the primary key SID and COURSE. However, table REGISTRATION is still only in 2NF as there is a *transitive dependency*. A transitive dependency arises when a non-key column that is a determinant of the primary key is the determinate of other columns.

To transform REGISTRATION into 3NF, a new table is created called CAMPUS_DEGREE and the columns campus and degree are moved into it. Degree is deleted from the original table, campus is left behind to serve as a foreign key to table CAMPUS_DEGREE, and the original table is renamed to STUDENT_CAMPUS to reflect its semantic meaning.

Normalization: 3NF

To transform REGISTRATION into 3NF, we create a new table called CAMPUS_DEGREE and move the columns campus and degree into it.

Degree is deleted from the original table, campus is left behind to serve as a foreign key to CAMPUS_DEGREE, and the original table is renamed to STUDENT_CAMPUS to reflect its semantic meaning.

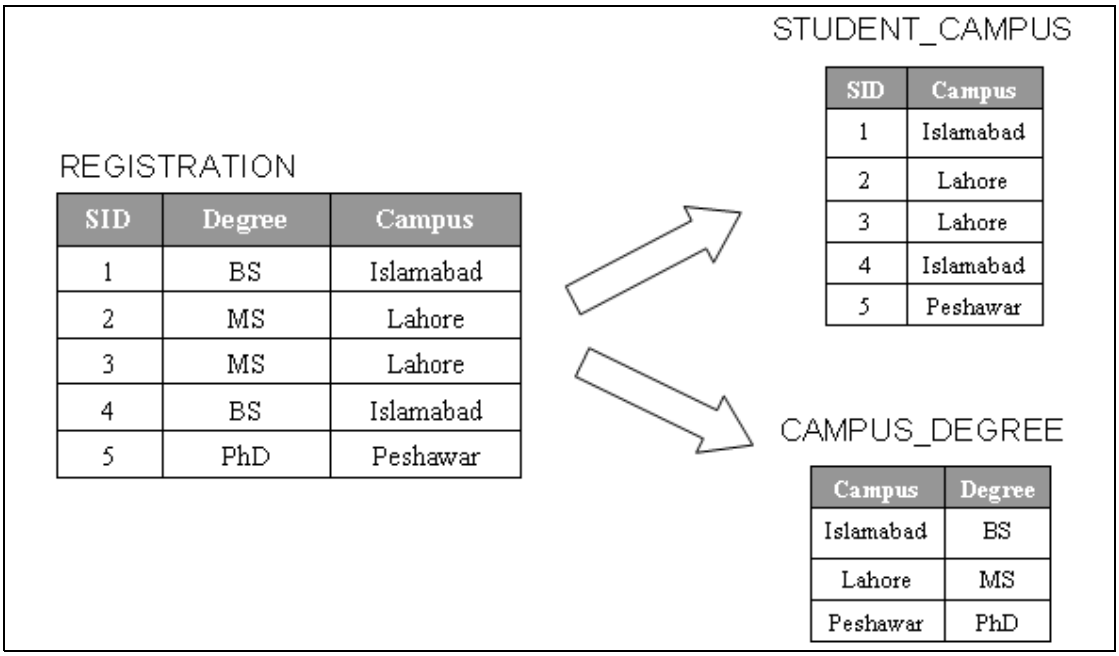


Figure-6.4: Part of example student database system in 3NF

Figure-6.4 shows that the table REGISTRATION is transformed into two tables i.e. Student_Campus and Campus_Degree. If we look at it in the context of memory, we observe that the storage space requirement has increased, for this particular example by about 7%.

Normalization: 3NF

Removal of anomalies and improvement in queries as follows:

- **INSERT:** Able to first offer a degree program, and then students registering in it.
- **UPDATE:** Migrating students between campuses by changing a single row.
- **DELETE:** Deleting information about a course, without deleting facts about all columns in the record.

Observe that by virtue of bringing a relational table into 3NF, storage of redundant data is further eliminated, that not only results in saving of space, but also reduces manipulation anomalies. For example, as a consequence following improvements have taken place:

INSERT: If Peshawar campus decides to offer a PhD program, this can be reflected even though there is no student currently registered in that campus. Similarly, facts about new students can be added even though they may not have registered for a course.

UPDATE: Changing the campus of a student (by migration) or a degree program of a campus requires modification of only a single row.

DELETE: Information about courses taken can be deleted without destroying information about a student or a campus.

6.5 Conclusions

- Normalization guidelines are cumulative.
- Generally a good idea to only ensure 2NF.
- 3NF is at the cost of simplicity and performance.
- There is a 4NF with no multi-valued dependencies.
- There is also a 5NF.

Normalization guidelines are cumulative. For a database to be in 2NF, it must first fulfill all the criteria of a 1NF database.

It is generally a good idea to ensure that the relational table is at least in 2NF. The goal is to avoid data redundancy so as to prevent slow corruption of data due to DML anomolies and make the best possible use of storage.

Although 3NF removes even more data redundancy, but this is at the cost of simplicity and performance. Hence it is upto the designer to find a balance between normalization and speed/simplicity.

There is a fourth normal form (4NF) with one additional requirement i.e. meet all the requirements of 3NF and there must be no multi-valued dependencies.

There is also a 5NF, but that is more of academic interest.

© Copyright Virtual University of Pakistan

Masters

Lecture 07: De-Normalization

Leaning Goals

- What is De-Normalization?
- Why De-Normalization used in DSS?
- How De-Normalization improves performance?
- Guidelines for De-normalization

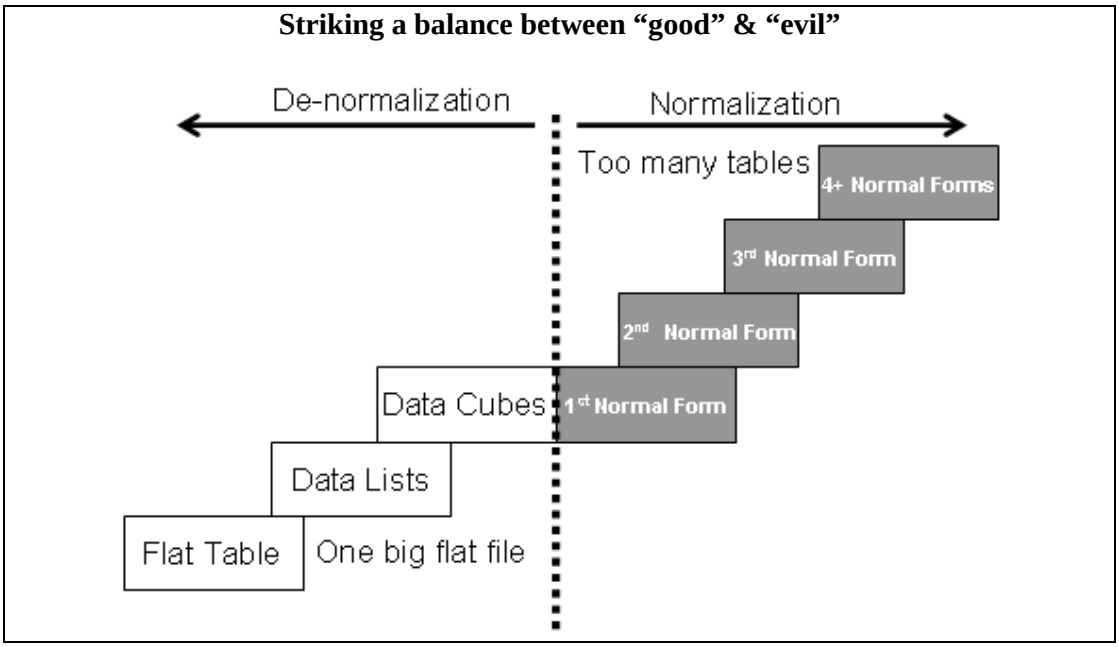


Figure-7.1: Striking a balance between normalization and de-normalization

There should be a balance between normalized and de-normalized forms. In a fully normalized form, too many joins are required and in a totally de-normalized form, we have a big, wide single table. Database should be aligned someplace in between so as to strike a balance, especially in the context of the queries and the application domain.

A "pure" normalized design is a good starting point for a data model and is a great thing to do and achieve in academia. However, as briefly mentioned in the previous lecture, in the reality of the "real world", the enhancement in performance delivered by some selective de-normalization technique can be a very valuable tool. The key to success is to undertake de-normalization as a design technique very cautiously and consciously. Do not let proliferation of the technique take over your data warehouse or you will end up with a single big flat file!

7.1 What is De-Normalization?

- It is not chaos, more like a "controlled crash" with the aim of performance enhancement without loss of information.

- Normalization is a rule of thumb in DBMS, but in DSS ease of use is achieved by way of denormalization.

- De-normalization comes in many flavors, such as combining tables, splitting tables, adding data etc., but all done very carefully.

'Denormalization' does not mean that anything and everything goes. Denormalization does not mean chaos or disorder or indiscipline. The development of properly de-normalized data structures follows software engineering principles, which insure that information will not be lost. De-normalization is the process of selectively transforming normalized relations into un-normalized physical record specifications, with the aim of reducing query processing time. Another fundamental purpose of denormalization is to reduce the number of physical tables that must be accessed to retrieve the desired data by reducing the number of joins required to answer a query. Some people tend to confuse dimensional modeling with de-normalization. This will become clear when we will cover dimensional modeling, where indeed tables are collapsed together.

7.2 Why De-Normalization in DSS?

- Bringing "close" dispersed but related data items.
- Query performance in DSS significantly dependent on physical data model.
- Very early studies showed performance difference in orders of magnitude for different number de-normalized tables and rows per table.
- The level of de-normalization should be carefully considered.

The efficient processing of data depends how close together the related data items are. Often all the attributes that appear within a relation are not used together, and data from different relations is needed together to answer a query or produce a report. Although normalized relations solve data maintenance anomalies (discussed in last lecture), however, normalized relations if implemented one for one as physical records, may not yield efficient data processing times. DSS query performance is a function of the performance of every component in the data delivery architecture, but is strongly associated with the physical data model. Intelligent data modeling through the use of techniques such as de-normalization, aggregation, and partitioning, can provide orders of magnitude performance gains compared to the use of normalized data structures.

The processing performance between totally normalized and partially normalized DSSs can be dramatic. Inmon (grand father of data warehousing) reported way back in 1988 through a study, by quantifying the performance of fully and partially normalized DSSs. In his study, a fully normalized DSS contained eight tables with about 50,000 rows each, another partially normalized DSS had four tables with roughly 25,000 rows each, and yet another partially normalized DSS had two tables. The results of the study showed that the less than fully normalized DSSs could muster a performance as much as an order of magnitude better than the fully normalized DSS. Although such results depend greatly on the DSS and the type of processing, yet these results suggest that one should carefully consider whether the physical records should exactly match the normalized relations for a DSS or not?

7.3 How De-Normalization improves performance?

De-normalization specifically improves performance by either: *Important*

- Reducing the number of tables and hence the reliance on joins, which consequently speeds up performance.
- Reducing the number of joins required during query execution, or
- Reducing the number of rows to be retrieved from the Primary Data Table

The higher the level of normalization, the greater will be the number of tables in the DSS as the depth of snowflake schema would increase. The greater the number of tables in the DSS, the more joins are necessary for data manipulation. Joins slow performance, especially for very large tables for large data extractions, which is a norm in DSS not an exception. De-normalization reduces the number of tables and hence the reliance on joins, which consequently speeds up performance.

De-normalization can help minimize joins and foreign keys and help resolve aggregates. By storing values that would otherwise need to be retrieved (repeatedly), one may be able to reduce the number of indexes and even tables required to process queries.

7.4 Four Guidelines for De-normalization *Important*

1. Carefully do a cost-benefit analysis (frequency of use, additional storage, join time).

2. Do a data requirement and storage analysis.

3. Weigh against the maintenance issue of the redundant data (triggers used).

4. When in doubt, don't de-normalize.

Guidelines for Denormalization:-
Following are some of the basic guidelines to help determine whether it's time to de-normalize the DSS design or not:

1. Balance the frequency of use of the data items in question, the cost of additional storage to duplicate the data, and the acquisition time of the join.

2. Understand how much data is involved in the typical query; the amount of data affects the amount of redundancy and additional storage requirements.

3. Remember that redundant data is a performance benefit at query time, but is a performance liability at update time because each copy of the data needs to be kept up to date. Typically triggers are written to maintain the integrity of the duplicated data.

4. De-normalization usually speeds up data retrieval, but it can slow the data modification processes. It may be noted that both on-line and batch system performance is adversely affected by a high degree of de-normalization. Hence the golden rule is: When in doubt, don't de-normalize.

7.5 Areas for Applying De-Normalization Techniques

- Dealing with the abundance of star schemas.
- Fast access of time series data for analysis.
    - Fast aggregate (sum, average etc.) results and complicated calculations.
    - Multidimensional analysis (e.g. geography) in a complex hierarchy.
    - Dealing with few updates but many join queries.

De-normalization will ultimately affect the database size and query performance.

De-normalization is especially useful while dealing with the abundance of star schemas that are found in many data warehouse installations. For such cases, de-normalization provides better performance and a more natural data structure for supporting decision making. The goal of most analytical processes in a typical data warehouse environment is to access aggregates such as averages, sums, complicated formula calculations, top 10 customers etc. Typical OLTP systems contain only the raw transaction data, while decision makers expect to find aggregated and time-series data in their data warehouse to get the big picture through immediate query and display. Important and common parts of a data warehouse that are a good candidate for de-normalization include (but are not limited to):

- Aggregation and complicated calculations.
- Multidimensional analysis in a complex hierarchy.
- Few updates, but many join queries.

Geography is a good example of a multidimensional hierarchy (e.g. Province, Division, District, city and zone). Basic design decisions for example, the selection of dimensions, the number of dimensions to be used and what facts to aggregate will ultimately affect the database size and query performance.

7.6 Five principal De-Normalization techniques *Important*

Now we will discuss de-normalization techniques that have been commonly adopted by experienced database designers. These techniques can be classified into four prevalent strategies for denormalization which are:

1. Collapsing Tables.
    - Two entities with a One-to-One relationship.
    - Two entities with a Many-to-Many relationship.

2. Pre-joining.

3. Splitting Tables (Horizontal/Vertical Splitting).

4. Adding Redundant Columns (Reference Data).

5. Derived Attributes (Summary, Total, Balance etc).

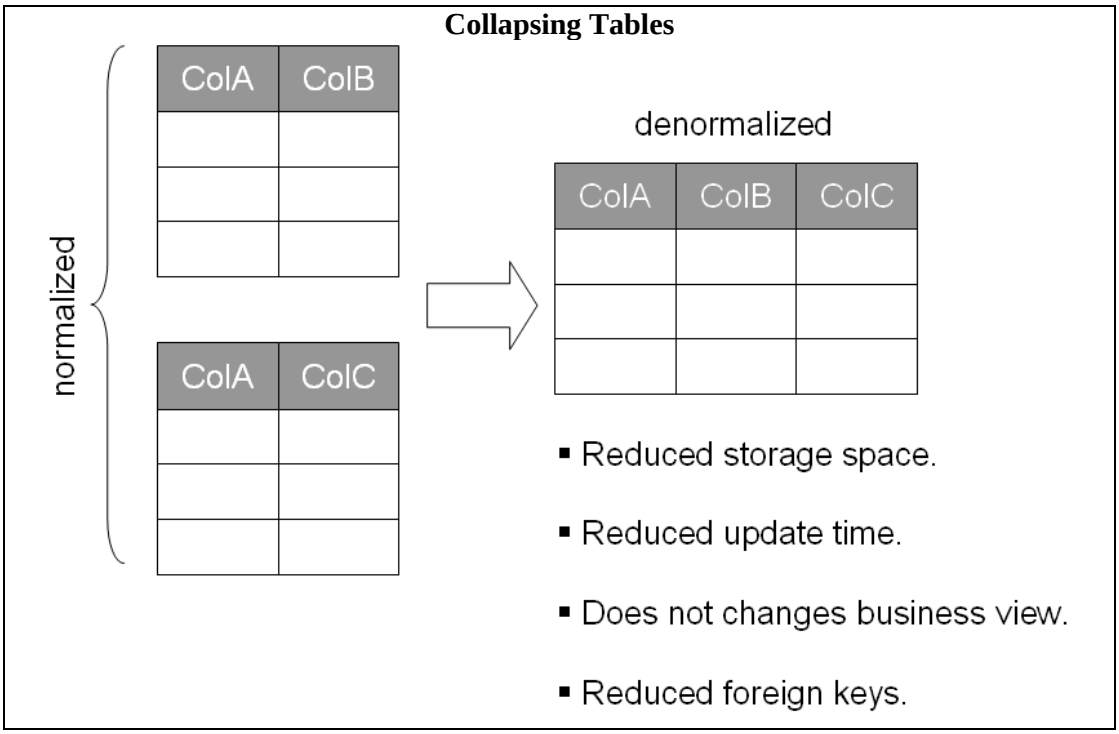


Figure-7.1: Collapsing Tables

1. Collapsing Tables

One of the most common and safe denormalization techniques is combining of One-to-One relationships. This situation occurs when for each row of entity A, there is only one related row in entity B. While the key attributes for the entities may or may not be the same, their equal participation in a relationship indicates that they can be treated as a single unit. For example, if users frequently need to see COLA, COLB, and COLC together and the data from the two tables are in a One-to-One relationship, the solution is to collapse the two tables into one. For example, SID and gender in one table, and SID and degree in the other table.

In general, collapsing tables in One-to-One relationship has fewer drawbacks than others. There are several advantages of this technique, some of the obvious ones being reduced storage space, reduced amount of time for data update, some of the other not so apparent advantages are reduced number of foreign keys on tables, reduced number of indexes (since most indexes are created based on primary/foreign keys). Furthermore, combining the columns does not change the business view, but does decrease access time by having fewer physical objects and reducing overhead.

Masters

Lecture 08: De-Normalization Techniques

Learning Goals

- Splitting Tables
- Pre-Joining
- Adding Redundant Columns

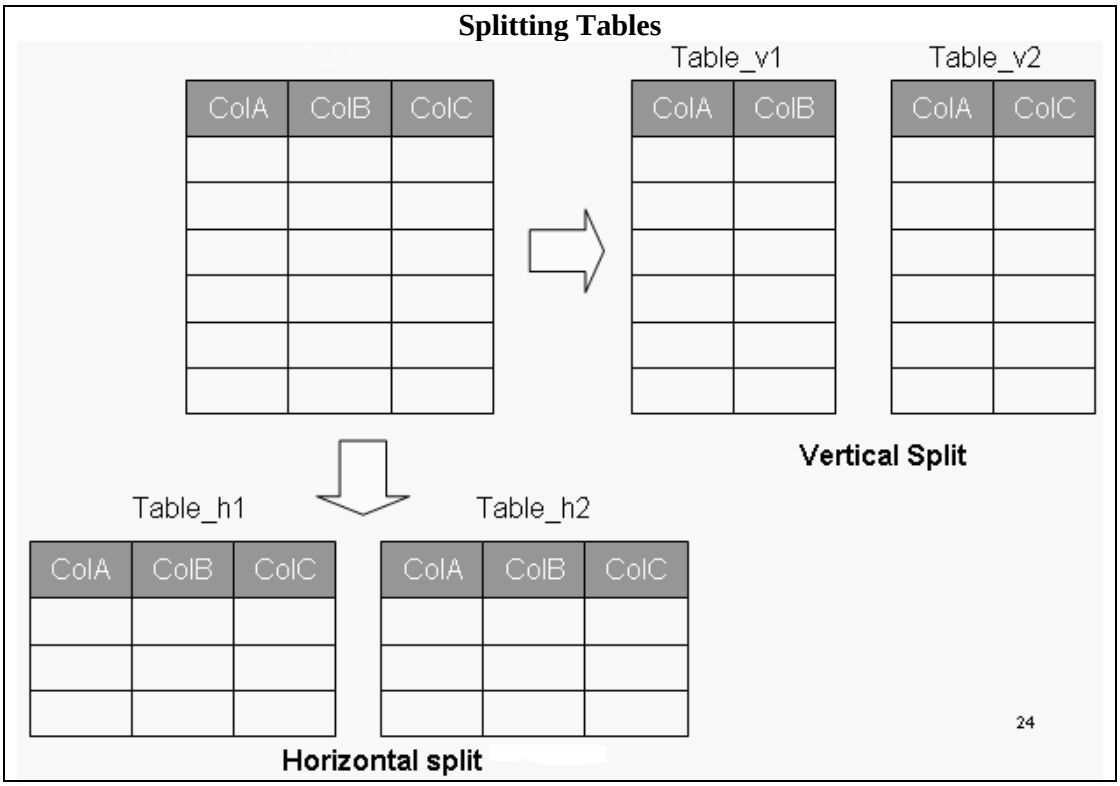


Figure-8.1: Splitting Tables

8.1 Splitting Tables

The denormalization techniques discussed earlier all dealt with combining tables to avoid doing run-time joins by decreasing the number of tables. In contrast, denormalization can be used to create more tables by splitting a relation into multiple tables. Both horizontal and vertical splitting and their combination are possible. This form of denormalization - record splitting- is especially common for a distributed DSS environment.

Splitting Tables: Horizontal splitting Imp Important

Breaks a table into multiple tables based upon common column values. Example: Campus specific queries.

GOAL

- Spreading rows for exploiting parallelism.
- Grouping data to avoid unnecessary query load in WHERE clause.

Splitting Tables

Horizontal splitting breaks a relation into multiple record set specifications by placing different rows into different tables based upon common column values. For the multi-campus example being considered; students from Islamabad campus in the Islamabad table, Peshawar students in corresponding table etc. Each file or table created from the splitting has the same record lay out or header.

Goals of Horizontal splitting:

There are typically two specific goals for horizontal partitioning: (1) spread rows in a large table across many HW components (disks, controllers, CPUs, etc.) in the environment to facilitate parallel processing, and (2) segregate data into separate partitions so that queries do not need to examine all data in a table when WHERE clause filters specify only a subset of the partitions. Of course, what we would like in an ideal deployment is to get both of these benefits from table partitioning.

Advantages of Splitting tables:

Horizontal splitting makes sense when different categories of rows of a table are processed separately: e.g. for the student table if a high percentage of queries are focused towards a certain campus at a time then the table is split accordingly. Horizontal splitting can also be more secure since file level of security can be used to prohibit users from seeing certain rows of data. Also each split table can be organized differently, appropriate for how it is individually used. In terms of page access, horizontally portioned files are likely to be retrieved faster as compared to un-split files, because the latter will involve more blocks to be accessed.

Splitting Tables: Horizontal splitting

ADVANTAGE

- Enhance security of data.
- Organizing tables differently for different queries.
- Reduced I/O overhead.
- Graceful degradation of database in case of table damage.
- Fewer rows result in flatter B-trees and fast data retrieval.

Other than performance, there are some other very useful results of horizontal splitting the tables. As we discussed in the OLAP lecture, security is one of the key features required from an OLAP system. Actually DSS is a multi-user environment, and robust security needs to ensure. By splitting the tables and restricting the users to a particular split actually improves the security of the system. Consider time-based queries, if the queries have to cover last years worth of data, then splitting the tables on the basis of year will defiantly improve the performance as the amount of data to be accessed is reduced. Similarly, if for a multi-campus university, most of the queries are campus specific, then splitting the tables based on the campus would result in improved performance. In both of the cases of splitting discussed, i.e. time and space, as the number of records to be

retrieved is reduced, resulting in more records per block that translates into fewer page faults and high performance. If the table is not partitioned, and for some reason the table is damaged, then in the worst case all data might be lost. However, when the table gets partitioned, and even if a partition is damaged, ALL of the data is not lost. Assuming a worst case scenario that tables crash i.e. all of them, the system will not go down suddenly, but would go down gradually i.e. gracefully.

Splitting Tables: Vertical Splitting

- Splitting and distributing into separate files with repeating primary key.
- Infrequently accessed columns become extra "baggage" thus degrading performance.
- Very useful for rarely accessed large text columns with large headers.
- Header size is reduced, allowing more rows per block, thus reducing I/O.
- For an end user, the split appears as a single table through a view.

Splitting Tables: Vertical Splitting

Vertical splitting involves splitting a table by columns so that a group of columns is placed into the new table and the remaining columns are placed in another new table. Thus columns are distributed into separate files, such that the primary key is repeated in each of the files. An example of vertical splitting would be breaking apart the student registration table by creating a personal_info table by placing SID along with corresponding data into one record specification, the SID along with demographic-related student data into another record specification, and so on.

Vertical splitting can be used when some columns are rarely accessed rather than other columns or when the table has wide rows or header or both. Thus the infrequently accessed columns become extra "baggage" degrading performance. The net result of splitting a table is that it may reduce the number of pages/blocks that need to be read because of the shorter header length allowing more rows to be packed in a block, thus reducing I/O. A vertically split table should contain one row per primary key in the split tables as this facilitates data retrieval across tables and also helps in dissolving the split i.e. making it reversible. In reality, the users are unaware of the split, as view of a joined table is presented to the users.

8.2 Pre-Joining

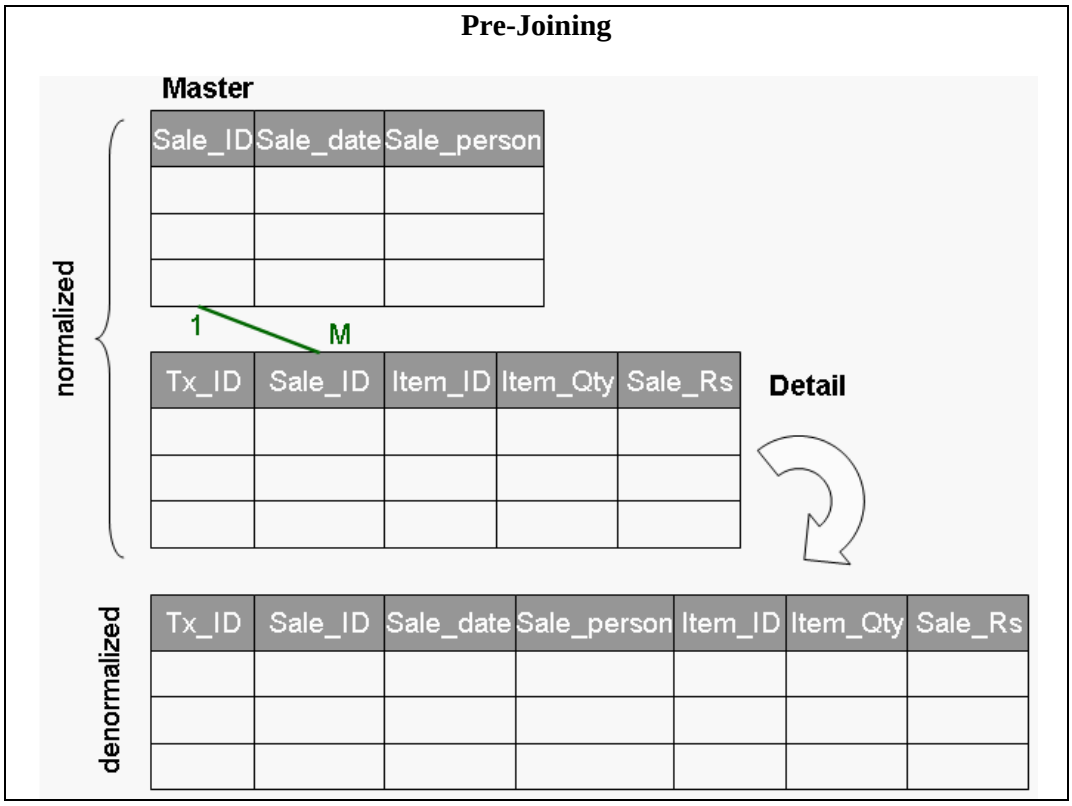


Figure-8.2: Pre-joining Tables

The objective behind pre-joining is to identify frequent joins and append the corresponding tables together in the physical data model. This technique is generally used when there is a one-to-many relationship between two (or more) tables, such as the master-detail case when there are header and detail tables in the logical data model. Typically, referential integrity is assumed from the foreign key in one table (detail) to the primary key in the other table (header).

Additional space will be required, because information on the master table is stored once for each detail record i.e. multiple times instead of just once as would be the case in a normalized design.

Pre-Joining

- Typical of Market basket query
- Join ALWAYS required
- Tables could be millions of rows
- Squeeze Master into Detail
- Repetition of facts. How much?

- Detail 3-4 times of master

Figure-8.2 shows a typical case of market basket querying, with a master table and a detail table. The sale_ID column is the primary key for the master table and uniquely identifies a market basket. There will be one "detail" record for each item listed on the "receipt" for the market basket. The tx_ID column is the primary key for the detail table.

Observe that in a normalized design the store and sale date for the market basket is on one table (master) and the item (along with quantity, sales Rs, etc.) are on a separate table (detail). Almost all analysis will require product, sales date, and (sometimes) sale person (in the context of HR). Moreover, both tables can easily be millions of rows for a large retail outlet with significant historical data. This means that a join will be forced between two very large tables for almost every query asked of the data warehouse. This could easily choke the system and degrade performance.

Note that this same header/detail structure in the data applies across many industries as we have discussed in the very early lectures, such as healthcare, transportation, logistics, billing etc.

To avoid the run-time join, we use the pre-join technique and "squeeze" the sales master information into the detail table. The obvious drawback is repetition of facts from the master table into the detail table. This avoids the join operation at run-time, but stores the header information redundantly as part of the sales detail. This redundant storage is a violation of normalization, but will be acceptable if the cost of storage is less then the performance achieved by virtue of eliminating the join.

8.3 Adding Redundant Columns

This technique can be used when a column from one table is frequently accessed in a large scale join in conjunction with a column from another table. To avoid this join, the column is added (redundant) or moved into the detail table(s) to avoid the join. For example, if frequent joins are performed using Table_1 and Table_2 using columns ColA, ColB and ColC, then it is suitable to add ColC to Table_1.

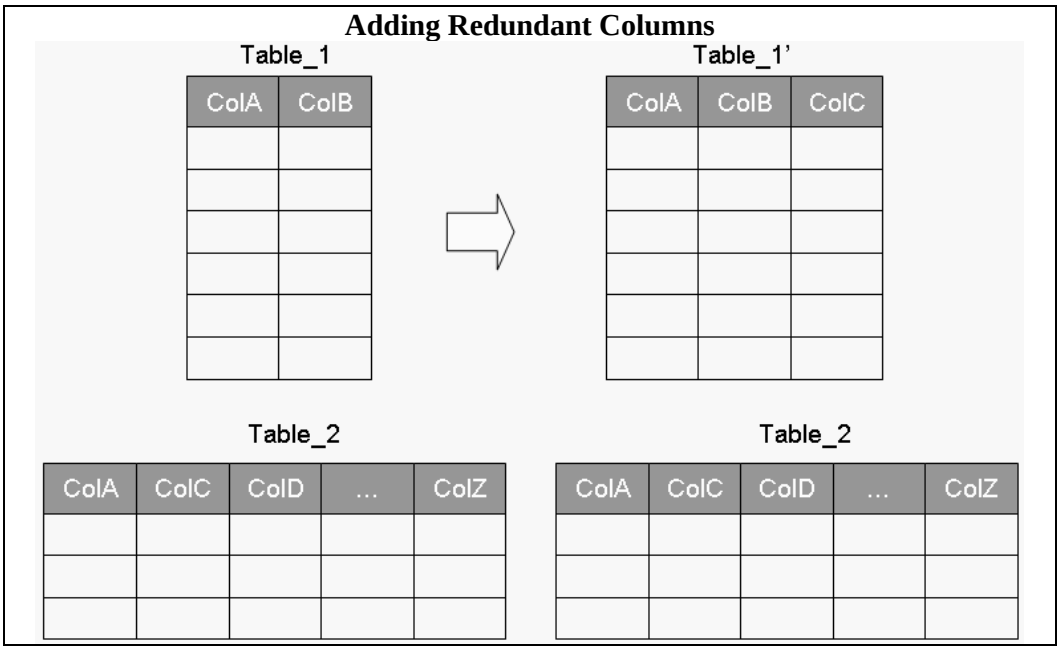


Figure-8.3: Adding redundant columns

Note that the columns can also be moved, instead of making them redundant. If closely observed, this technique is no different from a pre-joining. In pre-joining all columns are moved from the master table into the detail table, but in the current case, a sub-set of columns from the master table is made redundant or moved into the detail table. The performance, and storage trade-offs are also very similar to pre-joining.

Adding Redundant Columns

Columns can also be moved, instead of making them redundant. Very similar to pre-joining as discussed earlier.

EXAMPLE

Frequent referencing of code in one table and corresponding description in another table.

- A join required is required.
- To eliminate the join, a redundant attribute added in the target entity which is functionally independent of the primary key.

A typical scenario for column redundancy/movement is frequent referencing of code in one table and the corresponding description in another table. The description of a code is retrieved via a join. In such a case, redundancy will naturally pay off. This is implemented by duplicating the descriptive attribute in the entity, which would otherwise contain only the code. The result is a redundant attribute in the target entity which is functionally independent of the primary key. Note that this foreign key relationship was created in the first place to normalize the corresponding description reduce update anomalies.

Redundant Columns: Surprise

Note that:

- Actually increases in storage space, and increase in update overhead.
- Keeping the actual table intact and unchanged helps enforce RI constraint.
- Age old debate of RI ON or OFF.

Redundant Columns Surprise

Creating redundant columns does not necessarily reduce the storage space requirements, as neither the reference table is removed, nor the columns duplicated from the reference table.

The reason being to ensure data input RI constraint, although this reasoning falls right in the middle of the age old debate that Referential Integrity (RI) constraint should be turned ON or OFF in a DWH environment. However, it is obvious that column redundancy does eliminate the join and increase the performance.

8.4 Derived Attributes

- Objectives
  - Ease of use for decision support applications
  - Fast response to predefined user queries
  - Customized data for particular target audiences
  - Ad-hoc query support
- Feasible when…
  - Calculated once, used most
  - Remains fairly "constant"
  - Looking for absoluteness of correctness.
  - Pitfall of additional space and query degradation.

Derived Attributes

It is usually feasible to add derived attribute(s) in the data warehouse data model, if the derived data is frequently accessed and calculated once and is fairly stable. The justification of adding derived data is simple; it reduces the amount of query processing time at run-time while accessing the data in the warehouse. Furthermore, once the data is properly calculated, there is little or no apprehension about the authenticity of the calculation. Put in other words, once the derived data is properly calculated it kind of becomes absolute i.e. there is hardly any chance that someone might use a wrong formula to calculate it incorrectly. This actually enhances the credibility of the data in the data warehouse.

Derived Attributes

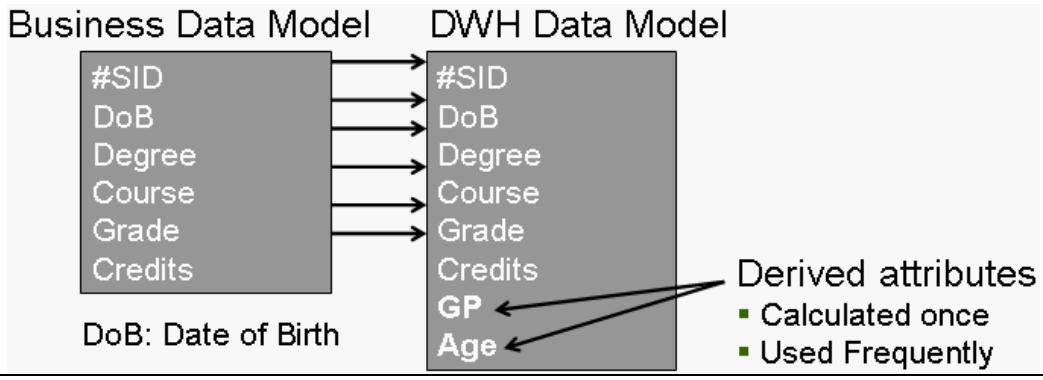


Figure-8.4: Business Data Model vs. DWH Data Model

GP (Grade Point) column in the data warehouse data model is included as a derived value. The formula for calculating this field is Grade*Credits.

Age is also a derived attribute, calculated as Current_Date – DoB (calculated periodically).

In most cases, it will only make sense to use derived data if the ratio of detail rows to derived rows is at least 10:1. In such cases, the 10% storage cost for keeping the derived data is less than the temporary and sort space storage costs for many concurrent queries aggregating at runtime.

Masters Lecture 09: Issues of De-Normalization

Learning Goals

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

The effects of denormalization on database performance are unpredictable: as many applications/users can be affected negatively by denormalization when some applications are optimized. If a decision is made to de-normalize, make sure that the logical model has been fully normalized to 3NF. Also document the pure logical model and keep your documentation of the physical model current as well. Consider the following list of effects of denormalization before you decide to undertake design changes.

The trade-offs of denormalization are as follows:

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

Each of these tradeoffs must be considered when deploying denormalization into a physical design. Typically, architects are pretty good at assessing performance and storage implications of a denormalization decision. Factors that are notoriously under estimated are the maintenance implications and the impact on usability/flexibility for the physical data model.

9.1 Storage Issues: Pre-joining

- 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.

By understanding the characteristics of the data, the storage requirements can actually be quantified before pre-joining. We need to know the size of the data from the master table that will be replicated for pre-joining into the detail table as well as the number of detail records (on average) in the header that will be denormalized as a result of pre-joining.

In this example, it is assumed that that each master table record has two detail record entries associated with it (on average). Note that this ratio will vary depending on the nature of each industry and business within an industry. The health-care industry would be much closer to a 1:2 ratio, depending on if the data is biased towards individual or organizational claims. A 1:3 ratio could be reasonable for a video rental store, but a grocery store with tens of thousands of items, the ratio would typically be on the plus side

of 1:30 detail records for each master table entry. It is important to know the characteristics in your specific environment to properly and correctly calculate the storage requirements of the pre-joining technique.

Storage Issues: Pre-joining
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.

The 12.5% investment in additional storage for pre-joining will dramatically increase performance for queries which would otherwise need to join the very large header and detail tables.

9.2 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.

How the corresponding query will perform with normalization and after denormalization? This a good question, with a surprising answer. Observe that with normalization there are unique values in the master table, and the number of records in the master table is the required answer. To get this answer, there is probably no need to touch that table, as the said information can be picked from the meta-data corresponding to that table. However, it is a different situation after pre-joining has been performed. Now there are multiple i.e. repeating member_IDs in the joined table. Thus accessing the meta-data is not going to help. The only viable option is to perform a count distinct, easier said than done. The reason being this will require a sort operation, and then dropping the repeating value. For large tables, it is going to kill the performance of the system.

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 2.5 times header size.
- Using a table which is 2 times larger.
- Resulting in 5 times degradation in performance.

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

Counter intuitively, the query with pre-joining will perform worse than the normalized design, basically for three reasons.

1. There is no simple way to count the number of distinct customers in the physical data model because this information is now "lost" in the detail table. As a result, there is no choice, but to use a "count distinct" on the member_ID to group identical IDs and then determine the number of unique patients. This is going to be achieved by sorting all qualifying rows (by date) in the denormalized tables. Note that sorting is typically a very expensive operation, the best being O(n log n).

2. The table header of the denormalized detail table is now 90 bytes as opposed to 40 bytes of the master table i.e. an increase of 250%.

3. The number of rows that need to be scanned in the details table are two times as many as compared to the normalized design i.e. scanning the master table. This translates to five times more I/Os in the denormalized scenario versus the normalized scenario!

Bottom line is that the normalized design is likely to perform many times faster as compared to the denormalized design for queries that probe the master table alone, rather than those that perform a join between the master and the detail table. Best and expensive approach would be to also keep the normalized master table, and a smart query coordinator that directs the queries to for increasing performance.

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.

The main problem with redundant columns is that if strict discipline is not enforced, it can very quickly result into chaos. The reason being, every DWH user has their own set of columns which they frequently use in their queries. Once they hear about the performance benefits (due to denormalization) they would want their "favorite" column(s) to be moved/copied into the main fact table in the data warehouse. If this is allowed to happen, sooner than later the fact table would become one large flat file with a header in kilo bytes, and result in degraded performance.

The reason being, each time the table width is increased, the number of rows per block decreases and the amount of I/O increases, and the table access becomes less efficient.

Hence the column redundancy can not be looked into isolation, with a view to benefit a only a subset of the queries. For a number of queries, the performance will degrade by avoiding the join, thus a detailed and quantifiable cost-benefit analysis is required.

9.3 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.

Maintenance is usually overlooked or underestimated while replicating columns. Because the cost of reflecting the change in the member_ID for this design, is considerably high when reflected across the relevant tables. For example, transactions in the detail table need to be updated with the new key, and for a retail warehouse, the detail table could be 30 times larger than the master table, which again is larger then the fact (member table). For an archival system that keeps backup of the historical transactions, maintenance becomes a nightmare, because keeping the member_id data consistent will be very risky.

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.

Hash partitioning is the most common partitioning strategy. Almost all parallel RDBMS products provide some form of built-in hash partitioning capability (mainframe DB2 is the most significant exception to this statement). Horizontal partitioning using a hashing algorithm will assign data rows to partitions according to a "repeatable" random algorithm. In other words, a particular row will always hash to the same partition (assuming that the hashing algorithm and number of partitions have not changed), but a large number of rows will be "randomly" distributed across the partitions as long as a well-selected partitioning key is selected and the hashing function is well-behaved.

Notice that the "random" assignment of data rows across partitions makes it nearly impossible to get any kind of meaningful partition elimination. Since data rows are hash

distributed across all partitions (for load-balancing purposes), there is not practical way to perform partition elimination unless a very small number (e.g., singleton) or data rows is selected from the table via the partitioning key (which doesn't happen often in a traditional DW workload).

Ease of Use Issues: Horizontal Splitting

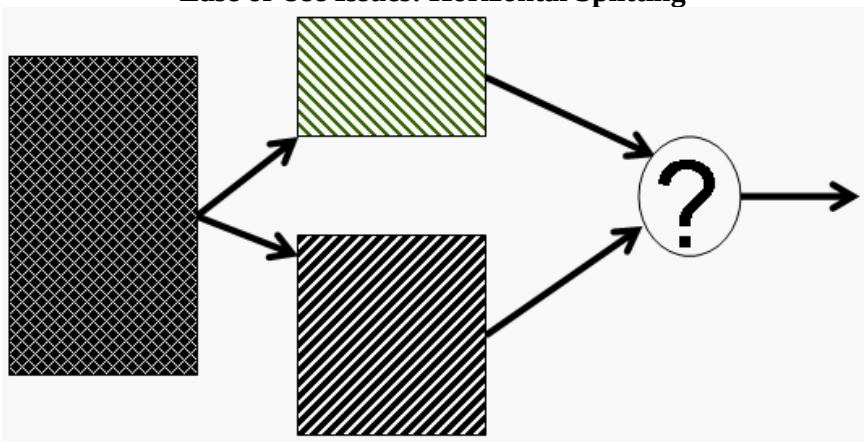


Figure-9.1: Irreversible partitioning

Note that we perform denormalization to get performance for a particular set of queries, and may like to bring the table back to its original form for another set of queries. If this can not be done, then extra effort or CPU cycles would be required to achieve this objective. As shown in Figure-9.1, it is possible to have a partitioning strategy, such that the partitioned tables can not be appended together to get the record sin the original order. This is further explained when we discuss the issues of horizontal partitioning.

Ease of Use Issues: Horizontal Splitting

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

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

Round-robin spreads data evenly across the partitions, but does not facilitate partition elimination (for the same reasons that hashing does not facilitate partition elimination). Round-robin is typically used only for temporary tables where partition elimination is not important and co-location of the table with other tables is not expected to yield performance benefits (hashing allows for co-location, but round-robin does not). Round-robin is the "cheapest" partitioning algorithm that guarantees an even distribution of workload across the table partitions.

The most common use of range partitioning is on date. This is especially true in data warehouse deployments where large amounts of historical data are often retained. Hot spots typically surface when using date range partitioning because the most recent data tends to be accessed most frequently.

Expression partitioning is usually deployed when expressions can be used to group data together in such a way that access can be targeted to a small set of partitions for a significant portion of the DW workload. For example, partitioning a table by expression so that data corresponding to different divisions or LOBs (lines of business) is grouped together will avoid scanning data in divisions or LOBs excluded from the WHERE clause predicates in a DSS query. Expression partitioning can lead to hot spots in the same way as described for range partitioning.

Performance Issues: Horizontal Splitting

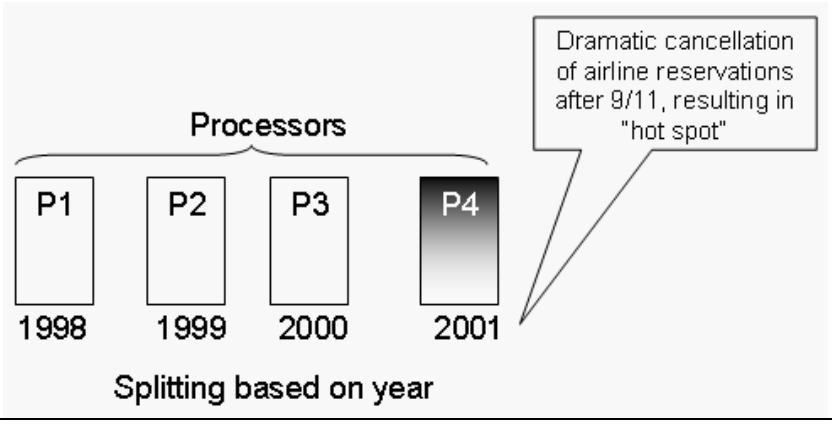


Figure-9.2: De-mertis of horizontal partitoiniong

Performance issues: Vertical Splitting

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+?

Recall in last lecture when we discussed partitioning on the basis of date to enhance query performance, this has its down sides too. Consider the case of airline reservations table horizontally split on the basis of year. After 9/11 people obviously got scared of flying, and there was a surge of cancellations of air line bookings. Thus the most number of cancellations, actually, probably highest ever occurred during the last quarter of year 2001. Thus the corresponding partition would have the largest number of records. Thus in a parallel processing environment, where partitioning is consciously done to improve performance it not going to work, because as shown in Figure-9.2 most of the work is being done by processor P4 which would become a bottleneck. Meaning, unless the results of processor P4 are made available, the overall results can not be combined, while the remaining processors are idling i.e. doing nothing.
Note that there will be a one-to-one relationship between rows in the two portions of the partitioned table.

Performance issues: Vertical Splitting

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.

Scanning the vertically partitioned claim table for frequently accessed data is five times faster than before splitting because the table is five times "thinner" without the infrequently used portions of the data.

However, this performance benefit will only be obtained if we do not have to join to the infrequently accessed portion of the table. In other words, all columns that we need must be in the frequently accessed portion of the table.

Performance issues: Vertical Splitting

Carefully identify and select the columns that get placed on which "side" of the frequently/infrequently used "divide" between the splits.

Moving a single five byte column to the frequently used table split (20 byte width) means that ALL table scans against the frequently used table will run 25% slower.

Don't forget the additional space required for the join key, this becomes significant for a billion row table.

Also, be careful when determining frequency of use. You may have 90% of the queries accessing columns in the "frequently used" partition of the table. However, the important measure is the percent of queries that access only the frequently used portion of the table with no columns required from the infrequently used data.

Masters

Subscribe to masters

Masters

Lecture 10: Online Analytical Processing (OLAP)

Learning Goals

- What is OLAP?
- Relationship between DWH & OLAP
- Data Warehouse & OLAP go together.
- Analysis supported by OLAP

OLAP is Analytical Processing instead of Transaction Processing. It is also NOT a physical database design or implementation technique, but a framework. Furthermore OLAP implementations are either highly de-normalized or completely de-normalized, while OLTP implementations are fully normalized. Therefore, a clear understanding of how OLAP (On-Line Analytical Processing) fits into the overall architecture of your data warehouse is essential for appropriate deployment. There are different methods of deployment of OLAP within a decision support environment - often obscured by vendor hype and product salesmanship. It is important that you as data warehouse architect drive your choice of OLAP tools - not driven by it!

10.1 DWH & OLAP

- Relationship between DWH & OLAP
- Data Warehouse & OLAP go together.
- Analysis supported by OLAP

A data warehouse is a "subject-oriented, integrated, time varying, non-volatile collection of data that is used primarily in organizational decision making. Typically, the data warehouse is maintained separately from the organization's operational databases and on different (and more powerful) servers. There are many reasons for doing this. The data warehouse supports on-line analytical processing (OLAP), the functional and performance requirements of which are quite different from those of the on-line transaction processing (OLTP) applications, traditionally supported by the operational data bases.

A data warehouse implementation without an OLAP tool is nearly unthinkable. Data warehousing and on-line analytical processing (OLAP) are essential elements of decision support, and go hand-in-hand. Many commercial OLAP products and services are available, and all of the principal database management system vendors have products in these areas. Decision support systems are very different from MIS systems, consequently this places some rather different requirements on the database technology compared to traditional on-line transaction processing applications.

Data Warehouse provides the best support for analysis while OLAP carries out the analysis task. Although there is more to OLAP technology than data warehousing, the classic statement of OLAP is "decision making is an iterative process; which must involve the users". This statement implicitly points to the not so obvious fact that while OLTP systems keep the users' records and assist the users in performing the routine operations tasks of their daily work. OLAP systems on the other hand generate information users require to do the non-routine parts of their jobs. OLAP systems deal

with unpredicted circumstances (for which if not impossible, it is very difficult to plan ahead). If the data warehouse doesn't provide users with helpful information quickly in an intuitive format, the users would be turned down by it, and will revert back to their old and faithful, yet inefficient but tried out procedures. Data warehouse developers sometimes refer to the importance of intuitiveness to the users as the "field of dreams scenario" i.e. naively assuming that just because you build it doesn't mean they will come and use it.

Supporting the human thought process

We will first learn the nature of analysis before going on to the actual structure and functionality behind OLAP systems; this is discussed in Figure-10.1. Consider a hypothetical (but a realistic scenario) when a decision maker (analyst, CEO) of a large enterprise i.e. a multinational company is confronted with the yearly sales figure that there is an enterprise wide fall in profit. Note that we are talking about a large multinational company, which is not only in sales, but also in production. To get a hold of the problem, decision maker has to start from somewhere, but that somewhere has to be meaningful too. This "somewhere" could be starting the exploration across the enterprise geographically or exploring across the enterprise functionally or doing it product-wise or doing it with reference to time.

One of the valid options of starting is with reference to time, and the query corresponding to the thought process of the decision maker turns out to be "What was the quarterly sales during last year?" The result of this query would be sales figures for the four quarters of last year. There can be a number of possible outcomes, for example, the sale could be down in all the quarters, or it could be down in three out of four quarters with four possible options. Another possibility could be sales down in any two quarters with six possible scenarios. For the sake of simplicity, over here we assume that the sales were down in only one quarter. If the results were displayed as histograms, the decision maker would immediately pick the quarter in which sales were down, which in this case comes out to be the last quarter.

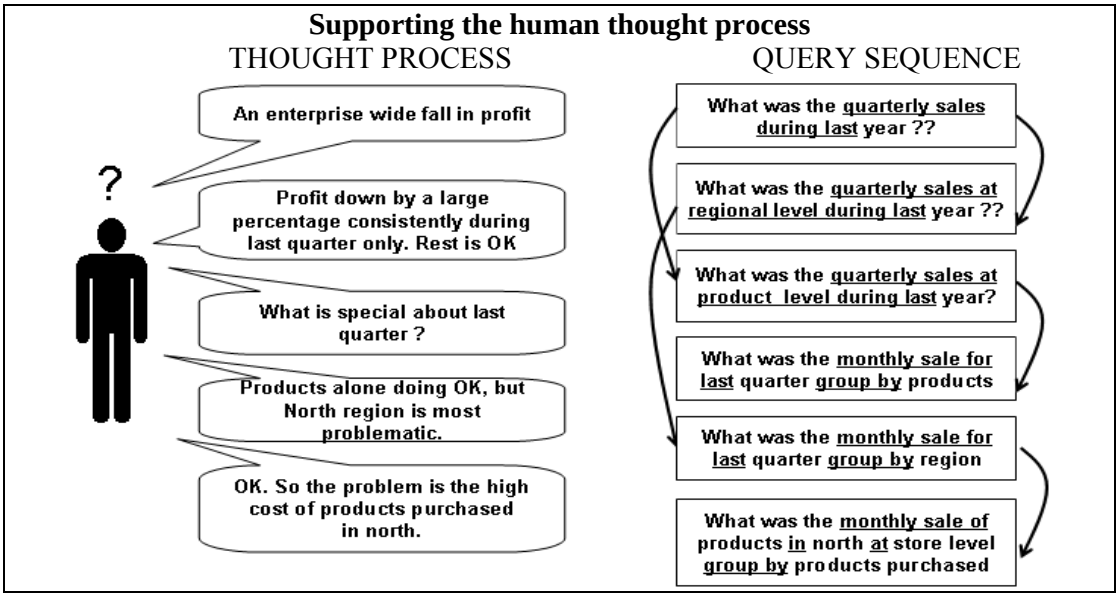


Figure-10.1: Supporting the human thought process

Now that the "lossy" quarter has been identified, the decision maker would naturally like to probe further, to go to the root of the problem, and would like to know what happened during the last quarter that resulted into a fall in profit? The objective here is to identify the root cause, once identified, the problem can be fixed. So the obvious question that comes into the mind of the decision maker is *what is special about the last quarter?* Before jumping directly into the last quarter to come up with hasty outcomes, the careful decision maker wants to look at the sales data of the last year, but with a different perspective, therefore, his/her question is translated into two queries i.e. (i) What was the quarterly sales at regional level during last year? (ii) What was the quarterly sales at product level during last year? There could be another sequence of questions i.e. probing deeper into the "lossy" quarter directly, which is followed in the next sequence of queries.

It may seem that the decision maker was overly careful, but he was not. Recognize that the benefit of looking at a whole year of data enables the decision maker to identify if there was a trend of sales going down, that resulted in least sales during the last quarter or the sales only went down during the last quarter? The finding was that the sales went down only during the last quarter, so the decision maker probed further by looking at the sales at monthly basis during the last quarter with reference to product, as well as with reference to region. Here is a finding, the sales were doing all right with reference to the product, but they were found to be down with reference to the region, specifically the Northern region.

Now the decision maker would like to know what happened in the Northern region in the last quarter with reference to the products, as just looking at the Northern regional alone would not give the complete picture. Although not explicitly shown in the figure-?, but the current exploratory query of the decision maker is further processed by taking the sales figures of last quarter on monthly basis and coupled together with the product purchased at store level. Now the hard work pays off, as the problem has finally been identified i.e. high cost of products purchased during the last quarter in the Northern region. Well actually it was not hard work in its true literal meaning, as using an OLAP tool, this would have taken not more than a couple of minutes, maybe even less.

10.2 Analysis of last example

- Analysis is Ad-hoc
- Analysis is interactive (user driven)
- Analysis is iterative
  - Answer to one question leads to a dozen more

- Analysis is directional
  - Drill Down
  - Roll Up
  - Drill Across

Now if we carefully analyze the last example, some points are very clear, the first one being analysis is ad-hoc i.e. there is no known order to the sequence of questions. The order of the questions actually depends on the outcome of the questions asked, the domain knowledge of the decision maker, and the inference made by the decision maker

about the results retrieved. Furthermore, it is not possible to perform such an analysis with a fixed set of queries, as it is not possible to know in advance what questions will be asked, and more importantly what answers or results will be presented corresponding to those questions. In a nut-shell, not only the querying is ad-hoc, but the universal set of pre-programmed queries is also very large, hence it is a very complex problem.

As you would have experienced while using an MIS system or done while developing and MIS system, such systems are programmer driven. One would disagree with that, arguing that development comes after system analysis, when user has given his/her requirements. The problem with decision support systems is that, the user is unaware of their requirements, until they have used the system. Furthermore, the number and type of queries are fixed as set by the programmer, thus if the decision maker has to use the OLTP system for decision making, he is forced to use only those queries that have been pre-written thus the process of decision making is driven not by the decision maker. As we have seen, the decision making is interactive i.e. there is noting in between the decision support system and the decision maker.

In a traditional MIS system, there is an almost linear sequence of queries i.e. one query followed by another, with a limited number of queries to go back to with almost fixed and rigid results. As a consequence, running the same set of queries in the same order will not result in something new. As we discussed in the example, the decision support processes is exploratory, and depending on the results obtained, the queries are further fine-tuned or the data is probed deeply, thus it is an iterative process. Going through the loop of queries as per the requirement of the decision maker fine tunes the results, as a result the process actually results in coming up with the answers to the questions.

10.3 Challenges

- Not feasible to write predefined queries.
  - Fails to remain user_driven (becomes programmer driven).
  - Fails to remain ad_hoc and hence is not interactive.
- Enable ad-hoc query support
  - Business user can not build his/her own queries (does not know SQL, should not know it).
  - On_the_go SQL generation and execution too slow.

Now that we have identified how decision making proceeds, this could lure someone with an OLTP or MIS mindset to approach decision support using the traditional tools and techniques. Let's look at this (wrong) approach one point at a time. We identified one very important point that decision support requires much more and many more queries as compared to an MIS system. But even if more queries are written, the queries still remain predefined, and the process of decision making still continues to be driven by the programmer (who is not a decision maker) instead of the decision maker. When something is programmed, it kind of gets hard-wired i.e. it can not be changed significantly, although minor changes are possible by changing the parameter values. That's where the fault lies i.e. hard-wiring. Decision making has no particular order associated with it.

One could naively say OK if the decision making process has to be user driven, let the end user or business user do the query development himself/herself. This is a very unrealistic assumption, as the business user is not a programmer, does not know SQL, and does not need to know SQL. An apparently realistic solution would be to let the SQL generation be on-the-go. This approach does make the system user driven without having the user to write SQL, but even if on-the-go SQL generation can be achieved, the data sets used in a data warehouse are so large, that it is no way possible for the system to remain interactive!

Challenges

- Contradiction
  - Want to compute answers in advance, but don't know the questions
- Solution
  - Compute answers to "all" possible "queries". But how?
  - NOTE: Queries are multidimensional aggregates at some level

So what are the challenges? We have a paradox or a contradiction here. OK we manage to assemble a very large team of dedicated programmers, with an excellent project manager and all sit and plan to write all queries in advance. Can we do that? The answer is a resounding NO. The reason being, since even the user dose not knows the queries in advance, and consequently the programmers also do not know the queries in advance, so how can the queries be written in advance? It is just no possible.

So what is the solution then? We need a paradigm shift from the way we have been thinking. In a decision support environment, we are looking at the big picture; we are interested in all sales figures across a time interval or at a particular time interval, but not for a particular person. So the solution lies in computing all possible aggregates, which would then correspond to all possible queries. Over here a query is a multidimensional aggregate at a certain level of the hierarchy of the business.

To understand this concept, let's look at Figure-10.2 that shows multiple levels or hierarchies of geographies. Pakistan is logically divided into provinces, and each province is administratively divided into divisions, each division is divided into districts and within districts are cities. Consider a chain store, with outlets in major cities of Pakistan, with large cities such as Karachi and Lahore having multiple outlets, which you may have observed also. Now the sales actually take place at the business outlet which could be in a zone, such as Defense and Gulberg in our example, of course there may be many more stores in a city, as already discussed.

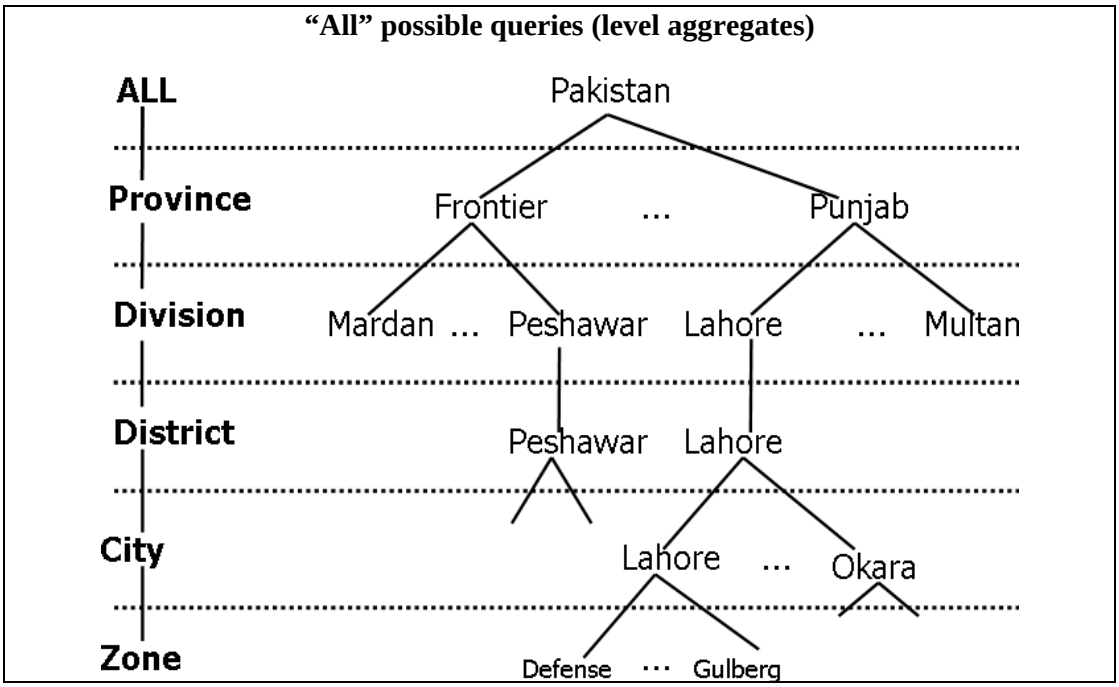


Figure-10.2: Hierarchies in data

10.4 OLAP: Facts & Dimensions

- FACTS: Quantitative values (numbers) or "measures."
  - e.g., units sold, sales $, Co, Kg etc.

- DIMENSIONS: Descriptive categories.
  - e.g., time, geography, product etc.
  - DIM often organized in hierarchies representing levels of detail in the data (e.g., week, month, quarter, year, decade etc.).

The foundation for design in this environment is through use of dimensional modeling techniques which focus on the concepts of "facts" and "dimensions" for organizing data. Facts are the quantities, numbers that can be aggregated (e.g., sales $, units sold etc.) that we measure and dimensions are how we filter/report on the quantities (e.g., by geography, product, date, etc.). We will discuss in detail, actually have number of lectures on dimensional modeling techniques.

10.5 Where Does OLAP Fit In?

- It is a classification of applications, NOT a database design technique.
- Analytical processing uses multi-level aggregates, instead of record level access.
- Objective is to support very
- (i) fast
- (ii) iterative and
- (iii) ad-hoc decision-making.

OLAP is a characterization of applications. It is NOT a database design technique. People often confuse OLAP with specific physical design techniques or data structure. This is a mistake. OLAP is a characterization of the application domain centered around slice-and-

dice and drill down analysis. As we will see, there are many possible implementations capable of delivering OLAP characteristics. Depending on data size, performance requirements, cost constraints, etc. the specific implementation technique will vary.

Where does OLAP fit in?

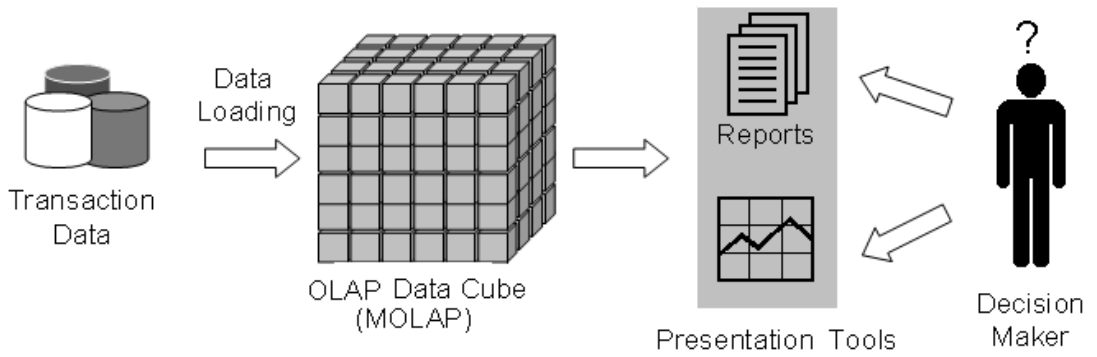


Figure-10.3: How does OLAP pieces fir together

Figure-10.3 shows that using the transactional databases (after Extract Transform Load) OLAP data cubes are generated. Over here multidimensional cube is shown i.e. a MOLAP cube. Data retrieved as a consequence of exploring the MOLAP cube is then used as a source to generate reports or charts etc. Actually in typical MOLAP environments, the results are displayed as tables, along with different types of charts, such as histogram, pie etc.

10.6 Difference between OLTP and OLAP

OLTP vs. OLAP

Feature	OLTP	OLAP
Level of data	Detailed	Aggregated
Amount of data retrieved per transaction	Small	Large
Views	Pre-defined	User-defined
Typical write operation	Update, insert, delete	Bulk insert, almost no deletion
"age" of data	Current (60-90 days)	Historical 5-10 years and also current
Tables	Flat tables	Multi-Dimensional tables
Number of users	Large	Low-Med
Data availability	High (24 hrs, 7 days)	Low-Med
Database size	Med (GB- TB)	High (TB – PB)
Query Optimizing	Requires experience	Already "optimized"

Table-10.1: Difference between OLTP and OLAP

The table summarizes the fundamental differences between traditional OLTP systems and typical OLAP applications. In OLTP operations the user changes the database via transactions on detailed data. A typical transaction in a banking environment may transfer money from one account to another account. In OLAP applications the typical user is an analyst who is interested in selecting data needed for decision support. He/She is primarily not interested in detailed data, but usually in aggregated data over large sets of data as it gives the big picture. A typical OLAP query is to find the average amount of money drawn from ATM by those customers who are male, and of age between 15 and 25 years from (say) Jinnah Super Market Islamabad after 8 pm. For this kind of query there are no DML operations and the DBMS contents do not change.

10.7 OLAP FASMI Test

Fast: Delivers information to the user at a fairly constant rate i.e. O(1) time. Most queries answered in under 5 seconds.

Analysis: Performs basic numerical and statistical analysis of the data, pre-defined by an application developer or defined ad-hocly by the user.

Shared: Implements the security requirements necessary for sharing potentially confidential data across a large user population.
Multi-dimensional: The essential characteristic of OLAP.

Information: Accesses all the data and information necessary and relevant for the application, wherever it may reside and not limited by volume.

Explanation:

FAST means that the system is targeted to deliver response to moat of the end user queries under 5 seconds, with the simplest analyses taking no more than a second and very few queries taking more than 20 seconds (for various reasons to be discussed). Because if the queries take significantly longer time, the thought process is broken, as users are likely to get distracted, consequently the quality of analysis suffers. This speed is not easy to achieve with large amounts of data, particularly if on-the-go and *ad-hoc* complex calculations are required.

ANALYSIS means that the system is capable of doing any business logic and statistical analysis, which is applicable for the application and to the user, and also keeps it easy enough for the user i.e. at the point-and-click level. It is absolutely necessary to allow the target user to define and execute new *ad hoc* calculations/queries as part of the analysis and to view the results of the data in any desired way/format, without having to do any programming.

SHARED means that the system is not supposed to be a stand-alone system, and should implement all the security requirements for confidentiality (possibly down to cell level) for a multi-user environment. The other point is kind of contradiction of the point discussed earlier i.e. writing back. If multiple DML operations are needed to be performed, concurrent update locking should be executed at an appropriate level. It is true that not all applications require users to write data back, but it is true for a considerable majority.

MULTIDIMENSIONAL is the key requirement to the letter and at the heart of the cube concept of OLAP. The system must provide a multidimensional logical view of the aggregated data, including full support for hierarchies and multiple hierarchies, as this is certainly the most logical way to analyze organizations and businesses. There is no "magical" minimum number of dimensions that must be handled as it is too application specific.

INFORMATION is all of the data and derived information required, wherever it is and however much is relevant for the application. The objective over here is to ascertain the capacity of various products in terms of how much input data they can handle and process, instead of how many Gigabytes of raw data they can store. There are many considerations here, including data duplication, Main Memory required, performance, disk space utilization, integration with data warehouses etc.

© Copyright Virtual University of Pakistan

Masters Lecture 11: Multidimensional OLAP (MOLAP)

Learning Goals

- OLAP implemented with a multi-dimensional data structure.
- OLAP implemented with a relational database.
- OLAP implemented as a hybrid of MOLAP and ROLAP.
- OLAP implemented for desktop decision support environments

11.1 OLAP Implementations

1. **MOLAP:** OLAP implemented with a multi-dimensional data structure.

2. **ROLAP:** OLAP implemented with a relational database.

3. **HOLAP:** OLAP implemented as a hybrid of MOLAP and ROLAP.

4. **DOLAP:** OLAP implemented for desktop decision support environments.

MOLAP physically builds "cubes" for direct access - usually in the proprietary file format of a multi-dimensional database (MDD) or a user defined data structure. therefore ANSI SQL is not supported.

ROLAP or a Relational OLAP provides access to information via a relational database using ANSI standard SQL.

HOLAP provides a combination of relational database access and "cube" data structures within a single framework. The goal is to get the best of both MOLAP and ROLAP: scalability (via relational structures) and high performance (via pre-built cubes).

DOLAP allows download of "cube" structures to a desktop platform without the need for shared relational or cube server. This model facilitates a mobile computing paradigm. DOLAP is particularly useful for sales force automation types of applications by supporting extensive slide and dice. A DOLAP implementation needs to be much more efficient in disk and memory utilization than typical server implementations because computing power is often limited on a laptop computer.

11.2 MOLAP Implementations

OLAP has historically been implemented using a multidimensional data structure or "cube".

- Dimensions are key business factors for analysis:
  - Geographies (city, district, division, province,...)
  - Products (item, product category, product department,...)
  - Dates (day, week, month, quarter, year,...)

- Very high performance achieved by O(1) time lookup into "cube" data structure to retrieve pre_aggregated results.

Multi-dimensional databases (MDDs) typically use proprietary file formats to store pre-summarized cube structures. There are dozens of vendors who provide MDD products. Essbase from Hyperion and Powerplay from Cognos are some examples of market share leaders in this space. Some RDBMS vendors have also begun packaging MDD capabilities into their product offerings. Express from Oracle (acquired from IRI) and the MOLAP Store in Microsoft's OLAP Server architecture (acquired from Panorama) are examples of products from RDBMS vendors.

MOLAP cubes are built around dimensions, as corresponding to each dimension is the index of the multi-dimensional array data structure. Meaning, if there are four dimensions, then the data will be stored in a four dimensional array i.e. with four indexes i, j, k and l. The other significance of dimension from the business point of view are the key factors they correspond to, typically dimensions are the features on which the WHERE clause is executed. Some examples of dimension are geography, time, products etc. Note that typically dimensions have hierarchies, for example geography dimension has the hierarchy of country, then province, then division, district, city and zone as discussed in the last lecture.

The performance in a MOLAP cube comes from the O(1) look-up time for the array data structure. Recall that to access an array, only the indexes are required i.e. there is no scanning of the array (like a file data structure), there is no hashing it a constant access time operations, similar to a random access memory (or RAM). The only time the time complexity goes beyond O(1) is when the cube size is so large that it can not fit in the main memory, in such a case a page or a block fault will occur.

MOLAP Implementations

- No standard query language for querying MOLAP
  - *No SQL !*
- Vendors provide proprietary languages allowing business users to create queries that involve pivots, drilling down, or rolling up.
  - E.g. MDX of Microsoft

As it should be clear by now that, in a MOLAP environment there are no tables, there are no traditional relational structures, hence ANSI SQL can not be used. As a matter of fact, there is no standard query language for querying a MOLAP cube. The cube is sometimes probed by a complex mapping mechanism that maps the user requirements into a set of indices, which are then used to retrieve the aggregate from the corresponding matrix location. However, vendors have come up with proprietary languages for non IT business users, the most popular being MDX by Microsoft. These "languages" are graphical environments where the business users can click on actions and perform drag-and-drop operations to provide input to retrieve data/information from the cube. APIs are also available in the market for programming cube data retrieval by more experienced programmers for running of complex quires. But none of these approaches use SQL.

© Copyright Virtual University of Pakistan

Aggregations in MOLAP

- Sales volume as a function of (i) product, (ii) time, and (iii) geography
- A cube structure created to handle this.

Figure-11.1: Filled cube

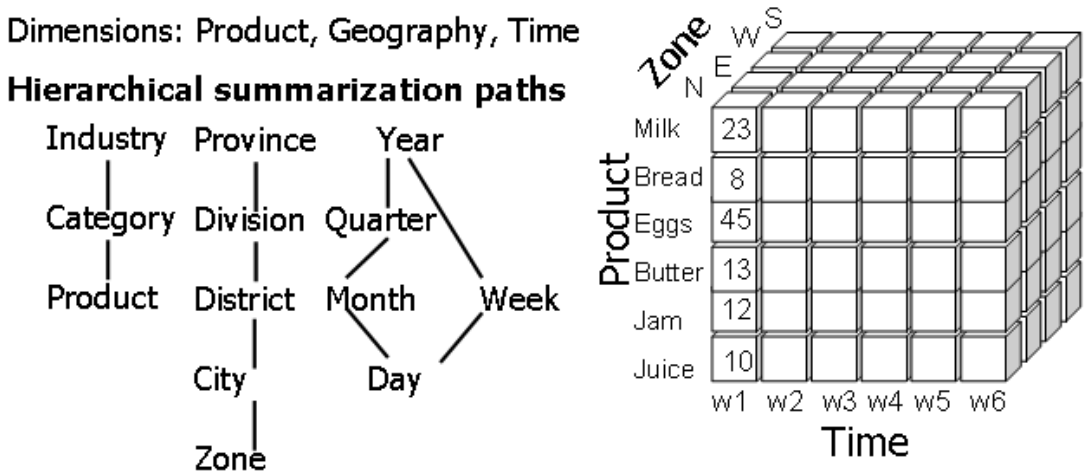


Figure11.1: Aggregation and a MOLAP cube

As already stated a number of times, in a Data Warehouse decision support environment we are interested in the big picture, we want to look at the data from a macro level instead of the micro level. For a macroscopic view aggregates are used. In this example we look at the sales volume i.e. number of items sold as a function of product, time and geography. Note that all three of them are dimensions. The proceed with the analysis, a cube structure will be first created such that each dimension of the cube will correspond to each identified dimension, and within each dimension will be the corresponding hierarchy. The example further shows how the dimensions are "rolled-out" i.e. Province into divisions, then division into district, then district into city and finally cities into zones. Note that weeks could be rolled into a year and at the same time months can be rolled into quarters and quarters rolled into years. Based on these three dimensions a cube is created and shown in figure-11.1.

11.3 Cube Operations *Important*

- Rollup: summarize data
  - e.g., given sales data, summarize sales for last year by product category and region
- Drill down: get more details
  - e.g., given summarized sales as above, find breakup of sales by city within each region, or within Sindh
- Slice and dice: select and project
  - e.g.: Sales of soft-drinks in Karachi during last quarter
- Pivot: change the view of data

The word cube is synonymously used with a MOLAP, therefore, when we talk of cube operations we are never referring to SQL based querying, instead how we view the data stored in a cube. There are four fundamental cubes operations which are (i) rollup (ii) drill down (iii) slice and dice and (iv) pivoting. In the rollup operation the level of aggregation is increased i.e. data is presented at a higher level of aggregation with less detail. For example while looking at the weekly sales data, the decision maker decides to look at the monthly sales data, this is a rollup operation. On the other hand, while looking at the weekly sales data, the decision makers wants to look at the higher level of detail thus he/she drills down and looks at the daily sales data or was looking at the division level of the sales data, and drills down into district level sales data by looking at the sales in different cities corresponding to a district, among one of the many districts present in a division. Slice and dice is a combination of looking at a subset of data based on more than one dimension. As shown sin the slide, a combination of geography and time is used to look at part of the cube across more than a singe dimension. Finally pivoting is changing the view of the data i.e. replacing the dimension across the (say) x-axis. This will become clear when we look at specific examples in the following slides.

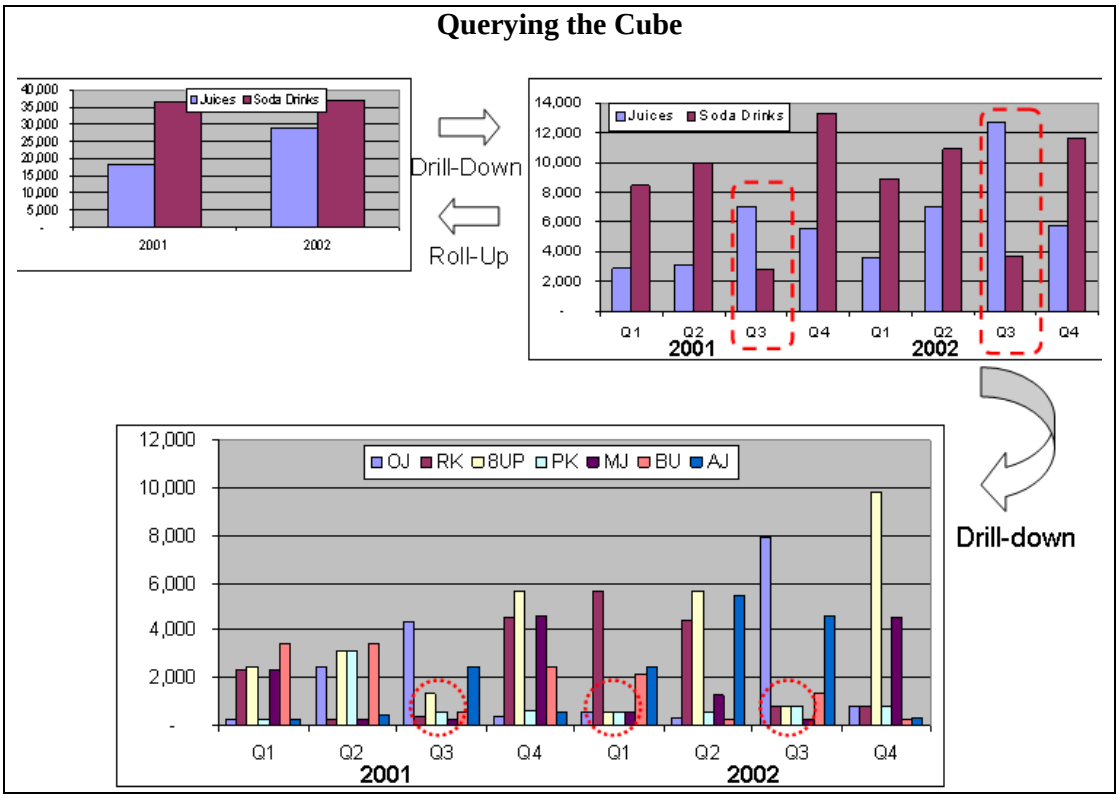


Figure-11.2: Different cube operations

Question: Juices are selling well or the sodas?
Looking at the highest level summary, with the lowest data granularity, shows that the sales of Soda Drinks are higher for years 2001 and 2002.

Question: Which sodas are doing well?
Just clicking on the Sodas column shows the second histogram. It shows that the sales of 8-UP cola are highest in both years i.e. 2001 and 2002.

Question: Is 8-UP a clear winner?
Flipping the dimensions i.e. instead of product on x-axis putting time on x-axis and further drilling-down shows that there is a trend of increasing sales of 8-UP, BUT the sale is alarmingly down during QTR3 in 2001 and QTR1 and QTR3 during 2002.

All of the above information was extracted by exploring 46 table entries and three levels of aggregates. The total exploration time took less than 3 seconds.

Question: Why erratic sales of 8-UP?
This may probably require exploring other dimensions or their combinations.

Querying the Cube: Pivoting

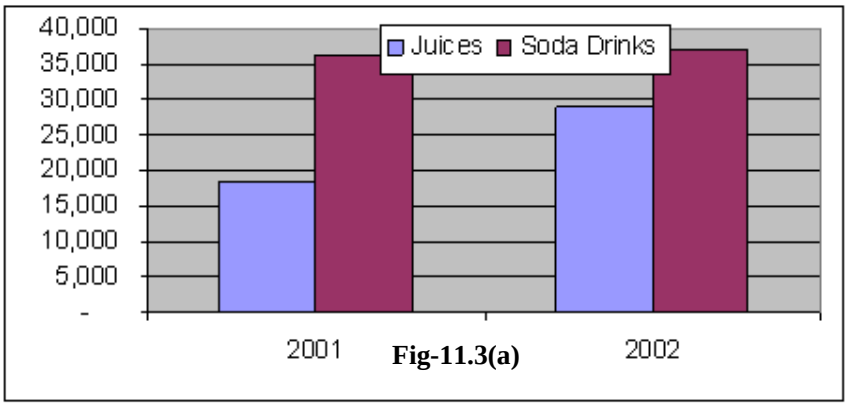


Fig-11.3(a)

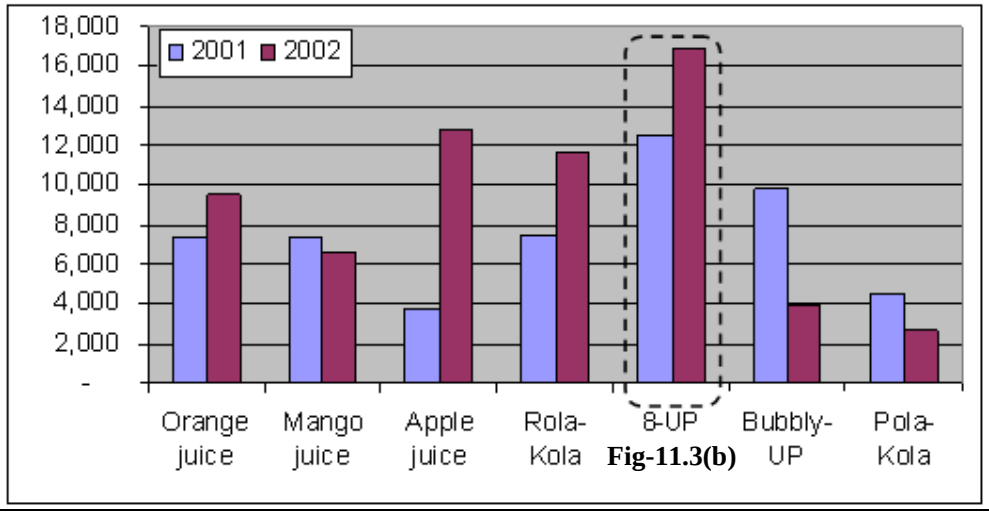


Fig-11.3(b)

Figure-11.3: Pivot cube operations

This is a good example of pivoting. In the first figure (11.3(a)), along x-axis we have the time dimension, y-axis is the sales volumes, but volumes for the product are under two categories i.e. juices and soda drinks. By the pivoting operation, the dimensions are interchanged, and we end up with having the product along the x-axis and the yearly sales figures (i.e. time dimension) along the y-axis. The purpose of pivoting and other operations are not just to provide a set of different operations, but to enable the decision maker to look at the data from different angles. From fig-11.3(a) it seems that the sales of soda drinks are up consistently through year 2001 and 2002, but after pivoting and doing

a drilldown operation, we see that the sales of apple juice was higher than all soda drinks, except 8-UP, but the sale of 8-UP was so high both in years 2001 and 2002, that it shot up the overall sales of soda drinks. Thus all soda drinks are not the winners; actually Pola-Kola has the weakest sales of all.

11.4 MOLAP Evaluation

Advantages of MOLAP:

- Instant response (pre-calculated aggregates).
- Impossible to ask question without an answer.
- Value added functions (ranking, % change).

MOLAP implementations with pre-defined cubes as pre-aggregated data perform very well when compared to relational databases, but often have difficulty scaling when the size of dimensions becomes large. The breakpoint is typically around 64,000. Some implementations are also limited to a file size for the cube representation that must be less than 2GB (this is less often an issue than a few years ago).

Actually the cubes are less than 1% full.

Drawbacks of MOLAP:

- Long load time (pre-calculating the cube may take days!).
- Very sparse cube (wastage of space) for high cardinality (sometimes in small hundreds). e.g. number of heaters sold in Jacobabad or Sibi.

A MOLAP is in no way a win-win situation, it has its won intrinsic pitfalls, which does not make it an overall winner. The biggest drawback is the extremely long time taken to pre-calculate the cubes, remember that in a MOLAP all possible aggregates are computed. The number of aggregates suffers from something called as the "curse of dimensionality" i.e. as the number of dimensions increases, the number of possible aggregates increases exponentially, this will be further clarified in an upcoming slide. Because of long calculation times, cube creation is a batch process and usually has the lowest priority and scheduled to be done late in the night. This actually turns out to be a big mistake (as we will discuss subsequently) as it may so happen that the cube generation may not actually take place, and the decision makers are presented with the old and stale data, and they tend to lose faith in the OLAP paradigm. Although the number of possible aggregates is very large, but NOT all the aggregates may have values, there can be and will be quite a few aggregates which will have null values. For example, many of the items sold in winter are not sold in summer and not even kept in the store (and vice-a-versa). Consequently, there are no corresponding sales, and if the cube is generated that includes all the items, there will be many null aggregates, resulting in a very sparse cube. This will result in requirement of large amount of memory, most of which would be wasted. For these very reasons cube compression is a hot area of research and development. We will not discuss it any further.

11.5 MOLAP Implementation Issues

Maintenance issue: Every data item received must be aggregated into *every* cube (assuming "to-date" summaries are maintained). Lot of work.

Storage issue: As dimensions get less detailed (e.g., year vs. day) cubes get much smaller, but storage consequences for building hundreds of cubes can be significant. Lot of space.

Scalability:
Often have difficulty scaling when the size of dimensions becomes large. The breakpoint is typically around 64,000 cardinality of a dimension.

Maintenance Considerations: Every data item received into MDD must be aggregated into *every* cube (assuming "to-date" summaries are maintained). Have to update the cubes every time a change occurs (insert, delete, update), otherwise the user will run into synchronization problems.

Storage Considerations: Although cubes get much smaller (e.g., more dense) as dimensions get less detailed (e.g., year vs. day), storage implications for building hundreds of cubes can be significant. The reason being, as each possible combination of dimensions has to be pre-calculated i.e. every combination of every aggregate, so soon faced with a combinatorial explosion. Therefore, the number of entries in each dimension has to be of some reasonable number. If you have entries above 100 in each dimension then things tend to blow up. When does it actually blows up depends on the tool being used and how sparse matrices are stored. So you might end up requiring more cube space then allocated or even available. Consider taking each data element by week, and then 65 weeks times' geography is one cube. Now consider taking each data element by month, this is a different cube i.e. by geography by 12 months, and then a different cube by geography by year and so on. So the combinatorial explosion gets way out of hand.

Scalability:
MOLAP implementations with pre-defined cubes as pre-aggregated data perform very well when compared to relational databases, but often have difficulty scaling when the size of dimensions becomes large. The breakpoint is typically around 64,000 cardinality of a dimension. Typically beyond tens (sometimes small hundreds) of thousands of entries in a single dimension will break the MOLAP model because the pre-computed cube model does not work well when the cubes are very sparse in their population of individual cells. Some implementations are also limited to a file size for the cube representation that must be less than 2GB (this is less often an issue than a few years ago). You just can not build cubes big enough, or enough of them to have every thing pre-computed. So you get into the problems of scale. As already discussed, it is difficult to scale because of combinatorial explosion in the number and size of cubes when dimensions of significant cardinality are required. There are two possible, but limited solutions addressing the scalability problem i.e. Virtual cubes and partitioned cubes.

11.6 Partitioned Cubes

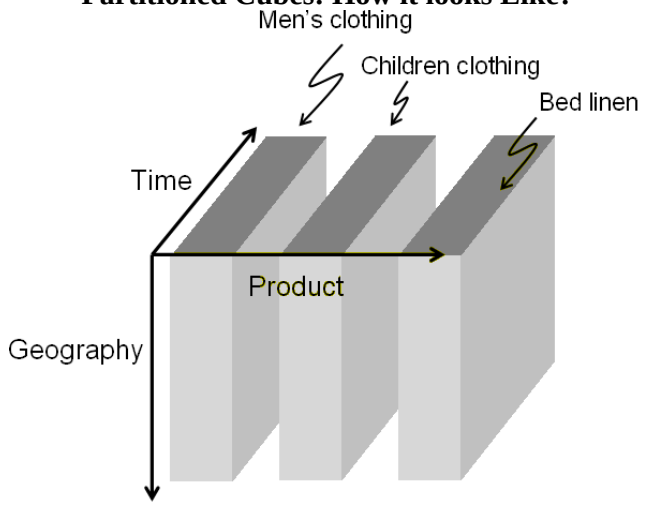
- To overcome the space limitation of MOLAP, the cube is partitioned.

- One logical cube of data can be spread across multiple physical cubes on separate (or same) servers.
- The divide&conquer cube partitioning approach helps alleviate the scalability limitations of MOLAP implementation.
- Ideal cube partitioning is completely invisible to end users.
- Performance degradation does occur in case of a join across partitioned cubes.

When the cardinality of a dimension forces a cube to become larger than what can reasonably be supported in a single cube, it is common practice to partition the cube into multiple "sub-cube" instantiations. The sub-cubes are usually defined around logical partitions within the dimensional hierarchies. For example, if there are a large number of entries at the most detailed level in a product hierarchy, it may be advisable to create distinct cubes along product category boundaries to divide-and-conquer. Another example would be to take a geographical dimension and partition into cubes whereby each region has its own cube.

Advanced MOLAP implementations (e.g., Microsoft Analytic/OLAP Services) will automatically distribute queries across partitioned cubes (potentially on distinct server platforms) for parallel retrieval without exposing the end user to the underlying physical deployment of the partitioned cubes.

Partitioned Cubes: How it looks Like?



Sales data cube partitioned at a major cotton products sale outlet

Figure-11.4: Partitioned Cube

There is unlikely to be a relationship between the sales of men's and children's clothing, thus the sales cube is partitioned, as shown in figure-11.4. Usually when men buy, they buy for themselves in most cases, except of course when they buying a gift for someone. When women buy, they may also buy for the children, but not men. But then this may be incorrect for a certain culture, hence is not a rule of thumb.

Bed linen is something very different, hence the original single cube of all sales aggregates can be partitioned along these two logical boundaries into three cubes which require less storage space per cube. As MOLAP is intrinsically main memory resident, hence cube partitioning does not results in notable speed-up, but does results in memory requirements, if for most queries one or two cube partitions are required with infrequent joins across them.

11.7 Virtual Cubes

Used to query two dissimilar cubes by creating a third "virtual" cube by a join between two cubes.

- Logically similar to a relational view i.e. linking two (or more) cubes along common dimension(s).
- Biggest advantage is saving in space by eliminating storage of redundant information.

Example: Joining the store cube and the list price cube along the product dimension, to calculate the sale price without redundant storage of the sale price data.

Advanced MOLAP implementations will often facilitate the creation of virtual cubes by combining multiple cubes (at run-time) along common dimensions between cubes.

There is a run-time cost to "joining" across cubes at the time of query execution, but the savings in storage can be quite large by eliminating redundancy in the underlying data representation. Typically, the virtual cubes involve "joining" cubes along common dimensions whereby one cube has a (smaller) subset of the dimensions from the other (bigger) cube.

Example: Usually the sale price of different items varies based on the geography and the time of the year. Instead of storing this information using an additional dimension, the said price is calculated at run-time, this may be slow, but can result in tremendous saving in space, as all the items are not sold throughout the year.

Masters

Lecture 12: Relational OLAP (ROLAP)

Learning Goals

- Why ROLAP?
- How to create a "Cube" in ROLAP
- Problem with simple approach
- ROLAP Issues

12.1 Why ROLAP?

Issue of scalability i.e. curse of dimensionality for MOLAP

- Deployment of significantly large dimension tables as compared to MOLAP using secondary storage.
- Aggregate awareness allows using pre-built summary tables by some front-end tools.
- Star schema designs usually used to facilitate ROLAP querying (in next lecture).

The biggest problem with MOLAP is the requirement of large main memory as the cube size increases. There may be many reasons for the increase in cube size, such as increase in the number of dimensions, or increase in the cardinality of the dimensions, or increase in the amount of detail data or a combination of some or all these aspects. Thus there is an issue of scalability which limits its applications to large data sets.

Despite advances in MOLAP technologies, high-end OLAP implementations will normally require assistance from a relational database. Hence a ROLAP or Relational OLAP. ROLAP tools will query the relational database using SQL generated to conform to a framework using the facts and dimensions paradigm using the star schema.

The other approach is "aggregate awareness" i.e. the environment is smart enough to develop or compute higher level aggregates using lower level or more detailed aggregates.

12.2 ROLAP as a "Cube"

- OLAP data is stored in a relational database (e.g. a star schema)
- The fact table is a way of *visualizing as* an "un-rolled" cube.
- So where is the cube?
  - It's a matter of perception
  - Visualize the fact table as an elementary cube.

ROLAP as a "Cube"

Fact Table

Month	Product	Zone	Sale K Rs.
M1	P1	Z1	250
M2	P2	Z1	500

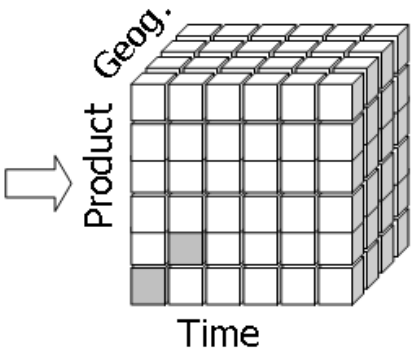


Figure-12.1: ROLAP as a cube

Recall from the last lecture that an implementation of the OLAP framework will be accepted as OLAP after it passes the FAS**M**I test i.e. it has to be **M**ulti-dimensional. The question is "how to convert something into a cube" which is intrinsically not a cube. And more importantly, if MOLAP has the space requirement limitations, and to overcome those limitations we are using a different implementation, then wouldn't the conversion back to a "cube" defeat the purpose? This is actually a very good question, and needs some detailed explanation. Fig-12.1 shows two rows of a fact table, and the aggregates corresponding to the cells of the fact table are shown correspondingly in a three dimensional cube. Extending this argument, each and every cell of the fact table can have a corresponding mapping. To clarify this further, consider the analogy of the map of the world. Our planet earth is NOT flat, it is spherical in nature i.e. three dimensional, yet the map of the earth is flat i.e. two dimensional in nature i.e. there is a mapping from 3D space to 2D space. If you look closely at the map, you will see that the map is divided into a grid based on longitude and latitude, and the corresponding cells or rectangles are not of the same size. Similarly a 2D table can be considered to be a mapping or representation of a multi-dimensional cube or vice-a-versa.

How to create a "Cube" in ROLAP

- **Cube is a logical entity containing values of a certain fact at a certain aggregation level at an intersection of a combination of dimensions.**
- **The following table can be created using 3 queries**

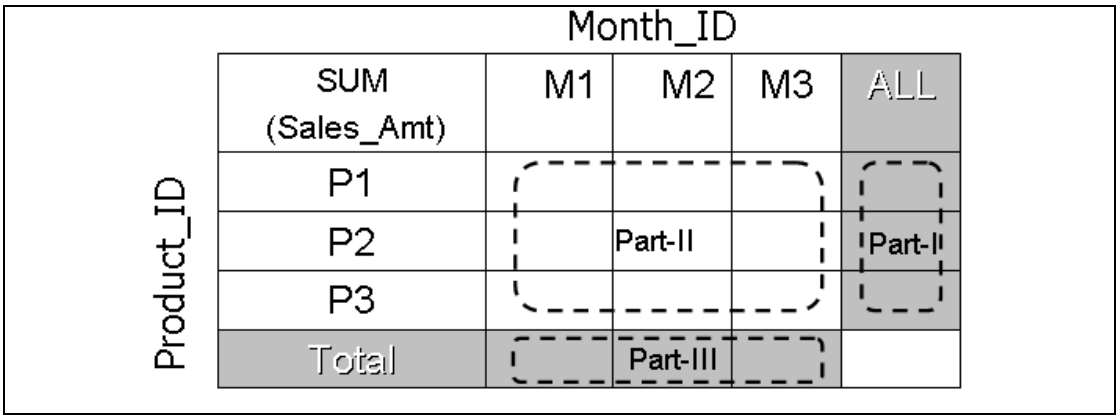


Figure-12.2: Creating tables from queries

When we talked of a cube for a MOLAP, it was not actually a physical cube, but was a logical entity. We continue with that concept, and assume that what was stored in a cube at a certain combination of indexes, corresponding to such a group of indices, we store the aggregates in a two dimensional table, and we use such groups of tables to store the same data that was stored in a MOLAP. The table shown in fig-12.2 is divided into three parts shown shaded and also shown by dotted lines. Corresponding to each part of the table, there is a query and consequently the table can actually be filled using three SQL queries as follows:

- **For the table entries, without the totals**

```
SELECT     S.Month_Id, S.Product_Id, SUM(S.Sales_Amt)
FROM        Sales
GROUP BY S.Month_Id, S.Product_Id;
```

- **For the row totals**

```
SELECT       S.Product_Id, SUM (Sales_Amt)
FROM        Sales
GROUP BY S.Product_Id;
```

- **For the column totals**

```
SELECT      S.Month_Id, SUM (Sales)
FROM        Sales
GROUP BY   S.Month_Id;
```

The first query can be used to fill Part-II of the table, the second query used to fill Part-I of the table, and the third query used to fill Part-III of the table, thus using these three queries, we create a ROALP structure.

Problem with simple approach

- Number of required queries increases exponentially with the increase in number of dimensions.
    - It's wasteful to compute all queries.
    - In the example, the first query can do most of the work of the other two queries
    - If we could save that result and aggregate over Month_Id and Product_Id, we could compute the other queries more efficiently

Using typical SQL to fill-up the tables quickly runs into a problem, as the number of dimensions increases, the number of aggregates also increases, and the number of queries required to calculate those aggregates also increases. Actually it becomes extremely wasteful to compute all queries, wasteful, because if we are smart, we can use the results of the queries already computed to get the answers to new queries. How to do this? it is not very difficult. For example for the column total queries, we could just add the aggregates over the results of the months. So the moral of the story is "Work smart not hard".

Cube clause

- The CUBE clause is part of SQL:1999
    - GROUP BY CUBE (v1, v2, …, vn)
    - Equivalent to a collection of GROUP BYs, one for each of the subsets of v1, v2, …, vn

The other problem with using standard SQL is that one has to write too many statements and that could lead to mistakes. Therefore, back in 1999 a CUBE clause was made part of SQL, and that clause is equivalent to a collection of GROUP BY clauses.

Some students who did a BS final year project with me of an HOLAP implementation, used dynamic web page generation to dynamically generate SQL instead of hard-coding the queries to generate the aggregates. Meaning, they used SQL to generate aggregates to fill a MOLAP cube. The project was a success, all got jobs based on this work; the first prize in the 13th Annual National Software Competition along with a cash prize of Rs. 30,000 was a bonus.

12.3 ROLAP and Space Requirement

If one is not careful, with the increase in number of dimensions, the number of summary tables gets very large

Consider the example discussed earlier with the following two dimensions on the fact table...

Time: Day, Week, Month, Quarter, Year, All Days

Product: Item, Sub-Category, Category, All Products

OK so we worked smart and got around the problem of aggregate generation. But the aggregates once generated have to be stored somewhere too i.e. in tables as this is a ROLAP environment.

Be warned: Pre-aggregates can very quickly get out of control in a ROLAP environment. Do not try to pre-aggregate all combinations of all dimensions at all levels of the hierarchies. If you do, the storage and maintenance costs will quickly overwhelm your implementation.

For example, consider the combinatorial explosion with just two dimensions as shown above...

EXAMPLE: ROLAP & Space Requirement

A naïve implementation will require all combinations of summary tables at each and every aggregation level.

❷	2001				2002			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Orange juice	232	2,432	4,353	354	535	345	7,897	789
Rola-Kola	2,342	243	353	4,535	5,655	4,424	789	798
8-UP	2,424	3,131	1,313	5,675	567	5,675	789	9,797
Pola-Kola	242	3,112	567	646	567	567	789	798
Mango juice	2,342	243	243	4,564	564	1,232	242	4,553
Bubbly-UP	3,453	3,453	535	2,422	2,131	242	1,321	245
Apple juice	253	456	2,433	567	2,442	5,453	4,566	345

❸	2001				2002			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Soda Drinks	8,461	9,939	2,768	13,278	8,920	10,908	3,688	11,638
Juices	2,827	3,131	7,029	5,485	3,541	7,030	12,705	5,687

❹	2001	2002
Orange juice	7,371	9,566
Mango juice	7,392	6,591
Apple juice	3,709	12,806
Rola-Kola	7,473	11,666
8-UP	12,543	16,828
Bubbly-UP	9,863	3,939
Pola-Kola	4,567	2,721

❿	2001	2002
Juices	18,472	28,963
Soda Drinks	36,447	37,156

24 summary tables, add in geography, results in 120 tables

Figure-12.3: Sample ROALP tables

There are 24 possible pre-aggregates just with the product and Time dimensions! Add in the geography dimension and we are quickly up to 120 pre-aggregates (and the number of levels in the hierarchies that we are assuming is very conservative). The largest aggregate will be the summarization by day and product because this is the most detailed. Clearly, we do not want to pre-aggregate all of these combinations.

Smart tools will allow less detailed aggregates to be constructed from more detailed aggregates (full aggregate awareness) at run-time so that we do not go all the way down to the detail for every aggregation. However, for this to work, the metrics must be additive (e.g., no ratios, averages, etc.). More detailed pre-aggregates are larger, but can also be used to build less detailed aggregates on-the-go.

12.4 ROLAP Issues

- Maintenance.
- Non standard hierarchy of dimensions.
- Non standard conventions.
- Explosion of storage space requirement.
- Aggregation pit-falls.

Creating an aggregate is relatively easy as compared to keeping it updated, the maintenance issue will eat you alive, if you (say) you get data late, and it is also to be reflected in the aggregate archives.
Dimensions are not just simply atomic item, then SKU, then product category. They can cross boundaries too and can become expensive to create.

The conventions for (say) week may be absolute within an organization, but differ across organizations, worst they could differ across the organization too e.g. marketing people looking differently at the week as compared to the accounts people.

ROLAP Issue: Maintenance

Summary tables are mostly a maintenance issue (similar to MOLAP) than a storage issue.

- Notice that summary tables get much smaller as dimensions get less detailed (e.g., year vs. day).

- Should plan for twice the size of the un-summarized data for ROLAP summaries in most environments.

Assuming "to-date" summaries, every detail record that is received into warehouse must aggregate into EVERY summary table.

It is not unusual for the collection of pre-aggregate summary tables to take up significantly more space (e.g., double or higher) than the raw data tables in a deployment that is focused heavily on query performance delivery. However, the maintenance cost for keeping all of the pre-aggregates up-to-date with the detailed data (very important for consistent query results) is very high. Pre-aggregate summary tables in the relational environment have the same “update problem” as cubes in a MOLAP environment.

ROLAP Issue: Hierarchies

Dimensions are NOT always simple hierarchies

Dimensions can be more than simple hierarchies i.e. item, subcategory, category, etc.

The product dimension might also branch off by trade style that cross simple hierarchy boundaries such as:

Looking at sales of air conditioners that cross manufacturer boundaries, such as COY1, COY2, COY3 etc.

Looking at sales of all "green colored" items that even cross product categories (washing machine, refrigerator, split-AC, etc.)

Looking at a combination of both.
It is quite common that dimensions are more than simple hierarchies. A simple product hierarchy might include item, subcategory, category, etc. However, the product dimension might also branch off by trade style in ways that cross boundaries of the simple hierarchy. We may want to look at sales of air conditioners that cross manufacturer boundaries, such as COY1, COY2, COY3 etc. Looking at all "green colored" items will cross product categories (washing machine, refrigerator, split-AC, etc.). Looking at a combination of both, or other will result in a combinatorial explosion, as these combinations get very large - such that brute-force pre-aggregation is not practical.

Providing analytic capability along multiple calendar hierarchies is even more unmanageable in terms of pre-aggregate construction.

ROLAP Issue: Semantics

Conventions are NOT absolute

Example: What is calendar year? What is a week?

- Calendar:

01 Jan. to 31 Dec or
01 Jul. to 30 Jun. or
01 Sep to 30 Aug.

- Week:

Mon. to Sat. or Thu. to Wed.

Conventions may vary across organizations, and even within an organization. Consider the apparently simple case of calendar year. There is no absolute standard definition of calendar year i.e. is it Jan to Dec. or Jun to Jul or any other time period covering 12 months? Similarly what is meant by week i.e. it starts from which day and ends at which day? Even within an organization the nomenclature may vary, for example finance people may consider a Mon. to Sun. week, while the marketing people may consider a Wed. to Tue. Week. Consequently the results of aggregation will vary across the organization for the same data, thus creating lot of confusion. To bring all the departments on the same grid may require lot of arm twisting from the highest level and will involve office politics too.

ROLAP Issue: Storage Space Explosion

Summary tables required for non-standard grouping

Summary tables required along different definitions of year, week etc.

Brute force approach would quickly overwhelm the system storage capacity due to a combinatorial explosion.

We just saw the problem of semantics and thought it was complex, we were wrong! There are many more and other non-standard grouping too. For example during the month of August to capitalize on the patriotism because of the Independence Day, different manufacturers print the national flag on the packages. This adds a new grouping, as the decision maker would be interested in knowing the sales of item with flag as compared to items without a flag printed on them.

In some cases bringing all the departments on the same grid for agreeing on the same definition of year or week may not be advisable, in such a case the number of summary tables increases, as tables are required for each definition of the week and the year. One may naively follow the brute force path i.e. creating all possible summary tables with all possible definitions, groupings and nomenclatures. This may work for small databases, but for VLDB (Very Large Data Bases) very soon the memory requirements of the system will be chocked. Thus this is not a viable option for large data sets.

ROLAP Issue: Aggregation Pitfalls

- Coarser granularity correspondingly decreases potential cardinality.
- Aggregating whatever that can be aggregated.
- Throwing away the detail data after aggregation.

As data is aggregated to higher levels in the dimensional hierarchy the number of rows retained will obviously reduce. However, this reduction is not usually directly proportional to the ratio of entries at each level in the hierarchy. For example, one might (incorrectly) think that the weekly summary table will be about 14% or 1/7th of the daily summary table as there are seven days in a week. Therefore, keeping the data at a daily level of granularity should be seven times the size of the weekly level, right? Wrong. The reason being all products that sell in a week definitely do not sell every day of the week. As a result, the factor is 2× rather than 7×. For example rental of movies usually goes up during the weekends. Most heart patients suffer a stroke on Monday, so are most of stock sales.

A general rule of thumb is that aggregation from one level of detail to another should only be undertaken if there is a 10x (or more) reduction in table size from the more detailed level to the coarser level of aggregation. Of course, this is only a guideline and will be influenced by the frequency of query activity at each level of potential aggregation.

Keeping the detail is important because it is inevitable that advanced analysis will require drill down to the most granular level of detail available. Remember Murphy's Law, the day you throw away the detail, is the day it would be required.

12.5 How to reduce summary tables?

Many ROLAP products have developed means to reduce the number of summary tables by:

- Building summaries on-the-fly as required by end-user applications.
- Enhancing performance on common queries at coarser granularities.
- Providing smart tools to assist DBAs in selecting the "best" aggregations to build i.e. trade-off between speed and space.

Tools from vendors such as Microsoft and Micro Strategy are "fully aggregate aware" with the ability to take summaries from one (more detailed) level of aggregation and roll them up into a less detailed summary at run-time. In this way, aggregate metrics can be delivered without forcing run-time aggregation from the most detailed data in the warehouse or mart.

Wizards have come into the marketplace with the ability to make suggestions as to optimal aggregates that should be built for maximizing performance for a defined workload.

12.6 Performance vs. Space Trade-off

- Maximum performance boost implies using lots of disk space for storing every pre-calculation.
- Minimum performance boost implies no disk space with zero pre-calculation.
- Using Meta data to determine best level of pre-aggregation from which all other aggregates can be computed.

Theoretically there can be two extremes i.e. free space and free performance. If storage is not an issue, then just pre-compute every cube at every unique combination of dimensions at every level as it does not cost anything. This will result in maximum query performance. But in reality, this implies huge cost in disk space and the time for constructing the pre-aggregates. In the other case where performance is free i.e. infinitely fast machines and infinite number of them, then there is not need to build any summaries. Meaning zero cube space and zero pre-calculations, and in reality this would result in minimum performance boost, in the presence of infinite performance.

What is meant by Meta data? Meta data is data about data. So what is the data about data one would be interested in? For example, how many entries are there at each level of the hierarchy? How many stores are there? How many stores per zone? How many zones per district? Etc. When the density of each dimension is known, it gives a fair idea where the aggregation is going to have the biggest bang for the buck. Because if I have a dimension that has for example on average two UPCs (Universal Product Code) per SKU (Stock Keeping Unit), its really not very interesting to build a summary of UPCs in SKUs because all it saves is adding two records together.

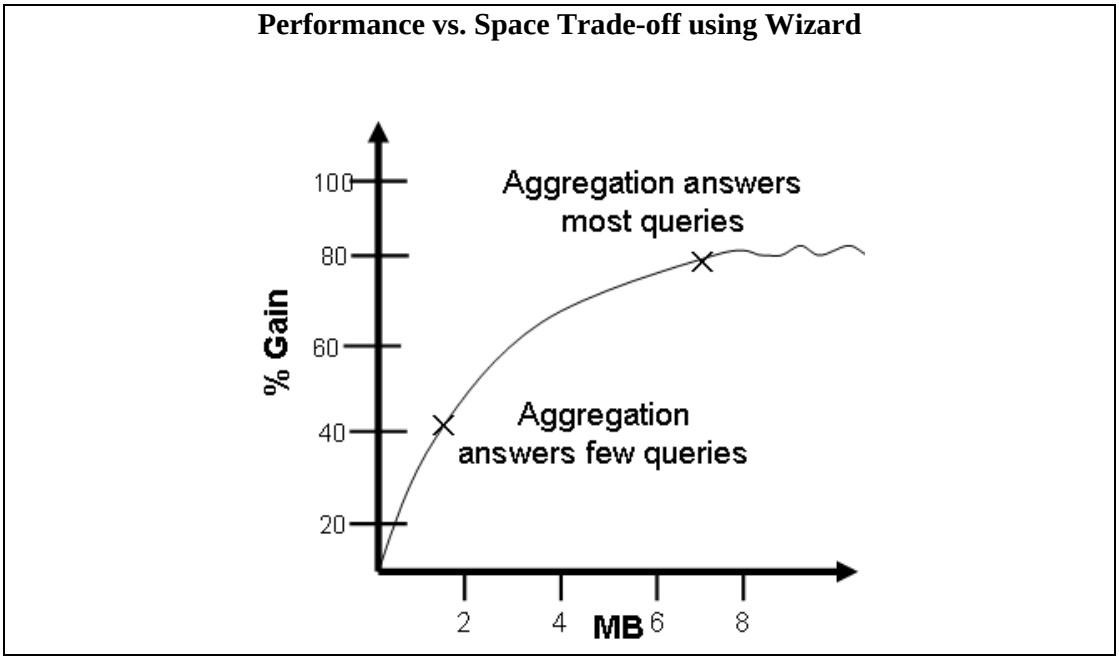


Figure-12.4: Aggregation vs. Performance

Aggregation design wizards allow a cube or pre-aggregate designer to specify the tradeoff between disk storage and performance to determine the maximum volume of pre-calculated aggregates, as shown in Figure-12.4. Suggestions about amount of aggregation are based on the amount of data reduction along different dimensions and the size of each aggregate.

Optimizing the amount of aggregation is a very hard problem, thus heuristics are used. Heuristics in the aggregate design wizard can be improved by Usage-Based Optimization Wizards via examination of query logs to determine which pre-aggregate will deliver best bang for your storage buck.

Microsoft's Usage-Based Optimization Wizard (shown here) also allows the DBA to tell Analytic (OLAP) Services to create a new set of aggregations for all queries exceeding a defined response time threshold.

12.7 HOLAP

- Target is to get the best of both worlds.
- HOLAP (Hybrid OLAP) allow co-existence of pre-built MOLAP cubes alongside relational OLAP or ROLAP structures.
- How much to pre-build?

The hybrid OLAP (HOLAP) solution is a mix of MOLAP and relational ROLAP architectures that supports queries against summary and transaction data in an integrated

fashion. HOLAP environments use MOLAP cubes to support common access paths with reasonably small dimensional cardinality and number of dimensions and relational structures when greater scalability for OLAP queries is required. This coexistence strategy allows exploiting the best of both worlds. Microsoft OLAP Services supports a HOLAP environment, as do tools such as HOLOS. The HOLAP approach enables a user to perform multidimensional analysis on data in the MDDB along with query based probing. However, if the user reaches the bottom of the multidimensional hierarchy and requires further detail data, the smart HOLAP engine automatically generates SQL to retrieve the detail data from the source RDBMS and returns it to the end user. This is done transparently to the user. Several MOLAP vendors, such as Arbor and Oracle, have transitioned to HOLAP architectures that include a ROLAP component. However, these HOLAP architectures are typically more complex to implement and administer than ROLAP or MOLAP architectures individually.

12.9 DOLAP

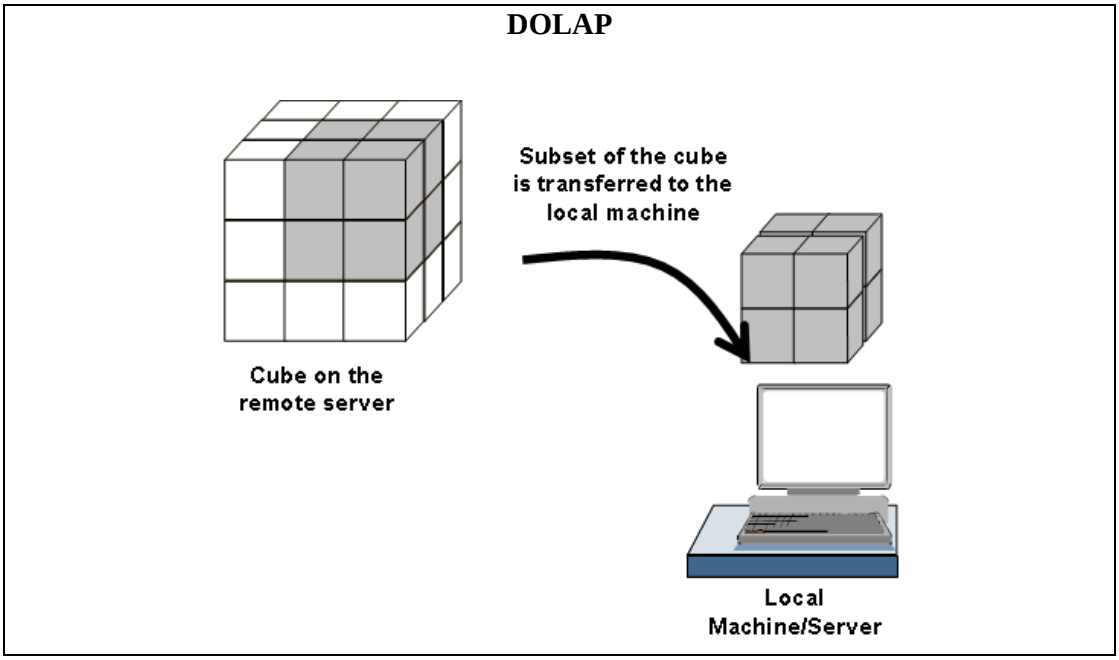


Figure-12.5: Concept of Desktop OLAP or DOLAP

DOLAP typically is the simplified version of MOLAP or ROLAP. DOLAP is inexpensive, it is fast and easy to setup on small data sets comprising of thousands of rows instead of millions of rows. It provides specific cube for the analysis. The DOLAP systems developed are extensions of production system report writers, while the systems developed in the early days of client /server computing aimed to take advantage of the power of the emerging PC desktop machine. DOLAP also provides the mobile operations of OLAP for the people who travel and move extensively, such as sales people. The one obvious disadvantage of DOLAP is that it lacks the ability to manage large data sets. But this is just another technique to suit the business requirement.

Masters
Subscribe to masters

Masters

Lecture 13: Dimensional Modeling (DM)

Dimensional modeling is a data modeling technique where you break data up into "facts" and "dimensions" to organize and describe entities within your data warehouse.

Learning Goals

- The need for ER modeling?
- Why ER Modeling has been so successful?
- Need for DM: Un-answered Qs
- Need for DM: The Paradox
- ER vs. DM
- What is DM?
- Features of Schemas

Relational modeling techniques are used to develop OLTP systems. A typical OLTP system involves a very large number of focused queries i.e. accesses a small number of database rows, which are accessed mostly via
unique or primary keys using indexes. The relational model, with well thought-out de-normalization, can result in a physical database design that meets the requirements of OLTP systems. Decision support systems, however, involve fundamentally different queries. These systems involve fewer queries that are usually not focused i.e. access large percentages of the database tables, often performing joins on multiple tables and in many cases aggregating and sorting the results. OLTP systems are NOT good at performing joins.

In the early days of DSS, OLTP systems were used for decision support with very large databases and performance problems were encountered. Some surprising findings were the deterioration of the performance of the DBMS optimizers when the number of tables joined exceeded a certain threshold. This problem led to development of a second type of model that "flattened out" the relational model structures by creating consolidated reference tables from several pre-joined existing tables i.e. converting snow-flakes into stars.

However, according to one school of thought, DM should not be the first reaction, but followed after trying indexing, and views and weighing different options carefully.

13.1 The need for ER modeling?

- Problems with early COBOLian data processing systems.
- Data redundancies.
- From flat file to *Table*, each entity ultimately becomes a *Table* in the physical schema.
- Simple $O(n^2)$ Join to work with *Tables.*

ER is a logical design technique that seeks to remove the redundancy in data. Imagine the WAPDA line-man going from home to home and recording the meter reading from each customer. In the early COBOLian days of computing (i.e. long before relational databases were invented), this data was transferred into the computer from the original hand

recordings as a single file consisting of number of fields, such as name of customer, address, meter number, date, time, current reading etc. Such a record could easily have been 500 bytes distributed across 10 fields. Other than the recording date and meter reading, EVERY other field was getting repeated. Although having all of this redundant data in the computer was very useful, but had its down sides too, from the aspect of storing and manipulating data. For example, data in this form was difficult to keep consistent because each record stood on its own. The customer's name and address appeared many times, because this data was repeated whenever a new reading was taken. Inconsistencies in the data went unchecked, because all of the instances of the customer address were independent, and updating the customer's address was a chaotic transaction as there are literally a dozen ways to just represent "house no"!

Therefore, in the early days of COBOLian computing, it became evident (I guess first to Dr. Codd) to separate out the redundant (or repeating) data into distinct tables, such as a customer master table etc., but at a price. The then available systems for data retrieval and manipulation became overly complex and inefficient because, this required careful attention to the processing algorithms for linking these sets of tables together i.e. join operation. Note that a typical nested loop join will run in $O(n^2)$ time if the two tables are separate such as customer and product bought by the customers. This requirement resulted in database system that was very good at linking tables, and paved the way for the relational database revolution.

13.2 Why ER Modeling has been so successful?

- Coupled with normalization drives out all the redundancy from the database.
- Change (or add or delete) the data at just one point.
- Can be used with indexing for very fast access.
- Resulted in success of OLTP systems.

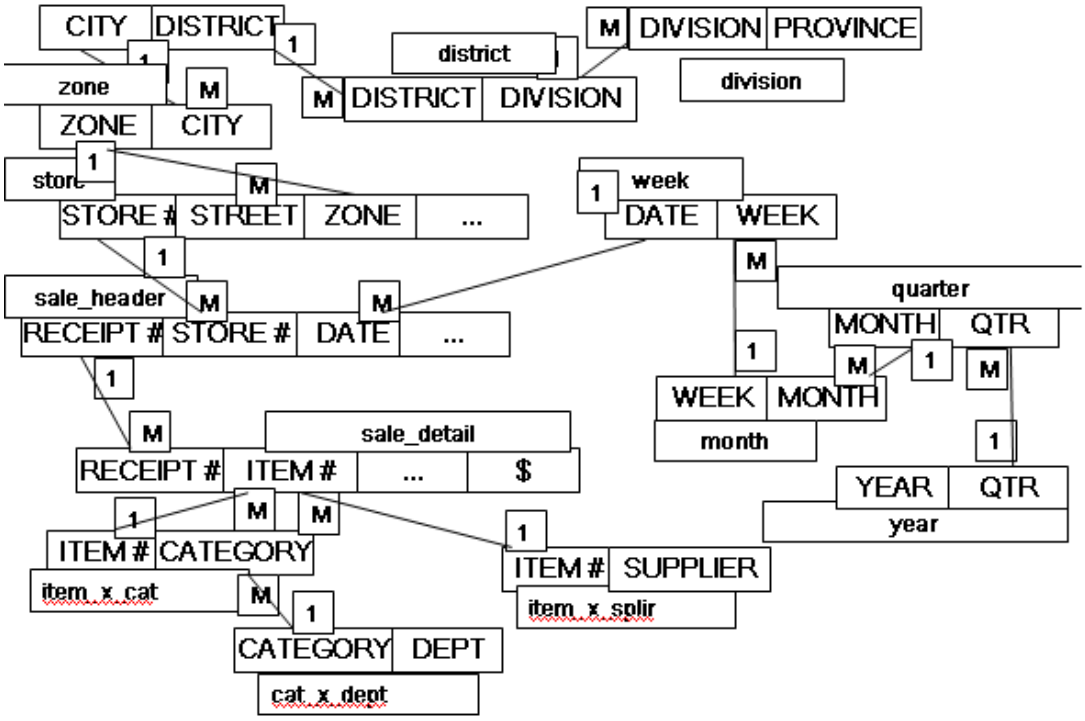
The ER modeling technique is a discipline used to highlight the microscopic relationships among data elements or entities. The pinnacle of achievement of ER modeling is to remove all redundancy from the data. This is enormously beneficial for the OLTP systems, because transactions are made very simple and deterministic i.e. no surprises in the overall query execution time i.e. it must finish (say) under 5 seconds. The transaction for updating customer's address gets reduced to a single record lookup in a customer address master table. This lookup is controlled by a customer address key, which defines uniqueness of the customer address record i.e. PK. This lookup is implemented using an index, hence is extremely fast. It can be stated without hesitation, that the ER techniques have contributed to the success of the OLTP systems.

13.3 Need for DM: Un-answered Qs

- Lets have a look at a typical ER data model first.
- Some Observations:
  - All tables look-alike, as a consequence it is difficult to identify:

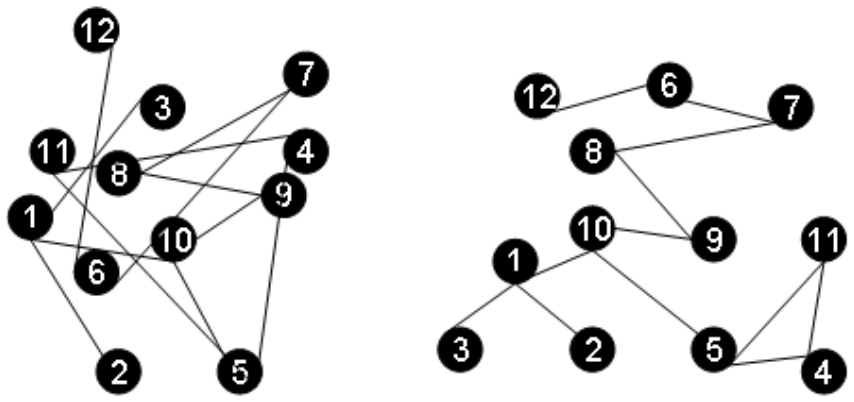
- Which table is more important ?
- Which is the largest?
- Which tables contain numerical measurements of the business?
- Which table contain nearly static descriptive attributes?

Is DM really needed? In order to better understand the need for DM lets have a look at the diagram showing the retail data in simplified 3NF.



Having a close look at the diagram, it reveals that a separate table has been maintained to store different entities resulting in a complex overall model. Probing the model to get informative insight about data is not straightforward. For example, we can not tell by looking at the model the relative importance of tables. We can not know the table sizes (though we can guess by looking at just headers), and most importantly we can not tell about business dimensions by just looking at the table headers i.e. which tables contain business measurements of the business , which tables contain the static descriptive data and so on.

- **Many topologies for the same ER diagram, all appearing different.**
  - **Very hard to visualize and remember.**



- **A large number of possible connections to any two (or more) tables**

Figure-13.1: Need for DM: Un-answered Qs

Even for the simple retail example, there are more than a dozen tables that are linked together by puzzling spaghetti of links; and this is just the beginning. This problem is further complicated by the fact that there can be numerous (actually exponentially large) topologies for the same system created by different people, and re-orienting makes them better (or worse).

The problem becomes even more complex for the ER model for an enterprise, which has hundreds of logical entities, and for a high end ERP systems for a large multinational there could be literally thousands of such entities. Each of these entities typically translates into a physical table when the database is implemented. Making sense of this "mess" is almost impossible, communicating it to someone even more difficult.

13.4 Need for DM: The Paradox

- **The Paradox:** Trying to make information accessible using tables resulted in an inability to query it efficiently!

- ER and Normalization result in large number of tables which are:
  - Hard to understand by the users (DB programmers)

  - Hard to navigate optimally by DBMS software

- Real value of ER is in using tables individually or in pairs

- Too complex for queries that span multiple tables with a large number of records

However, there is a paradox! In the fervor to make OLTP systems fast and efficient, the designers lost sight of the original, most important goal i.e. querying the databases to retrieve data/information. Thus this defeats the purpose as follows:

- Due to the sheer complexity of the ER graph, end users cannot understand or remember an ER model, and as a consequence cannot navigate an ER model. There is no graphical user interface (GUI) that takes a general ER model and makes it usable to end users, because this tends to be an NP-Complete problem.

- ER models are typically chaotic and "random", hence software cannot usefully query them. Cost-based optimizers that attempt to do this are infamous for making the wrong choices, and disastrous performance consequences.

- And finally, use of the ER modeling technique defeats the basic attraction of data warehousing, namely intuitive and high-performance retrieval of data.

ER vs. DM

ER	DM
Constituted to optimize OLTP performance.	Constituted to optimize DSS query performance.
Models the micro relationships among data elements.	Models the macro relationships among data elements with an overall deterministic strategy.
A wild variability of the structure of ER models.	All dimensions serve as equal entry points to the fact table.
Very vulnerable to changes in the user's querying habits, because such schemas are asymmetrical.	Changes in user querying habits can be catered by automatic SQL generators.

Table 13.1: ER vs. DM

1- ER models are constituted to (a) remove redundancy from the data model, (b) facilitate retrieval of individual records having certain critical identifiers, and (c) therefore, optimize On-line Transaction Processing (OLTP) performance.

2- ER modeling does not really model a business; rather, it models the micro relationships among data elements. ER modeling does not have "business rules," it has "data rules."

3- The wild variability of the structure of ER models means that each data warehouse needs custom, handwritten and tuned SQL. It also means that each schema, once it is tuned, is very vulnerable to changes in the user's querying habits, because such schemas are asymmetrical.

1-In DM, a model of tables and relations is constituted with the purpose of optimizing decision support query performance in relational databases, relative to a measurement or set of measurements of the outcome(s) of the business process being modeled.

2-Even a big suite of dimensional models has an overall deterministic strategy for evaluating every possible query, even those crossing many fact tables.

3-All dimensions serve as equal entry points into the fact table. Changes in users' querying habits don't change the structure of the SQL or the standard ways of measuring and controlling performance.

13.5 How to simplify an ER data model?

- Two general methods:
    - De-Normalization
    - Dimensional Modeling (DM)

The shortcomings of ER modeling did not unnoticed. Since the beginning of the relational database revolution, many DB designers tried to deliver the design data to end users as rather look-alike "simpler designs with a "dimensional" i.e. ease of understanding and performance as the highest goals. There are actually two ways of “simplifying” the ER model i.e. (i) De-normalization and (ii) Dimensional Modeling.

13.6 What is DM?

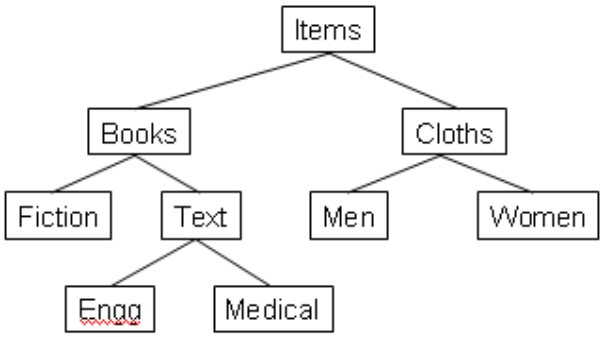
- A simpler logical model optimized for decision support.
- Inherently dimensional in nature, with a single central fact table and a set of smaller dimensional tables.
- Multi-part key for the fact table
- Dimensional tables with a single-part PK.
- Keys are usually system generated.
-
- Results in a star like structure, called star schema or star join.
    - All relationships mandatory M-1.
    - Single path between any two levels.
    - Supports ROLAP operations.

DM is a logical design technique that seeks to present the data in a standard, instinctive structure that supports high-performance and ease of understanding. It is inherently dimensional in nature, and it does adhere to the relational model, but with some important restrictions. Such as, every dimensional model is composed of one “central” table with a

multipart key, called the fact table, and a set of smaller tables called dimension tables. Each dimension table has a single-part primary key that corresponds exactly to one of the components of the multipart key in the fact table. This results in a characteristic "star-like" structure or star schema.

DM is a logical design technique that seeks to present the data in a standard, instinctive structure that supports high-performance and ease of understanding. It is inherently dimensional in nature, and it does adhere to the relational model, but with some important restrictions. Such as, every dimensional model is composed of one "central" table with a multipart key, called the fact table, and a set of smaller tables called dimension tables. Each dimension table has a single-part primary key that corresponds exactly to one of the components of the multipart key in the fact table. This results in a characteristic "star-like" structure or star schema.

Dimensions have Hierarchies



Analysts tend to look at the data through dimension at a particular "level" in the hierarchy

Figure-13.2: Dimensions have Hierarchies

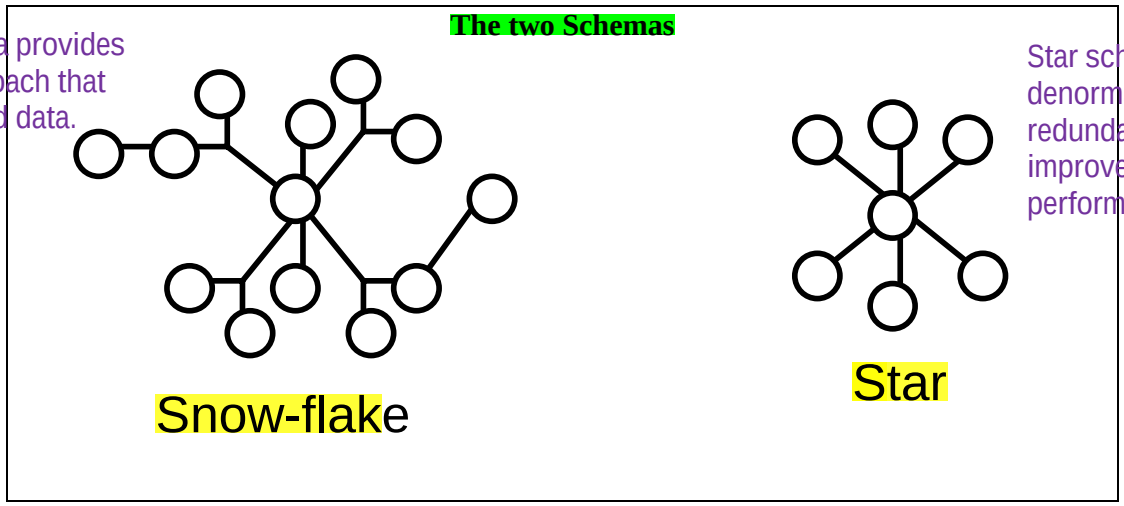
The foundation for design in this environment is through use of dimensional modeling techniques which focus on the concepts of "facts" and "dimensions" for organizing data.

Facts are the quantities or numerical measures (e.g., sales $) that we can count and the most useful being those that are additive. The most useful facts in a fact table are numeric and additive. Additive nature of facts is important, because data warehouse applications almost never retrieve a single record form the fact table; instead, they fetch back hundreds, thousands, or even millions of these records at a time, and the only useful thing to do with so many records is to add them up. Example, **what is the average salary of customers who's age > 35 and experience more than 5 years?**

13.7 Schemas

Dimensions are the descriptive textual information and the source of interesting constraints on how we filter/report on the quantities (e.g., by geography, product, date, etc.). For the DM shown, we constrain on the clothing department via the Dept attribute in the Product table. It should be obvious that the power of the database shown is proportional to the quality and depth of the dimension tables.

Snowflake schema provides a bottom-up approach that utilizes normalized data.



Star schema utilizes denormalization and redundancy, which improves read performance.

Figure-13.3: The two schemas

Fig-13.3 shows the snow-flake schema i.e. with multiple hierarchies that is typical of an OLTP or MIS system. The other is a simplified star schema with no hierarchies and a central node. Such schemas are typical of Data Warehouses.

Snowflake Schema: Sometimes a pure star schema might suffer performance problems. This can occur when a de-normalized dimension table becomes very large and penalizes the star join operation. Conversely, sometimes a small outer-level dimension table does not incur a significant join cost because it can be permanently stored in a memory buffer. Furthermore, because a star structure exists at the center of a snowflake, an efficient star join can be used to satisfy part of a query. Finally, some queries will not access data from outer-level dimension tables. These queries effectively execute against a star schema that contains smaller dimension tables. Therefore, under some circumstances, a snowflake schema is more efficient than a star schema.

Star Schema: A star schema is generally considered to be the most efficient design for two reasons. First, a design with de-normalized tables encounters fewer join operations. Second, most optimizers are smart enough to recognize a star schema and generate access plans that use efficient "star join" operations. It has been established that a "standard template" data warehouse query directly maps to a star schema.

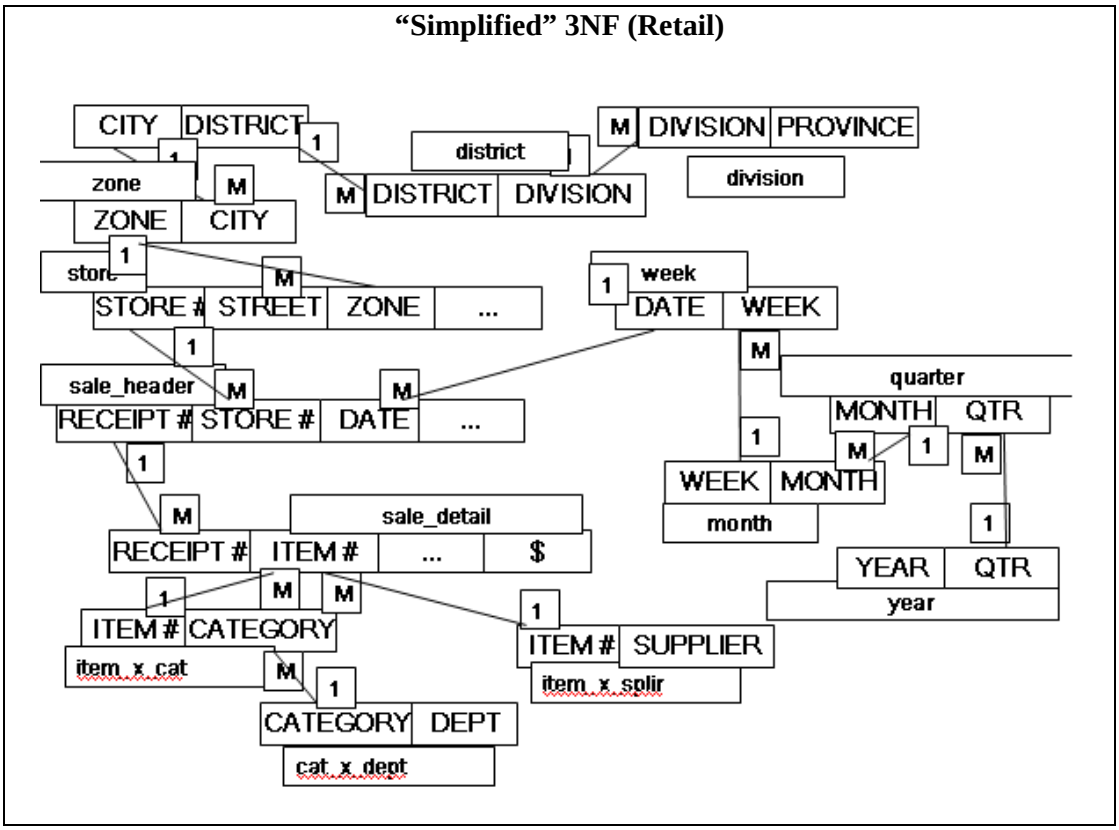


Figure-13.4: "Simplified" 3NF (Retail)

In Fig-13.4 a (simplified) retail data model is shown in the third normal form representation keeps each level of a dimensional hierarchy in a separate table (e.g., store, zone, region or item, category, department). The sale header and detail information is also maintained in two separate tables.

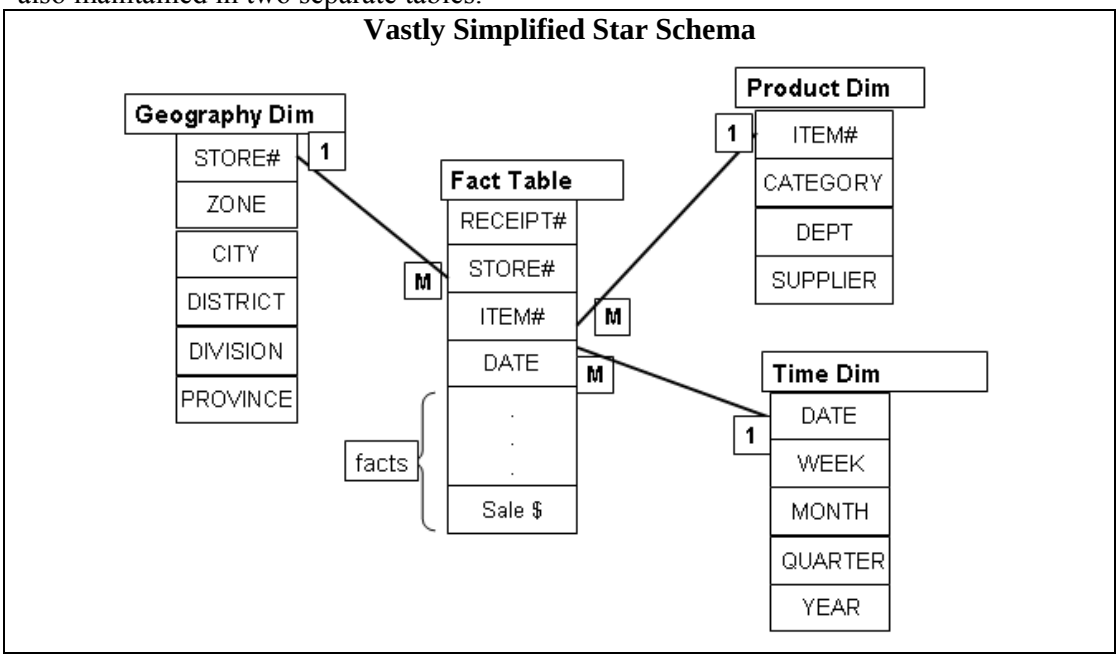


Figure-13.5: Vastly Simplified Star Schema

The goal of a star schema design is to simplify the physical data model so that RDBMS optimizers can exploit advanced indexing and join techniques in a straightforward manner, as shown in Fig-13.5. Some RDBMS products rely on star schemas for performance more than others (e.g., Red Brick versus Teradata).

The ultimate goal of a star schema design is to put into place a physical data model capable of very high performance to support iterative analysis adhering to an OLAP model of data delivery. Moreover, SQL generation is vastly simplified for front-end tools when the data is highly structured in this way.

In some cases, facts will also be summarized along common dimensions of analysis for additional performance.

The Benefit of Simplicity

Beauty lies in close correspondence with the business, evident even to business users.

The ultimate goal of a star schema design is to put into place a physical data model capable of very high performance to support iterative analysis adhering to an OLAP model of data delivery. Moreover, SQL generation is vastly simplified for front-end tools when the data is highly structured in this way.

In some cases, facts will also be summarized along common dimensions of analysis for additional performance.

Features of Star Schema

Dimensional hierarchies are collapsed into a single table for each dimension. Loss of information

A single fact table created with a single header from the detail records, resulting in:

- A vastly simplified physical data model!
- Fewer tables (thousands of tables in some ERP systems).
- Fewer joins resulting in high performance.
- Some requirement of additional space.

By "flattening" the information for each dimension into a single table and combining header/detail records, the physical data model is vastly simplified. The simplified data model of a star schema allows for straightforward SQL generation and makes it easier for RDBMS optimizers detect opportunities for "star joins" as a means of efficient query execution.

Notice that star schema design is merely a specific methodology for deploying the pre-join de-normalization that we discussed earlier.

13.8 Quantifying space requirement

Quantifying use of additional space using star schema

There are about 10 million mobile phone users in Pakistan.
Say the top company has half of them = 500,000

Number of days in 1 year = 365
Number of calls recorded each day = 250,000 (assumed)
Maximum number of records in fact table = 91 billion rows
Assuming a relatively small header size = 128 bytes
Fact table storage used = 11 Tera bytes
Average length of city name = 8 characters ≈ 8 bytes
Total number of cities with telephone access = 170 (1 byte)
Space used for city name in fact table using Star = 8 x 0.091 = 0.728 TB
Space used for city code using snow-flake = 1x 0.091= 0.091 TB
Additional space used ≈ 0.637 Tera byte i.e. about 5.8%

By virtue of flattening the dimensions, instead of storing the city code, now in the "flattened" table the name of the city will be stored. There is a 1: 8 ratio between the two-representations. But out of a header size of 128, there has been an addition of 7 more bytes i.e. an increase in storage space of about 5%. This is not much, if there are frequent queries for which the join has not been eliminated.

Masters

Lecture 14: Process of Dimensional Modeling

Learning Goals

- Business Process
- Grain
- Choose the Facts
- Choose the Dimensions

14.1 The Process of Dimensional Modeling

Four Step Method from ER to DM

1. Choose the Business Process
2. Choose the Grain
3. Choose the Facts
4. Choose the Dimensions

A typical ER diagram covers the entire spectrum of the business, actually covers every possible business process. However, in reality those multitudes of process do not co-exist in time and space (tables). As a consequence, an ER diagram is overly complex, and is a demerit to itself. This is precisely the point that differentiates a DM from an ER diagram, as a single ER diagram can be divided into multiple DM diagrams. Thus a step-wise approach is followed to separate the DMs from an ER diagram, and this consists of four steps.

Step-1: Separate the ER diagram into its discrete business processes and to model each business process separately.

Step-2: Grain of a fact table = the meaning of one fact table row. Determines the maximum level of detail of the warehouse.

Step-3: Select those many-to-many relationships in the ER model containing numeric and additive non-key items and designate them as fact tables. Actually all business events to be analyzed are gathered into fact tables.

Step-4: De-normalize all of the remaining tables into flat tables with single-part keys that connect directly to the fact tables. These tables become the dimension tables. They are like reference tables that define how to analyze the fact information. They are typically small and relatively static.

Let's discuss each of the steps in detail, one by one.

14.2 Step-1: Choose the Business Process

- A business process is a major operational process in an organization.
- Typically supported by a legacy system (database) or an OLTP.
  - Examples: Orders, Invoices, Inventory etc.

Business Processes are often termed as Data Marts and that is why many people criticize DM as being data mart oriented.

The first step in DM is the business process selection. What do we mean by a process? A process is a natural business activity in the organization supported by a legacy source data-collection system (database or OLTP). Example business processes include purchasing, orders, shipments, invoicing, inventory, and general ledger etc.

Many people consider business processes as Data marts i.e. in their view organizational or departmental function is referred as the business process. That is why such people criticize DM as being a data mart oriented approach. However, in Kimball's view, it is a wrong approach, the two must not be confused e.g. a single model is built to handle orders data rather than building separate models for marketing and sales departments, which both access the orders data. By focusing on business processes, rather than departments, consistent information can be delivered economically throughout the organization. Building departmental data models may result in data duplication, data inconsistencies, and data management issues. What is a possible solution? Yes, publishing data once can not only reduce the consistency problems, but can also reduce ETL development, data management and disk storage issues as well.

Step-1: Separating the Process

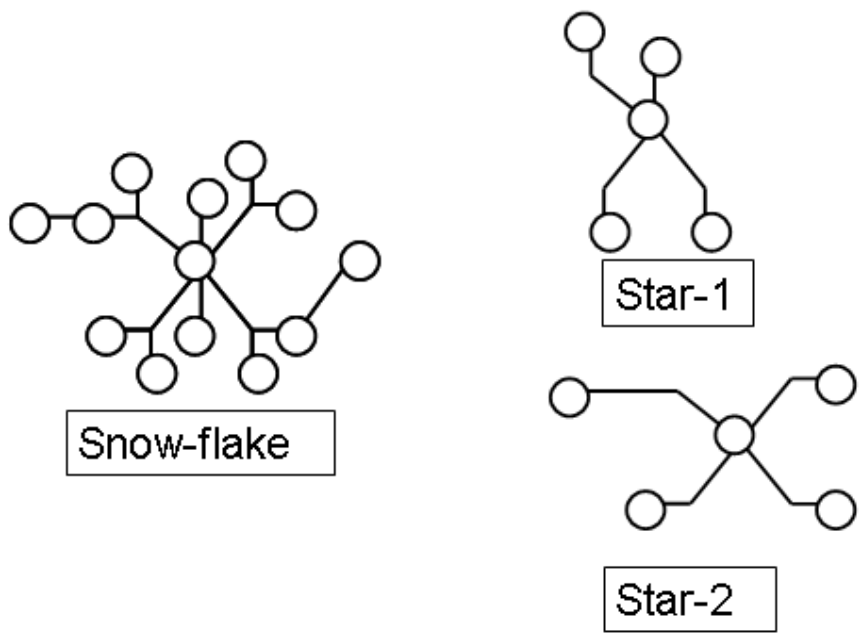


Figure-14.1: Step-1: Separating the Process

Fig-14.1 shows an interesting concept i.e. separating business processes to be modeled from a complex set of processes. This translates to splitting a snow-flake schema into multiple star schemas. Note that as the processes move into the star schema all the hierarchies collapse.

14.3 Step-2: Choosing the Grain

- Grain is the fundamental, atomic level of data to be represented.
- Grain is also termed as the unit of analyses.
- Example grain statements
- Typical grains
  - Individual Transactions
  - Daily aggregates (snapshots)
  - Monthly aggregates
- Relationship between grain and expressiveness.
- Grain vs. hardware trade-off.

Grain is the lowest level of detail or the atomic level of data stored in the warehouse. The lowest level of data in the warehouse may not be the lowest level of data recorded in the business system. It is also termed as the unit of analysis e.g. unit of weight is Kg etc.

Example grain statements: (*one fact row represents a…*)

- Entry from a cash register receipt
- Boarding pass to get on a flight
- Daily snapshot of inventory level for a product in a warehouse
- Sensor reading per minute for a sensor
- Student enrolled in a course

Finer-grained fact tables:

- are more expressive
- have more rows

Trade-off between performance and expressiveness

- Rule of thumb: Err in favor of expressiveness
- Pre-computed aggregates can solve performance problems

In the absence of aggregates, there is a potential to waste millions of dollars on hardware upgrades to solve performance problems that could have been otherwise addressed by aggregates.

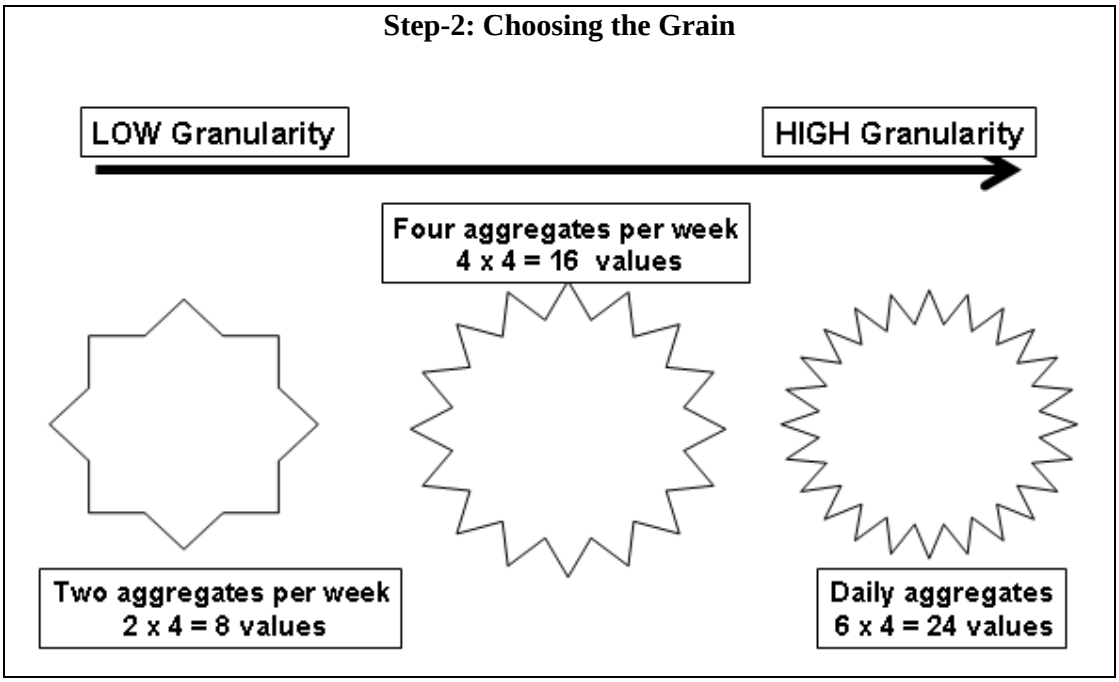


Figure-14.2: Step-2: Choosing the Grain

Note that you may come across definitions of grain as given in the notes and discussed in the lectures, but you may also come across definitions that are different from those discussed. This depends on the interpretation of the writer. We will follow the definition as per Fig-14.2.

14.4 The case FOR data aggregation

- Works well for repetitive queries.
- Follows the known thought process.
- Justifiable if used for max number of queries.
- Provides a "big picture" or macroscopic view.
- Application dependent, usually inflexible to business changes (remember lack of absoluteness of conventions).

There are both positives and negatives to data aggregation. These are a list of the reasons for the utilization of summary or aggregate data. As you can see, they all really fall under the area of "performance".

The negative side is that summary data does not allow a total solution with the flexibility and capabilities that some businesses truly require as compared to other businesses.

14.5 The case AGAINST data aggregation

- Aggregation is irreversible.

- Can create monthly sales data from weekly sales data, but the reverse is not possible.

- Aggregation limits the questions that can be answered.
  - What, when, why, where, what-else, what-next

- Aggregation can hide crucial facts.
  - The average of 100 & 100 is same as 150 & 50

Aggregation is one-way i.e. you can create aggregates, but can not dissolve aggregates to get the original data from which the aggregates were created. For example 3+2+1 = 6 at the same time 2+4 also equals 6, so does 5+1 and if we consider reals, then infinetly many ways of adding numbers to get the same result.

If you think about the "5 W's of journalism", these are the "6 W's of data analysis". Again it highlights the types of questions that end users want to ask and can not be answered by summary data.

By definition, a summarization will consider at least one of these points irrelevant. For example, a summary across the company takes out the dimension of "WHERE" and a summary by quarters takes out the element of "WHEN". The point to be noted is that although summary data has a purpose, yet one can take any summary and ask a question that the system can not answer.

Aggregation hides crucial facts

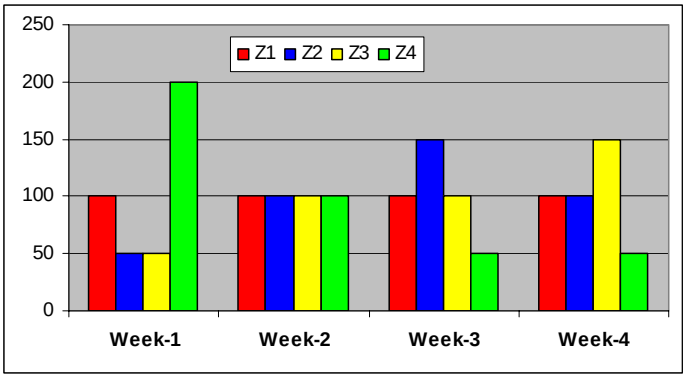
	Week-1	Week-2	Week-3	Week-4	Average
Zone-1	Just Looking at the averages i.e. aggregates				100
Zone-2					100
Zone-3					100
Zone-4					100
Average	100	100	100	100	

Table-14.1: Aggregation hides crucial facts

Consider the sales data of an item sold in a chain store in four zones, such that the sales data is aggregated across the weeks also. For this simple example, for the sake of conserving space the average sales across each zone and for each week is stored. Therefore, instead of storing 16 values only 8 values are stored i.e. a saving of 50% space.

Assume that a promotional scheme or advertisement campaign was run, and then the sales data was recorded to analyze the effectiveness of the campaign. If, we look at the averages (as shown in the table) there is no change in sales i.e. neither across time nor across the geography dimension. On the face of it, it was an in effective campaign. Now lets raise the curtain and look at the detailed sales records. The numbers are NOT constant! Drawing the graphs of the sales records shows a very different picture.

Aggregation hides crucial facts



Z1: Sale is constant (need to work on it)

Z2: Sale went up, then fell (need of concern)

Z3: Sale is on the rise, why?

Z4: Sale dropped sharply, need to look deeply.

W2: Static sale

Figure-14.3: Aggregation hides crucial facts

Z1: Sale is constant through out the month (need to work on it)

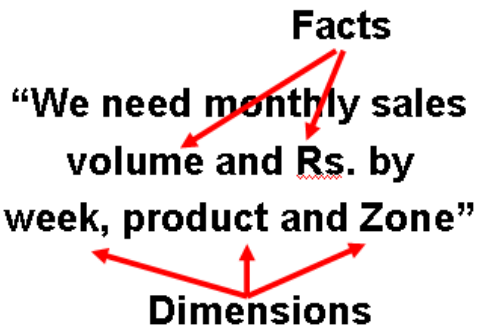
Z2: Sale went up, then fell (need of concern) i.e. the campaign was effective, but after week it fizzled down.

Z3: Sale is on the rise, why?

Z4: Sale dropped sharply, need to look deeply. It seems that the campaign had a negative effect on the sales?

W2: Static sale across all zones, very unique indeed.

Step 3: Choose Facts



Numeric facts are identified by answering the question "what are we measuring?" Many-to- many relationships in the ER model containing numeric and additive non-key items are selected and designated as fact tables. In the example numeric additive figures *volume* (quantity ordered) and *Rs*. (Rupees cost amount) are the facts because the numeric values of the two are of keen interest for the business user.

14.6 Step 3: Choose Facts

- Choose the facts that will populate each fact table record.
  - Remember that best Facts are Numeric, Continuously Valued and Additive.
  - Example: Quantity Sold, Amount etc.

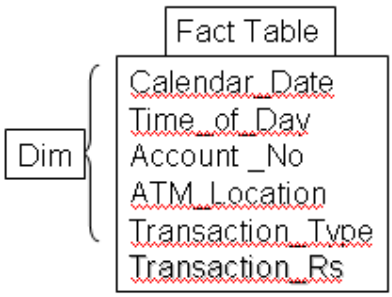
It should be remembered that facts are numeric, continuous, additive and non-key items that will populate the fact table. Example facts for a point of sales terminal (POS) are sales quantity, per unit sales price, and the sales amount in rupees. All the candidate facts in a design must be true to the grain described in previous slides. Facts that clearly belong to a different grain must be in a separate fact table.

14.7 Step 4: Choose Dimensions

- Choose the dimensions that apply to each fact in the fact table.
  - Typical dimensions: time, product, geography etc.
  - Identify the descriptive attributes that explain each dimension.
  - Determine hierarchies within each dimension.

Step-4: How to Identify a Dimension?

- **The single valued attributes during recording of a transaction are dimensions.**



Time_of_day: Morning, Mid Morning, Lunch Break etc.

Transaction_Type: Withdrawal, Deposit, Check balance etc.

Table-14.2: Step-4: How to Identify a Dimension?

The dimension tables usually represent textual attributes that are already known about things such as the product, the geography, or the time. If the database designer is very clear about the grain of the fact table, then choosing the appropriate dimensions for the fact table is usually easy. What is so special about it, seems to be pretty intuitive, but is not.

The success in selecting the right dimensions for a given fact table is dependent on correctly identifying any description that has a single value for an individual fact table record or transaction. Note that the fact table record considered could be a single transaction or weekly aggregate or monthly sums etc i.e. a grain is associated. Once this is correctly identified and settled, then as many dimensions can be added to the fact able as required. For the ATM customer transaction example, the following dimensions all have a single value during the recording of the transaction, as none of the above dimensions change during a single transaction:

- Calendar_Date
- Time_of_Day
- Account _No
- ATM_Location
- Transaction_Type (withdrawal, deposit, balance inquiry etc.)

Over here Time_of_Day refers to specific periods such as Morning, Mid Morning, Lunch Break, Office_Off etc. Note that during an atomic transaction, the value of Time_of_Day does not change (as a transaction takes less than a minute), hence it is a dimension. In the context of the ATM example, the only numeric attribute is the Transaction_Rs, so it is a fact. Observe that we use this convention in real life also, when people say we will visit you first time or second time of the day etc.

Step-4: Can Dimensions be Multi-valued?

- **Are dimensions ALWYS single?**
  - **Not really**
  - **What are the problems? And how to handle them**
    - Calendar_Date (of inspection)
    - Reg_No
    - Technician
    - Workshop
    - Maintenance_Operation
- **How many maintenance operations are possible?**
  - **Few**
  - **Maybe more for old cars**

Figure-14.4: Step-4: Can Dimensions be Multi-valued?

After convincing ourselves that dimensions are really single valued, perhaps we should consider whether there are ever legitimate exceptions i.e. is it possible to have multi-valued dimension in a fact table? If this is conceivable, what problems might arise?

Consider the following example from vehicle maintenance system used at a vehicle service center. You are handed a data sheet for which the grain is the individual line item on the customer's bill. The data source could be your periodic car maintenance visits to the company workshop or individual replacement charges on a repair/change bill. These individual line items have a rich set of dimensions such as:

- Calendar_Date (of inspection)
- Reg_No (of vehicle)
- Technician_ID
- Workshop
- Maintenance_Operation

The numeric additive facts in this design (which are the core of every fact table in a dimensional design) would include Amount_Charged and perhaps others including Amount_Paid, depending upon if the vehicle was insured etc.

On the face of it, this seems to be a very straightforward design, with obvious single values for all the dimensions. But there is a surprise. In many situations, there may be multiple values for services performed, such as oil change, air filter change, spark plug change etc. What do you do if for a certain car there are three separate changes at the moment the service was performed? How about really old and ill-kept cars that might have upto 5+ such changes? How do you encode the Maintenance_Operation dimension if you wish to represent this information?

Step-4: Dimensions & Grain

- Several grains are possible as per business requirement.
    - For some aggregations certain descriptions do not remain atomic.
    - Example: Time_of_Day may change several times during daily aggregate, but not during a transaction.
    - Choose the dimensions that are applicable within the selected grain.

Strangely, there is a relationship between the grain and the dimensions. When building a fact table, the most important step is to declare the grain (aggregation level) of the fact table. The grain declares the exact meaning of an individual fact record. Consider the case of transactions for an ATM machine. The grain could be individual customer transaction, or number of transaction per week or the amount drawn per month.

- Calendar_Date
- Time_of_Day
- Account _No
- ATM_Location
- Transaction_Type (withdrawal, deposit, balance inquiry etc.)

Note that none of the above dimensions change during a single transaction. However, for weekly transactions probably only Account _No and ATM_Location can be treated as a dimension.

Note that higher the level of aggregation of the fact table, the fewer will be the number of dimensions you can attach to the fact records. The converse of this is surprising. The more granular the data, the more dimensions make sense. Hence the lowest-level data in any organization is the most dimensional.

Masters

Lecture 15: Issues of Dimensional Modeling

Learning Goals

- Additive vs. Non-Additive facts
- Classification of Aggregation Functions
- Not recording Facts
- A Fact-less Fact Table
- OLTP & Slowly Changing Dimensions

Step 3: Additive vs. Non-Additive facts

- **Additive facts are easy to work with**
  - **Summing the fact value gives meaningful results**
  - **Additive facts:**
    - **Quantity sold**
    - **Total Rs. sales**
  - **Non-additive facts:**
    - **Averages (average sales price, unit price)**
    - **Percentages (% discount)**
    - **Ratios (gross margin)**
    - **Count of distinct products sold**

Month	Crates of Bottles Sold
May	14
Jun.	20
Jul.	24
TOTAL	58

Month	% discount
May	10
Jun.	8
Jul.	6
TOTAL	24% ← Incorrect!

There can be two types of facts i.e. additive and non-additive. Additive facts are those facts which give the correct result by an addition operation. Examples of such facts could be number of items sold, sales amount etc. Non-additive facts can also be added, but the addition gives incorrect results. Some examples of non-additive facts are average, discount, ratios etc. Consider three instances of 5, with the sum being 15 and average being 5. Now consider two numbers i.e. 5 and 10, the sum being 15, but the average being 7.5. Now, if the average of 5 and 7.5 is taken this comes to be 6.25, but if the average of the actual numbers is taken, the sum comes to be 30 and the average being 6. Hence averages, if added gives wrong results. Now facts could be averages, such as average sales per week etc, thus they are perfectly legitimate facts.

15.1 Step-3: Classification of Aggregation Functions

- How hard to compute aggregate from sub-aggregates?

- Three classes of aggregates:
  - Distributive
    - Compute aggregate directly from sub-aggregates
    - Examples: MIN, MAX ,COUNT, SUM
  - Algebraic
    - Compute aggregate from constant-sized summary of subgroup
    - Examples: STDDEV, AVERAGE
    - For AVERAGE, summary data for each group is SUM, COUNT
  - Holistic
    - Require unbounded amount of information about each subgroup
    - Examples: MEDIAN, COUNT DISTINCT
    - Usually impractical for a data warehouses!

We see that calculating aggregates from aggregates is desirable, but is not possible for non-additive facts. So we deal with three types of aggregates i.e. distributive that are additive in nature, and then algebraic which are non-additive in nature. Therefore, such aggregates have to be computed from summary of subgroups to avoid the problem of incorrect results. The of course are the holistic aggregates that give a complete picture of the data, such as median, or distinct values. However, such aggregates are not desirable for a data warehouse environment, as it requires a complete scanning, which is highly undesirable as it consumes lot of time.

Step-3: Not recording Facts

- Transactional fact tables don't have records for events that don't occur
  - Example: No records(rows) for products that were not sold.
- This has both advantage and disadvantage.
  - Advantage:
    - Benefit of sparsity of data
    - Significantly less data to store for "rare" events
  - Disadvantage: Lack of information
  - Example: What products on promotion were not sold?

Fact tables usually don't records events that don't happen, such as items that were not sold. The advantage of this approach is getting around the problem of sparsity. Recall that when we discussed MOLAP, we discussed the sales of different items not occurring in different geographies and in different time frames, resulting in sparse cubes. If however this data is not recorded, then significantly less data will be required to be stored. But what if, from the point of view of decision making, such data has to be retrieved, how to retrieve data corresponding to those items? To find such items, additional queries will be required to check the current item balance with the item balance when the items where

(say) brought into the store. So the biggest disadvantage of this approach is key data is not recorded.

15.2 Step-3: A Fact-less Fact Table

- "Fact-less" fact table
  - A fact table without numeric fact columns
  - Captures relationships between dimensions
  - Use a dummy fact column that always has value 1

The problem of not recording non-events is solved by using fact-less fact tables, as not recording such information resulted in loss of data. Such a fact-less fact table is one which does not have numeric values stored in the corresponding column, as such tables are used to capture the relationships between dimensions. Fact less fact table captures the many-to-many relationships between dimensions, but contains no numeric or textual facts. To achieve this dummy value of 1 is used in the corresponding column.

Step-3: Example: Fact-less Fact Tables

Examples:

- Department/Student mapping fact table
  - What is the major for each student?
  - Which students did not enroll in ANY course
- Promotion coverage fact table
  - Which products were on promotion in which stores for which days?
  - Kind of like a periodic snapshot fact

Some of the examples of fact-less fact tables: Consider the case of a department/student mapping fact table. The data is recorded for each student who registers for a course, but there may be students that do not register in any course. If data is useful from the point of view of identifying those students which are skipping a semester, there is no direct or simple way to identify such students, the solution is a fact-less fact table. Similarly which items on promotion are not selling, as the sales records are for only those items that are sold.

15.3 Step-4: Handling Multi-valued Dimensions?

- One of the following approaches is adopted:
  - Drop the dimension.

- Use a primary value as a single value.
- Add multiple values in the dimension table.

Use “Helper” tables.

For handling the exceptions in dimensions, designers adopt one of the following approaches:

- Drop the Maintenance_Operation dimension as it is multi-valued.
- Choose one value (as the "primary" maintenance) and omit the other values.
- Extend the dimension list and add a fixed number of maintenance dimensions.
- Put a helper table in between this fact table and the Maintenance dimension table.

Instead of ignoring the problem and dropping the dimension altogether, let’s tackle the problem.

Usually the designers go for the second alternative, as a consequence this will show up as the primary, or main maintenance operation. Such as 20,000 Km maintenance or 40,000 Km maintenance. It is known what would constitute for each mileage based maintenance, and these maintenance are also mutually exclusive i.e. single valued. In many cases, you may actually come across this practice being observed in the OLTP systems. The obvious advantage is that the modeling problem is resolved, but the disadvantage is that the usefulness of the data becomes questionable. Why? Because with the passage of time or with new models of vehicles coming or because of company policy, what constitutes service at 20,000 Km may actually change. Will need meta data to resolve this issue.

The third alternative of creating a fixed number of additional columns in the dimension table is a quick and dirty approach and should be avoided. There is likely to be car that may require more changes then reflected in the table, or the company policy changes and more items fall under the maintenance, and a long list will result in many null entries for a typical car, especially new ones. Furthermore, it is not easy to query the multiple separate maintenance dimensions and will result in slow queries. Therefore, multiple dimensions style of design should be avoided.

The last alternative is usually adopted, and a "helper" table is placed between the Maintenance dimension and the fact table, although this adulterates or dilutes the star schema. More details are beyond the scope of this course.

Step-4: OLTP & Slowly Changing Dimensions

OLTP systems not good at tracking the past. History never changes.

OLTP systems are not “static” always evolving, data changing by overwriting.

Inability of OLTP systems to track history, purged after 90 to 180 days.
Actually don’t want to keep historical data for OLTP system.

One major difference between an OLTP system and a data warehouse is the ability and the responsibility to accurately describe the past. OLTP systems are usually very poor at

correctly representing a business as of a month or a year ago for several reasons as discussed before. A good OLTP system is always evolving. Orders are being filled and, thus, the order backlog is constantly changing. Descriptions of products, suppliers, and customers are constantly being updated, usually by overwriting. The large volume of data in an OLTP system is typically purged every 90 t o 180 days. For these reasons, it is difficult for an OLTP system to correctly represent the past. In an OLTP system, do you really want to keep old order statuses, product descriptions, supplier descriptions, and customer descriptions over a multiyear period?

15.4 Step-4: DWH Dilemma: Slowly Changing Dimensions

The responsibility of the DWH to track the changes.

Example: Slight change in description, but the product ID (SKU) is not changed.

Dilemma: Want to track both old and new descriptions, what do they use for the key? And where do they put the two values of the changed ingredient attribute?

The data warehouse must accept the responsibility of accurately describing the past. By doing so, the data warehouse simplifies the responsibilities of the OLTP system. Not only does the data warehouse relieve the OLTP system of almost all forms of reporting, but the data warehouse contains special structures that have several ways of tracking historical data.

A dimensional data warehouse database consists of a large central fact table with a multipart key. This fact table is surrounded by a single layer of smaller dimension tables, each containing a single primary key. In a dimensional database, these issues of describing the past mostly involve slowly changing dimensions. A typical slowly changing dimension is a product dimension in which the detailed description of a given product is occasionally adjusted. For example, a minor ingredient change or a minor packaging change may be so small that production does not assign the product a new SKU number (which the data warehouse has been using as the primary key in the product dimension), but nevertheless gives the data warehouse team a revised description of t he product. The data warehouse team faces a dilemma when this happens. If they want the data warehouse to track both the old and new descriptions of the product, what do they use for the key? And where do they put the two values of the changed ingredient

Step-4: Explanation of Slowly Changing Dimensions…

- Compared to fact tables, contents of dimension tables are relatively stable.
  - New sales transactions occur constantly.
  - New products are introduced rarely.
  - New stores are opened very rarely.

- The assumption does not hold in some cases
  - Certain dimensions evolve with time
  - e.g. description and formulation of products change with time
  - Customers get married and divorced, have children, change addresses etc.
  - Land changes ownership etc.
  - Changing names of sales regions.

For example, a minor ingredient change or a minor packaging change may be so small that production does not assign the product a new SKU number (which the data warehouse has been using as the primary key in the product dimension), but nevertheless gives the data warehouse team a revised description of t he product. The data warehouse team faces a dilemma when this happens. If they want the data warehouse to track both the old and new descriptions of the product, what do they use for the key? And where do they put the two values of the changed ingredient attribute?

Other common slowly changing dimensions are the district and region names for a sales force. Every company that has a sales force reassigns these names every year or two. This is such a common problem that this example is something of a joke in data ware housing classes. When the teacher asks, "How many of your companies have changed the organization of your sales force recently?" everyone raises their hands.

Step-4: Explanation of Slowly Changing Dimensions…

Although these dimensions change but the change is not rapid, therefore called "Slowly" Changing Dimensions.
There can be many examples. For a young customer who is single, then after a while the customer gets married. After sometime there are children, in unfortunate cases the marriage breaks so the customer is separated or the husband dies and the customer becomes a widow. This just does not typically happen overnight but takes a while. Another example is inheritance, consider the example of land. Over a period of time the land changes hands, is split because of inheritance or its size increases by buying. Again things don't happen overnight, but take a while, hence slowly changing dimensions.

Step-4: Handling Slowly Changing Dimensions

- Option-1: Overwrite History
  - Example: Code for a city, product entered incorrectly
  - Just overwrite the record changing the values of modified attributes.
  - No keys are affected.
  - No changes needed elsewhere in the DM.
  - Cannot track history and hence not a good option in DSS.

The first technique is the simplest and fastest. But it doesn't maintain past history! Nevertheless, overwriting is frequently used when the data warehouse team legitimately decides that the old value of the changed dimension attribute is not interesting . For example, if you find incorrect values in the city and state attributes in a customer record, then overwriting would almost certainly be used. After the overwrite, certain old reports that depended on the city or state values would not return exactly the same values. Most of us would argue that this is the correct outcome.

Step-4: Handling Slowly Changing Dimensions

- Option-2: Preserve History
  - Example: The packaging of a part change from glued box to stapled box, but the code assigned (SKU) is not changed.
  - Create an additional dimension record at the time of change with new attribute values.
  - Segments history accurately between old and new description
  - Requires adding two to three version numbers to the end of key. SKU#+1, SKU#+2 etc.

Suppose you work in a manufacturing company and one of your main data warehouse schemas is the company's shipments. The product dimension is one of the most important dimensions in this dimensional schema. A typical product dimension would have several hundred detailed records, each representing a unique product capable of being shipped. A good product dimension table would have at least 50 attributes describing the products, including hierarchical attributes such as brand and category, as well as nonhierarchical attributes such as color and package type. An important attribute provided by manufacturing operations is the SKU number assigned to the product. You should start by using the SKU number as the key to the product dimension table. Suppose that manufacturing operations makes a slight change in packaging of SKU #38, and the packaging description changes from "glued box" to "pasted box." Along with this change, manufacturing operations decides not to change the SKU number of the product, or the bar code (UPC) that is printed on the box. If the data warehouse team decides to track this change, the best way to do this is to issue another product record, as if the pasted box version were a brand new product. The only difference between the two product records is the packaging description. Even the SKU numbers are the same. The only way you can issue another record is if you generalize the key to the product dimension table to be something more than the SKU number. A simple technique is to use the SKU number plus two or three version digits. Thus the first instance of the product key for a given SKU might be SKU# + 01. When, and if, another version is needed, it becomes SKU# + 02, and so on. Notice that you should probably also park t he SKU number in a separate dimension attribute (field) because you never want an application to be parsing the key to extract the underlying SKU number. Note the separate SKU attribute in the Product dimension in Figure 1.

This technique for tracking slowly changing dimensions is very powerful because new dimension records automatically partition history in the fact table. The old version of the dimension record points to all history in the fact table prior to the change. The new version of the dimension record points to all history after the change. There is no need for a timestamp in the product table to record the change. In fact, a timestamp in the dimension record may be meaningless because the event of interest is t he actual use of the new product type in a shipment. This is best recorded by a fact table record with the correct new product key.

Another advantage of this technique is that you can gracefully track as many changes to a dimensional item as you wish. Each change generates a new dimension record, and each record partitions history perfectly. The main drawbacks of the technique are the requirement to generalize the dimension key, and the growth of the dimension table itself.

Step-4: Handling Slowly Changing Dimensions

- Option-3: Create current valued field
  - Example: The name and organization of the sales regions change over time, and want to know how sales would have looked with old regions.
  - Add a new field called current_region rename old to previous_region.
  - Sales record keys are not changed.
  - Only TWO most recent changes can be tracked.

15.5 Creating a Current Value Field

You use the third technique when you want to track a change in a dimension value, but it is legitimate to use the old value both before and after the change. This situation occurs most often in the infamous sales force realignments, where although you have changed the names of your sales regions, you still have a need to state today's sales in terms of yesterday's region names, just to "see how they would have done" using the old organization. You can attack this requirement, not by creating a new dimension record as in the second technique, but by creating a new "current value" field. Suppose in a sales team dimension table, where the records represent sales teams, you have a field called "region." When you decide to rearrange the sales force and assign each team to newly named regions, you create a new field in the sales dimension table called "current_region." You should probably rename the old field "previous_region." No alterations are made to the sales dimension record keys or to the number of sales team records. These two fields now allow an application to group all sales fact records by either the old sales assignments (previous region) or the new sales assignments (current region). This schema allows only the most recent sales force change to be tracked, but it offers the immense flexibility of being able to state all of the history by either of the two sales force assignment schemas. It is conceivable, although somewhat awkward, to generalize this approach to the two most recent changes. If many of these sales force realignments take place and it is desired to track them all, then the second technique should probably be used.

15.6 Step-4: Pros and Cons of Handling

- Option-1: Overwrite existing value
  - \+ Simple to implement
  - \+ No tracking of history

- Option-2: Add a new dimension row
  - \+ Accurate historical reporting
  - \+ Pre-computed aggregates unaffected
  - \+ Dimension table grows over time

- Option-3: Add a new field
  - + Accurate historical reporting to last TWO changes
  - + Record keys are unaffected
  - + Dimension table size increases

There are number of ways of handling slowly changing dimensions. Some of the methods are simple, but not desirable; but all have their own pros and cons. The simplest possible "solution" is to overwrite history. If the customer was earlier single, and gets married, just change his/here status from single to married. Very simple to implement, but not desirable, as a DWH is about recording historical data, and by virtue of overwriting, the historical data is destroyed. Another option is to add a row when the dimension changes, the obvious benefit is that history is not lost, but over the period of time the dimension table will grow as new rows are added corresponding to the changes in the dimensions. The third, rather desirable approach is to add an additional column, that does increase the table size, but the increase is not non-deterministic. The column records the last two changes, if the dimension changes more than twice, then historical data is lost.

15.7 Step-4: Junk Dimension

- Sometimes certain attributes don't fit nicely into any dimension
  - Payment method (Cash vs. Credit Card vs. Check)
  - Bagging type (Paper vs. Plastic vs. None)

- Create one or more "mix" dimensions
  - Group together leftover attributes as a dimension even if not related
  - Reduces number of dimension tables, width of fact table
  - Works best if leftover attributes are
    - Few in number
    - Low in cardinality
    - Correlated

- Other options
  - Each leftover attribute becomes a dimension
  - Eliminate leftover attributes that are not useful

A junk dimension is a collection of random transactional codes, flags and/or text attributes that are unrelated to any particular dimension. The junk dimension is simply a structure that provides a convenient place to store the junk attributes. A good example would be a trade fact in a company that brokers equity trades.

The need for junk dimensions arises when we are considering single level hierarchies. The only problem with the single level hierarchies is that you may have a lot of them in any given dimensional model. Ideally, the concatenated primary key of a fact table should consist of fewer than 10 foreign keys. Sometimes, if all of the yes/no flags are represented as single level hierarchy dimensions, you may end up with 30 or more. Obviously, this is an overly complex design.

A technique that allows reduction of the number of foreign keys in a fact table is the creation of "junk" dimensions. These are just "made up" dimensions where you can put

several of these single level hierarchies. This cuts down the number of foreign keys in the fact table dramatically.

As to the number of flags before creating a junk dimension, if there are more than 15 dimensions, where five or more are single level hierarchies, I start seriously thinking about combining them into one or more junk dimensions. One should not indiscriminately combine 20 or 30 or 80 single level hierarchies.

Masters

Lecture 16: Extract Transform Load (ETL)

Learning Goals

- Understand Extract from the operational system
- Understand Transform the data, which includes data cleansing
- Understand Loading the data into the DWH

In a DWH project, 50% of the work is related to ETL. ETL stands for Extract Transform Load. Some people call it ETML i.e. Extract Transform Move Load, as you have to move the data into the DWH. There is a significant part after data transformation that involves data cleansing i.e. there is a C also here. However, ETL is a pretty standard definition, and normally when you say ETL people know what you mean.

E: Extract from the operational system
T: Transform the data, which includes data cleansing
L: Loading the data into the DWH

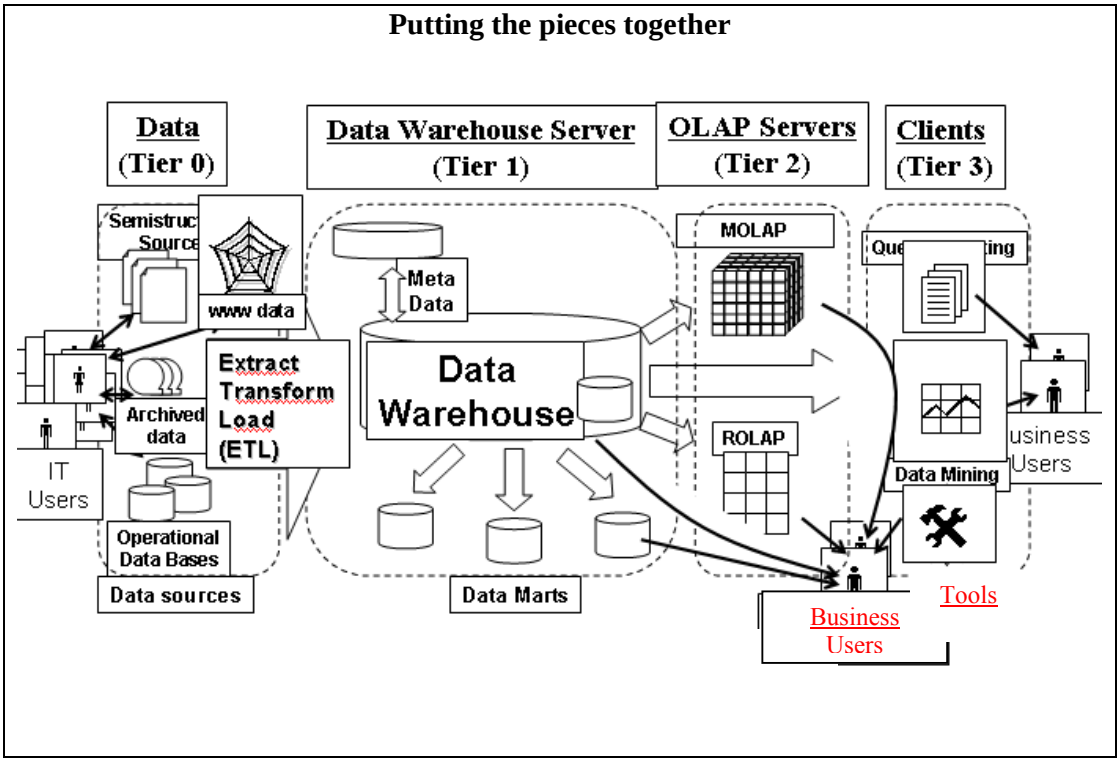


Figure-16.1: ETL: Putting the pieces together

Figure-16.1 shows a multi-tiered data warehousing architecture. We can see that at the lowest level (tier 0) lies the data sources like operational, archived and/or web etc. Data Warehouse Server lays in the next level (tier 1). We can see in the Figure 16.1 that ETL box lies at the boundaries of both the tiers 0 and 1. That means it serves as a bridge between data source systems and the actual data warehouse. Thus the source data is processed using some set of activities before bringing it into the data warehouse environment. These set of activities comprise the ETL process.

16.1 The ETL Cycle

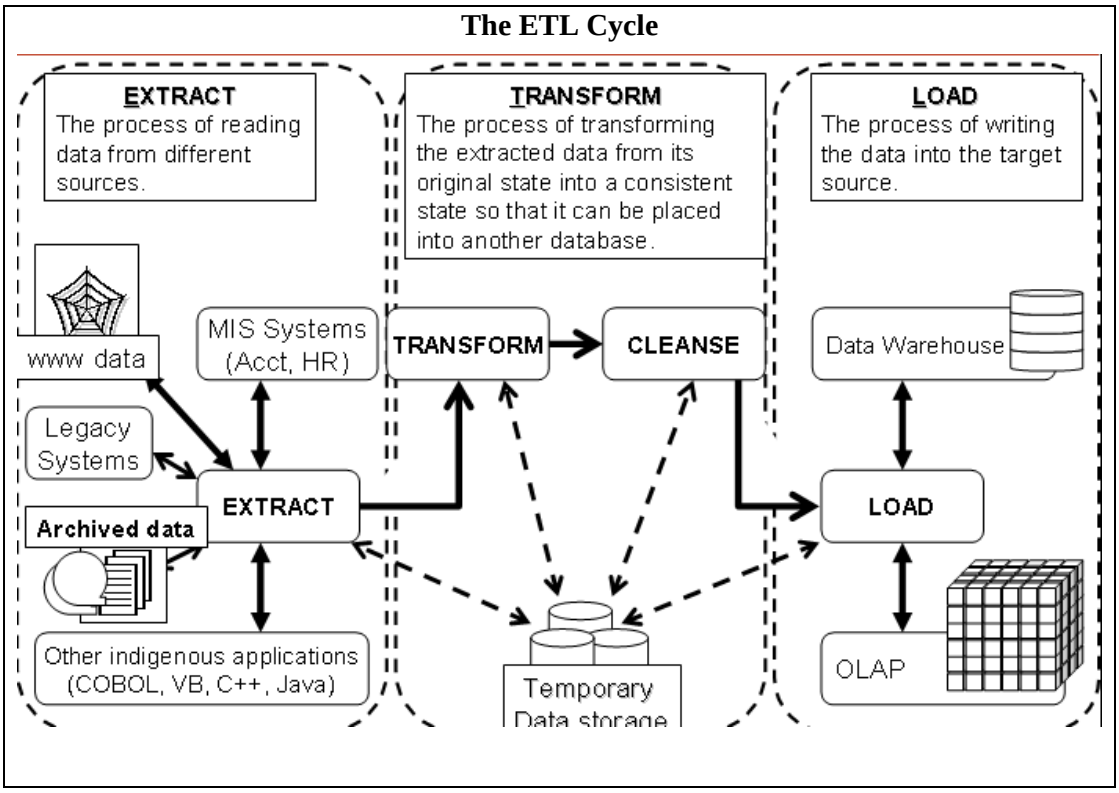


Figure-16.2: The ETL Cycle

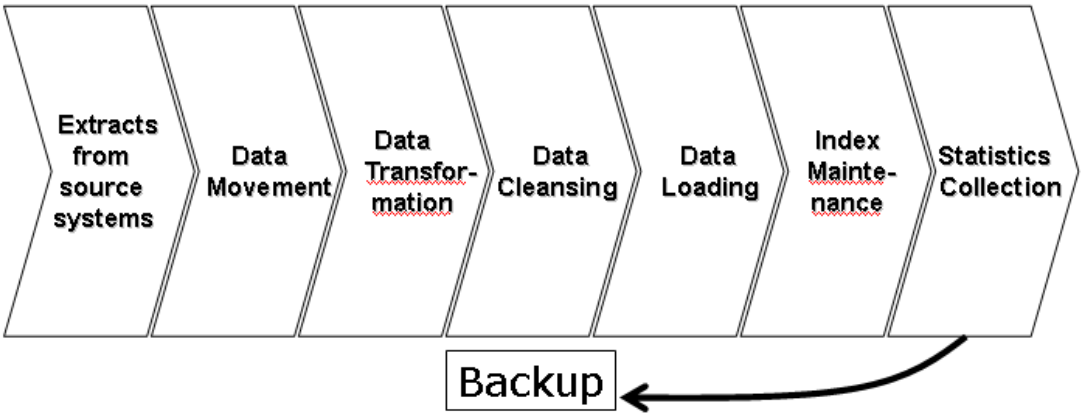
Now what are those activities in the ETL process shown as a box in the Figure 16.1? Let's have a look inside the box to find the sequence of activities that constitute the ETL process. As shown in Figure 16.2 ETL process consists of three major activities separated by dotted lines. The first step is the *Extract* which is the activity of reading data from multiple heterogeneous data sources. Next comes the *Transform* step which is the activity of transforming the input data from multiple heterogeneous environments into a single consistent state. Another important activity at this level is the *data cleansing* which is the activity of noise removal from input data before bringing it in the DWH environment. The final step is the *Load* which is an activity of loading cleansed data in to the target system.

ETL Processing

ETL is independent yet interrelated steps.

It is important to look at the big picture.

Data acquisition time may include…



Back-up is a major task, its a DWH not a cube

Figure-16.3: ETL Processing

When we look at ETL processing, it is important to look at the big picture as shown in Figure 16.3. It is not just the transformation; there is a whole lot of work you have to consider in the design and architecture of the ETL model. You have to design to extract from source system, how to get the data out of the operational databases. You have to move the data to the dedicated server where the transformations are being done or on the very least if the transformations will be done on the same systems, start moving the data into the warehouse environment. You have to look at the data movement and all the network implications of data movement. The transformations we will be talking about and the data loading strategies all have to be considered. It is not just good enough to talk about data loading, we have to do index maintenance, statistics collection, summary, data maintenance, data mart construction, data back ups, a whole process has to be followed each time you refresh or reload the data warehouse.

The architects really need to consider all of these. We will be focusing on all the major operations of ETL, so as to understand the big picture. It is certainly true that the volume of the data warehouse will make a difference on how you do the backups and the technology. Consider a cube which is not a data warehouse, it is a data mart. Typically a data mart is much smaller and probably pretty easier to backup just part of a file system backup on wherever your cube server is. The data warehouse is usually much bigger. So for a data warehouse you usually do a full tape backup, and a tape-backup would ideally use a robotic tape library and all the associated setup. Typically you don't need robotic libraries for the cubes and data marts, but you do need them for data warehouse. But again you have to come back to economics, robotic tape libraries are very expensive, and

so you may actually use manual tape mounting, because it is not worth paying for robots when you can have humans do it on a very different price range.

16.2 Overview of Data Extraction

First step of ETL, followed by many.

Source systems for extraction are typically OLTP systems.

A very complex task due to number of reasons:

- Very complex and poorly documented source system.
- Data has to be extracted not once, but number of times.

The process design is dependent on:

- Which extraction method to choose?
- How to make available extracted data for further processing?

Extraction is the operation of extracting data from a source system for further use in a data warehouse environment. This is the first step of the ETL process. After the extraction, this data can be transformed, cleansed and loaded into the data warehouse.

The source systems for a data warehouse are typically transaction processing applications. For example, one of the source systems for a sales analysis data warehouse might be an order entry system that records all of the current order activities.

Designing and creating the extraction process is often one of the most time-consuming tasks in the ETL process and, indeed, in the entire data warehousing process. The source systems might be very complex and poorly documented, and thus determining which data needs to be extracted can be difficult. The data has to be extracted normally not only once, but several times in a periodic manner to supply all changed data to the warehouse and keep it up-to-date. Moreover, the source system typically cannot be modified, nor can its performance or availability be adjusted, to accommodate the needs of the data warehouse extraction process. These are important considerations for extraction and ETL in general.

Designing this process means making decisions about the following two main aspects:

- Which extraction method to choose?

This influences the source system and the time needed for refreshing the warehouse.

- How to make available extracted data for further processing?

This influences the transportation method, and the need for cleaning and transforming the data.

16.3 Types of Data Extraction

- **Logical Extraction**
  - Full Extraction
  - Incremental Extraction

- **Physical Extraction**
  - Online Extraction

- Offline Extraction
- Legacy vs. OLTP

There are different types of data extraction which can be broadly categorized into two data extraction techniques, logical and physical. The extraction method you should choose is highly dependent on the source system and also on the business needs in the target data warehouse environment. Very often, there's no possibility to add additional logic to the source systems to enhance an incremental extraction of data due to the performance or the increased workload of these systems. Sometimes even the customer is not allowed to add anything to an out-of-the-box (shrink wrapped CD installable) application system as the performance of the source system has been maximized and carefully calibrated and monitored. The reason being creation of the DWH does not directly help the operational people in any way, for them asking for data every now and then is an annoyance.

The estimated amount of the data to be extracted and the stage in the ETL process (initial load or maintenance of data i.e. data already loaded) may also effect the decision of how to extract, from a logical and a physical perspective. Basically, you have to decide how to extract data logically and physically. Let's discuss logical techniques first in detail.

16.4 Logical Data Extraction

- **Full Extraction**
  - The data extracted completely from the source system.
  - No need to keep track of changes.
  - Source data made available as-is w/o any additional information.
- **Incremental Extraction**
  - Data extracted after a well defined point/event in time.
  - Mechanism used to reflect/record the temporal changes in data (column or table).
  - Sometimes entire tables off-loaded from source system into the DWH.
  - Can have significant performance impacts on the data warehouse server.

The two logical data extraction types are full and incremental extraction techniques.
Let's look at the two in detail.

Full Extraction

The data is extracted completely from the source system. Since this extraction reflects all the data currently available in the source system, there's no need to keep track of changes to the data source since the last successful extraction. The source data will be provided as-is and no additional logical information (for example, timestamps etc) is necessary on the

source site. An example for a full extraction may be an export file of a distinct table or a remote SQL statement scanning the complete source table.

Incremental Extraction

At a specific point in time, only the data that has changed since a well-defined event back in history will be extracted. This event may be the last time of extraction or a more complex business event like the last day of balancing of accounts. To identify this incremental change there must be a mechanism to identify all the changed information since this specific time event. This information can be either provided by the source data itself like an application column, reflecting the last-changed timestamp or a change table where an appropriate additional mechanism keeps track of the changes besides the originating transactions. In most cases, using the latter method means adding extraction logic to the source system.

Many data warehouses do not use any change-capture techniques as part of the extraction process. Instead, entire tables from the source systems are extracted to the data warehouse or staging area, and these tables are compared with a previous extract from the source system to identify the changed data. This approach may not have significant impact on the source systems, but it can clearly place considerable burden on the data warehouse processes, particularly if the data volumes are large.

16.5 Physical Data Extraction…

- **Online Extraction**
  - Data extracted directly from the source system.
  - May access source tables through an intermediate system.
  - Intermediate system usually similar to the source system.

- **Offline Extraction**
  - Data NOT extracted directly from the source system, instead staged explicitly outside the original source system.
  - Data is either already structured or was created by an extraction routine.
  - Some of the prevalent structures are:
    - Flat files
    - Dump files
    - Redo and archive logs
    - Transportable table-spaces

Physical Extraction Methods

Depending on the chosen logical extraction method and the capabilities and restrictions on the source side, the extracted data can be physically extracted by two mechanisms. The data can either be extracted online from the source system or from an offline structure. Such an offline structure might already exist or it might be generated by an extraction routine.

Online Extraction

The data is extracted directly from the source system itself. The extraction process can connect directly to the source system to access the source tables themselves or to an intermediate system that stores the data in a preconfigured manner (for example, snapshot logs or change tables). Note that the intermediate system is not necessarily physically

different from the source system. With online extractions, you need to consider whether the distributed transactions are using original source objects or prepared source objects.

Offline Extraction

The data is not extracted directly from the source system but is staged explicitly outside the original source system. The data already has an existing structure (for example, redo logs, archive logs or transportable table-spaces) or was created by an extraction routine.

You should consider the following structures:

- Flat files
- Data in a defined, generic format. Dump files
- DBMS-specific format. Redo and archive logs
- Transportable table-spaces

Physical Data Extraction

- **Legacy vs. OLTP**
  - Data moved from the source system
  - Copy made of the source system data
  - Staging area used for performance reasons

During extraction, data may be removed from the source system or a copy made and the original data retained in the source system. It is common to move historical data that accumulates in an operational OLTP system to a data warehouse to maintain OLTP performance and efficiency. Legacy systems may require too much effort to implement such offload processes, so legacy data is often copied into the data warehouse, leaving the original data in place. Extracted data is loaded into the data warehouse staging area (a relational database usually separate from the data warehouse database), for manipulation by the remaining ETL processes. Data extraction processes can be implemented using SQL stored procedures, Vendor tools, or custom applications developed in programming or scripting languages.

16.6 Data Transformation

- **Basic tasks**
  - Selection
  - Splitting/Joining
  - Conversion
  - Summarization
  - Enrichment

The next operation in the ETL process is the Data Transformation. The major tasks performed during this phase vary depending on the application; however the basic tasks are discussed here.

Selection: This takes place at the beginning of the whole process of data transformation. You select either whole records or parts of several records from the source systems. The task of selection usually forms part of the extraction function itself. However, in some cases, the composition of the source structure may not be supporting selection of the necessary parts during data extraction. In these cases, it is advised to extract the whole record and then do the selection as part of the transformation function.

Splitting/joining: This task includes the types of data manipulation you need to perform on the selected parts of source records. Sometimes (uncommonly), you will be splitting the selected parts even further during data transformation. Joining of parts selected from many source systems is more widespread in the data warehouse environment.

Conversion: This is an all-inclusive task. It includes a large variety of rudimentary conversions of single fields for two primary reasons (i) to standardize among the data extractions from disparate source systems, and (ii) to make the fields usable and understandable to the users.

Summarization: Sometimes you may find that it is not feasible to keep data at the lowest level of detail in your data warehouse. It may be that none of your users ever need data at the lowest granularity for analysis or querying. For example, for a grocery chain, sales data at the lowest level of detail for every transaction at the checkout may not be needed. Storing sales by product by store by day in the data warehouse may be quite adequate. So, in this case, the data transformation function includes summarization of daily sales by product and by store.

Enrichment: This task is the rearrangement and simplification of individual fields to make them more useful for the data warehouse environment. You may use one or more fields from the same input record to create a better view of the data for the data warehouse. This principle is extended when one or more fields originate from multiple records, resulting in a single field for the data warehouse.

To better understand lets discuss conversion and enrichment with examples.

Data Transformation Basic Tasks: Conversion

- **Convert common data elements into a consistent form i.e. name and address.**

Table-16.1 (a)

Field format		Field data
First-Family-title	→	Muhammad Ibrahim Contractor
Family-title-comma-first	→	Ibrahim Contractor, Muhammad
Family-comma-first-title	→	Ibrahim, Muhammad Contractor

- **Translation of dissimilar codes into a standard code.**

Table-16.1(b)

Natl. ID	→	NID
National ID	→	NID

F/NO-2
F-2
FL.NO.2
FL.2
FL/NO.2 → FLAT No. 2
FL-2
FLAT-2
FLAT#
FLAT,2
FLAT-NO-2
FL-NO.2

Table-16.1: Data Transformation Basic Tasks: Conversion

As discussed earlier, conversion is performed to standardize data by converting data from multiple heterogeneous sources into a single consistent form making it usable and understandable. For example, a data field containing name and job title can be represented in a number of ways in different source systems as shown in Table 16.1(a). Here three different formats have been shown for the same field. Similarly, Table 16.1(b) shows another example of code standardization of dissimilar code representations. Here, two different codes have been shown for representing *National Identity* which can be converted to a consistent form like NID. Same is the case for address representation like house number and flat number etc.

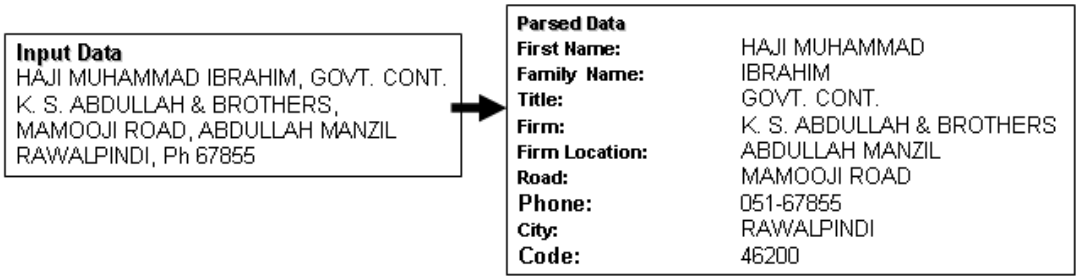
Data Transformation Basic Tasks: Conversion

- Data representation change
  - EBCIDIC to ASCII
- Operating System Change
  - Mainframe (MVS) to UNIX
  - UNIX to NT or XP
- Data type change
  - Program (Excel to Access), database format (FoxPro to Access).
  - Character, numeric and date type.
  - Fixed and variable length.

Why differences in data from different sources? Three common reasons are because data at different locations may have different data representation codes, different operating systems and different data types.

Data Transformation Basic Tasks: Enrichment

- **Data elements are mapped from source tables and files to destination fact and dimension tables.**



(a) **(b)**

- **Default values are used in the absence of source data.**
- **Fields are added for unique keys and time elements.**

Figure-16.2: Data Transformation by Enrichment

Enrichment is one of the basic tasks of data transformation as shown in Figure 16.2. Figure-16(a) shows input data that does not have any associated information that links semantics with the given data i.e. what means what? Hover after enrichment the contents of the input data are assigned to the corresponding attributes or fields. Recognize that this could be a very difficult task if the order of the contents of the input varies across the records. Assuming that is not the case, the assignment can be very straightforward. There are certain values that are not given in the input but are implied, such as the postal code, or the phone code etc. In such a case default values are used based on the input data using standard default values. Other information added could be unique keys for identification of data, such that the keys are independent of the business rules, also the data could be time stamped to ascertain its "freshness".

16.7 Significance of Data Loading Strategies

- Need to look at:
  - Data freshness
  - System performance
  - Data volatility

- Data Freshness
  - Very fresh low update efficiency
  - Historical data, high update efficiency
  - Always trade-offs in the light of goals

- System performance
  - Availability of staging table space
  - Impact on query workload

- Data Volatility
  - Ratio of new to historical data
  - High percentages of data change (batch update)

Once data has been transformed, the *loading* operation is performed. There are different loading strategies and choosing the one depends on data freshness and performance. We also need to look into data volatility, in other words, how much data is changed within the window of the refresh. So in general if real time or near real-time availability of data is required, meaning low update efficiency, because if want the data to be very fresh, then I would not allow accumulation of lot of data, because it means waiting a long time before data is inserted. Whereas, if I want the maximum update efficiency i.e. highest performance, then batch processing would be required. Thus as always there is a tradeoff between performance and data freshness and you need to look at the goals and objectives of your data warehouse, how you follow them and how important these characteristics are. So again there is no one right or wrong answer, depends on what are the goals of the data warehouse, and then you design appropriate to those goals.

The data loading strategy will also depend on the data storage requirements. Depending upon which strategy you use, some require more data storage than others, staging tables etc. You want to look at the impact on query workloads, so when loading the data, at the same time these queries are running, what does that mean? Meaning what is allowable and what is not allowable.

It is also important to look at the ratio of existing to new data? And that will determine or help determine what the strategy should be for implementation. So we need to look at what are the characteristics of workloads that I am interested in. So there are clearly some tradeoffs here. If we look at a loading strategy with a high percentage of data changes per data block, this normally means I am doing a batch update. In other words, I have got a lot of data that I am inserting into the table, and therefore each data block has a lot of rows that are changing per data block.

Three Loading Strategies

- Once we have transformed data, there are three primary loading strategies:
- Full data refresh with BLOCK INSERT or 'block slamming' into empty table.
- Incremental data refresh with BLOCK INSERT or 'block slamming' into existing (populated) tables.
- Trickle/continuous feed with constant data collection and loading using row level insert and update operations.

There can be a couple of lectures on loading strategies, but in this course will limit our selves to the basic concepts only. Once the data has been transformed, it is ready to be loaded into the DWH, for this purpose three loading strategies are prevalent. When the tables are populated for the first time it is a full data refresh, depending on the tool or the environment being used, this may be called as BLOCK INSERT or "block slamming" i.e. large blocks of data are loaded, usually in the form of batch updates. As the DWH evolves over time, there may be no empty tables, but already filled table have to be updated. For this purpose DWH has to be refreshed incrementally. Depending on the line of business and other reporting requirements the refresh period will vary and also the amount of data to be loaded. In extreme cases this may boil down to BLOCK INSERT or "block slamming" into the main DWH tables. This will have certain performance implications with reference to availability of data, and the outcome of the queries while the data is being refreshed and requirement of additional memory space in the form of "shadow tables", but the discussion is beyond the scope of this course. The third and final loading strategy is trickle feed. This strategy is more in the context of active data warehousing and has a high update overhead. Usually the criterion when to load depends on the time interval between uploads or the amount of data recorded or their combination.

Masters

Lecture 17: Issues of ETL

Learning Goals

- Why ETL Issues?
- Understand Complexity of problem/work underestimated
- Multiple sources for same data element

17.1 Why ETL Issues?

Data from different source systems will be different, poorly documented and dirty. Lot of analysis required.

Easy to collate addresses and names? Not really. No address or name standards.

Use software for standardization. Very expensive, as any "standards" vary from country to country, not large enough market.

It is very difficult to extract the data from different source systems, because by definition they are heterogeneous, and as a consequence, there are multiple representations of the same data, have inconsistent data formats, have very poor documentation and have dirty data etc. Therefore, you have to figure different ways of solving this problem, requiring lot of analysis.

Consider the deceptively simple task of automating the collating of addresses and names from different operational databases. You can't predict or legislate format or content, therefore, there are no address standards, and there are no standard address and name cleaning software.

There is software for certain geographies, but the addresses and names in US will be completely different from that in Austria or Brazil. Every country finds a way of writing the addresses differently, that is always a big problem. Hence automation does not solve the problem, and also has a high price tag in case of 1st generation ETL tools ($200K-$400K range).

Why ETL Issues?

- Things would have been simpler in the presence of operational systems, but that is not always the case.
- Manual data collection and entry. Nothing wrong with that, but potential to introduces lots of problems.
- Data is never perfect. The cost of perfection, extremely high vs. its value.

Most organizations have ERP systems (which are basically operational systems) from where they load their DWH. However, this is not a rule, but used in an environment where transaction processing systems are in place, and the data is generated in real time.

However, in the case of a country-wide census, there are no source systems, so data is being collected directly and manually. People go form door to door and get forms filled, subsequently the data in its most raw form is manually entered into the database. There is nothing wrong with that, the problem being human involvement in data recording and storage is very expensive, time consuming and prone to errors. There have to be control and processes to collect as clean a data as possible.

People have a utopian view that they will collect perfect data. This is wrong. The cost of collecting perfect data far exceeds its value. Therefore, the objective should be to get as clean data as can be based on the given constraints.

"Some" Issues

- Usually, if not always underestimated
- Diversity in source systems and platforms
- Inconsistent data representations
- Complexity of transformations
- Rigidity and unavailability of legacy systems
- Volume of legacy data
- Web scrapping

ETL is something which is usually always underestimated; consequently there are many many issues. There can be an entire course on ETL. Since this course is not only about ETL, therefore, we will limit our self to discussing only some of the issues. We will now discuss some of the more important issues one-by-one.

Complexity of problem/work underestimated

- Work seems to be deceptively simple.
- People start manually building the DWH.
- Programmers underestimate the task.
- Impressions could be deceiving.
- Traditional DBMS rules and concepts break down for very large heterogeneous historical databases.

At first glance, when data is moved from the legacy environment to the data warehouse environment, it appears there is nothing more going on than simple extraction of data from one place to the next. Because of the deceptive simplicity, many organizations start to build their data warehouse manually. The programmer looks at the movement of data from the old operational environment to the new data warehouse environment and declares "I can do that!" With pencil and writing pad in hand the programmer anxiously jumps into the creation of code in the first three minutes of the design and development of the data warehouse.

However, first impressions can be deceiving - in this case very deceiving. What at first appears to be nothing more than the movement of data from one place to another quickly turns into a large and complex task, actually a nightmare. The scope of the problem being far larger and far more complex than the programmer had ever imagined.

Diversity in source systems and platforms

Platform	OS	DBMS	MIS/ERP
Main Frame	VMS	Oracle	SAP
Mini Computer	Unix	Informix	PeopleSoft
Desktop	Win NT	Access	JD Edwards
	DOS	Text file	

Dozens of source systems across organizations

Numerous source systems within an organization

Need specialist for each

Table-17.1: Diversity in source systems and platforms

There are literally more than a dozen different source system technologies in use in the market today. Example of the kind of source system technologies that you often see are shown in Table 17.1. The team doing ETL processing in any given organization probably has to understand at least 6 of them. You will be surprised to find big banks and telecommunication companies that have every single one of these technologies; in a single company. It is a nightmare to understand them. The only viable solution is to have a technical expert that knows how to get data from every single one of these formats. In short it's a nightmare.

Same data, different representation

Date value representations

Examples:

970314	1997-03-14
03/14/1997	14-MAR-1997
March 14 1997	2450521.5 (Julian date format)

Gender value representations

Examples:

- Male/Female - M/F
- 0/1 - PM/PF

There are 127 different ways to spell AT&T; there are 1000 ways to spell duPont. How many ways are there to represent date? What is the Julian date format? All of these questions and many more are at the core of inconsistent data representation. Don't be surprised to find source system with Julian date format or dates stored as text strings.

Consider the case of gender representation. You might think it is just male and female. Wrong. It's not just male and female, it also includes unknown, because for some people you don't have their gender data. If this was not sufficient, there is another twist to the gender i.e. instead of M and F, you come across PM and PF. What is this? This is

probable male and probable female i.e. based on the name the gender has been guessed. For example, if the name is Mussarat it is probable female. It is not for sure, but we think it is. If it's Riffat, it is probable male.

So you get all these weird and strange representations in operational data, while in a DWH there has to be only ONE and consistent representation. You just can't dump all the operational data independent of which source system it came from that has a different data representation. Remember in the data model you should have non-overlapping consistent domains for every attribute.

Multiple sources for same data element

- Need to rank source systems on a per data element basis.
- Take data element from source system with highest rank where element exists.
- "Guessing" gender from name
- Something is better than nothing?
- Must sometimes establish "group ranking" rules to maintain data integrity.
- First, middle and family name from two systems of different rank. People using middle name as first name.

This means you need to establish ranking in the source system on per element (attribute) basis. Ranking is all about selecting the "right" source system. Rank establishment has to be based on which source system is known to have the cleanest data for a particular attribute. Obviously you take the data element from the source system with the highest rank where the element exists. However, you have to be clever about how you use the rank.

For example, consider the case of the gender data coming from two different source systems A and B. It may be the case that the highest quality data is from source system A, where the boxes for the gender were checked by the customers themselves. But what if someone did not check the gender box? Then you go on to the next cleanest source system i.e. B, where the gender was guessed based on the name.

Obviously the quality of available data for source system B is not as good as that of source system A, but since you do not have data for this particular individual in source systems A, so it is better to have something then nothing. This is arguable i.e. maybe it is better to have nothing than to have dirty data. The point is if you have some level of confidence in the data you should take it. Typically it is a good idea to remember from where you took the data, so you can use this information from an analytic point of view e.g. where you get clean data etc.

Consider the case of name i.e. first name, middle name and family name and two source systems i.e. C and D. Assume that source system C has higher quality data entry, and management and processes and controls as compared to D. Also assume that source

system C does not have the middle name, while source system D has the complete name. Since C has a higher precedence, so you take the first name and the family name from it. But you really would like to have the middle name so you take it from source system D. This turns out to be a big problem, because people sometimes use their middle name as their first name e.g. *Imran Baqai* instead of *Shezad Imran Baqai*.

17.2 Complexity of required transformations

Simple one-to-one scalar transformations
- 0/1 → M/F

One-to-many element transformations
- 4 x 20 address field → House/Flat, Road/Street, Area/Sector, City.

Many-to-many element transformations
- House-holding (who live together) and individualization (who are same) and same lands.

There is a spectrum of simple all the way up to very complex transformations that you can implement. And you probably end up with many in most DWH deployments. The transformations are typically divided into three categories as follows:

- Simple one-to-one scalar transformations.
- One-to-many element transformations.
- Complex many-to-many element transformations.

Simple scalar transformation is a one-to-one mapping from one set of values to another set of values using straightforward rules. For example, if 0/1 corresponds to male/female in one source system, then the gender is stored as male/female in the DW.

A one-to-many transformation is more complex than scalar transformation. As a data element form the source system results in several columns in the DW. Consider the 6×30 address field (6 lines of 30 characters each), the requirement is to parse it into street address lines 1 and 2, city, sate and zip code by applying a parsing algorithm.

The most complex is many-to-many element transformations. Good examples are house holding and individualization. This is achieved by using candidate keys and fuzzy matching to determine which individuals are the same individuals, and which individuals go in the same household and so on. This is a very complex transformation and will be discussed in BSN lecture.

Rigidity and unavailability of legacy systems

- Very difficult to add logic to or increase performance of legacy systems.
- Utilization of expensive legacy systems is optimized.
- Therefore, want to off-load transformation cycles to open systems environment.
- This often requires new skill sets.

Legacy systems typically mean mainframe type systems that are very expensive to buy and operate. We are talking about CPU cycles as well as storage, but mainly computing CPU cycles. The mainframe machines are usually 110% utilized in most businesses. The reason being, businesses want the "most bang for the buck", so they keep on putting more and more load on the machines.

Consider an expensive ERP system, when is the system available for the processing of DWH transformations? Probably never, as the system utilization has already been maxed up. So the big question is where to do the transformations? Where do the CPU cycles come from? And when the legacy data is in a mainframe environment, typically the cost of transformation is very high because of scarcity and cost of legacy cycles. But you don't want the cost of constructing the DWH to exceed the value of the DWH. So you need to think of ways to move the cycles to more cost efficient environments.

Therefore, in case of legacy environment, generally you want to off-load the transformation cycles to open systems environment. Open system i.e. Intel chipset, operating systems like Unix and NT etc. as it results in a much more cost effective approach than doing things in a mainframe type of environment. However, this may sometimes be problematic for some organizations requiring certain new skill sets. For example the IT department of a bank may have only used a mainframe for day-to-day operations, and moving on to open systems may be painful for them due to the learning curve etc.

It is NOT simply moving the data to open systems; there are lots of issues due to legacy systems, simple as well as complex. For example, in the mainframe environment you have EBCDIC character representation and in open systems it is always ASCII, so you have to do the conversion. In mainframe environment there is packed decimal encoding, zone decimal and weird date formats that are not understandable in the open systems environment. So you have to have utilities for doing the conversion and so on before the required transformation takes place.

Volume of legacy data

- Talking about not weekly data, but data spread over years.
- Historical data on tapes that are serial and very slow to mount etc.
- Need lots of processing and I/O to effectively handle large data volumes.
  - Need efficient interconnect bandwidth to transfer large amounts of data from legacy sources to DWH.

The volume of legacy data is typically very large. It is not just going back and getting tape for one month of billing data, but getting billing data for as much as one could get hold of (say), three years of billing data. Imagine the volume of work, lots of processing and I/Os especially when the data is on tapes. This would involve tape mounts, and manual tape mounts is painstakingly slow. It takes forever to mount hundreds and hundreds of tapes. The IT people will hate you after you are through with tape mounts and data retrieval.

Assuming transformation takes place on the mainframe; the next issue is to move large volumes of data from the mainframe into the DWH requiring network bandwidth. The data may have to be moved across the LAN, and maybe even across the WAN which is more painful. Therefore, the DWH architects need to pay a lot of attention to capacity planning issues of the DWH i.e. how to size the DWH, how much is it going to cost to extract all the data, prepare the data, and move the data from a network capacity planning perspective. Ensure that from a capacity planning point of view all these aspects are considered.

17.3 Web scrapping

- Lot of data in a web page, but is mixed with a lot of "junk".

- Problems:
  - Limited query interfaces
    - Fill in forms

  - "Free text" fields
    - E.g. addresses

  - Inconsistent output
    - i.e., html tags which mark interesting fields might be different on different pages.

  - Rapid change without notice.

During the last decade the web has grown from a curiosity to a required information repository. From news headlines and stories, financial data and reports, entertainment events, online shopping and business to business commerce, the Web has it all. The promise of the web is that all of this information will be easily maintained, monitored, queried, and browsed. Some of the promise has already been realized but much remains to be done. Web browsers (such as Internet Explorer and Netscape) have been very successful at displaying web-based information on a computer monitor.

However, there remains a divide between content presented on web pages and the ability to extract that content and make it actionable. What companies and individuals need is a method to extract or mine only that data or content in which they are interested. Furthermore, they need to be able to take that mined content, apply different kinds of data processing, and make it available for use in different applications, and have it delivered to a database, email, spreadsheet, mobile phone, or whatever delivery destination device is available.

Web scrapping is a process of applying screen scrapping techniques to the web. There are several web scrapping products in the market and target business users who want to creatively use the data, not write complex scripts. Some of the uses of scrapping are:

- Building contact lists
- Extracting product catalogs
- Aggregating real-estate info
- Automating search Ad listings

- Clipping news articles etc.

Beware of data quality (or lack of it)

- Data quality is always worse than expected.
- Will have a couple of lectures on data quality and its management.
- It is not a matter of few hundred rows.
- Data recorded for running operations is not usually good enough for decision support.
    - Correct totals don't guarantee data quality.
    - Not knowing gender does not hurt POS.
    - Centurion customers popping up.

Personal word of warning, the data quality will be always be worse than you expect, you can count on this. Everybody says that other people have dirty data (like everyone thinks they will win the lottery and other people will have an accident), but my data is clean. I don't believe you; your data is always dirty. You need to allocate time and resources to facilitate the data cleanup. In the next series of lectures, we will talk about a methodology using TQM techniques and apply to DWH data quality. There is a whole methodology on how to deliver data quality in a data warehouse environment. This is just a warning, data quality is always worse than you think it is. You have used data for transaction processing; it does not mean it is good for decision support.

Everybody always underestimate how dirty their data is. It is not a few hundred rows of dirty data. Data is always dirtier than you think. I have not seen a single business environment that has data that is as clean as it should be. Especially those things required for decision making. Most people concentrate on data quality on numbers, making sure the amounts are right, make sure the accounts are right those kinds of things because you need those to print statements to do business. But for decision making I need different things. I need to know the gender of the customer to understand my market place. I don't need to know the gender of the customer to process a transaction. So basically nobody cares about it, and they put any garbage they want in there and that becomes a problem later on. Absolutely do not assume the data is clean, assume the opposite, assume the data is dirty and help prove it otherwise.

17.4 ETL vs. ELT

There are two fundamental approaches to data acquisition:

ETL: Extract, Transform, Load in which data transformation takes place on a separate transformation server.

ELT: Extract, Load, Transform in which data transformation takes place on the data warehouse server.

Combination of both is also possible

You want to have a data flow driven architecture that you understand these points i.e. what data is required and how does it flow in the system driven by the meta data. Once we have those data flows then we can parallelize. You leverage pre-packaged tool for the transformation steps whenever possible. But then again have to look at the practical realities of the market place. That's why I say whenever possible. Whenever possible partially means whenever economically feasible.

In the architecture, ETL presents itself very suitable for data parallelism. Because I have got tons of data and I want to apply operations consistently across all of that data. So data parallelism is almost always used and you may use pipeline parallelism assuming I do not need too many sources so on. The difficulty of doing this parallelization by hand is very high. Because if you want to write a parallel program say a parallel C program, like parallel pro or something like that, it is very difficult. Again it is not just writing a parallel program, but you also need to make sure it works with check-points restarts and/or error conditions and all those things. I suspect that in your environment, but this is true for NADRA and PTML, that use the database to do the parallelization. So a different kind of approach, what if you are not willing to buy a tool that costs US$ 200,000?

This is a different kind of approach called ELT which is Extract Load Transform. We extract, we load into the database and then we transform in the parallel database. Then we get all the parallelism for free, because you already have a parallel database. You don't have to buy a separate tool in order to get the parallelization.

Masters

Lecture 18: ETL Detail: Data Extraction & Transformation

imp

Learning Goals

- Extracting of Data
- CDC in Modern Systems
- CDC in Legacy Systems
- CDC Advantages
- Major Transformation Types

Oftentimes, data warehousing involves the extraction and transportation of relational data from one or more source databases, into the data warehouse for analysis. Change Data Capture (CDC) quickly identifies and processes only the data that has changed, not entire tables, and makes the change data available for further use.

Without Change Data Capture, database extraction is a cumbersome process in which you move the entire contents of tables into flat files, and then load the files into the data warehouse. This ad hoc approach is expensive in a number of ways.

18.1 Extracting Changed Data

Incremental data extraction
Incremental data extraction i.e. what has changed, say during last 24 hrs if considering nightly extraction.

Efficient when changes can be identified
This is efficient, when the small changed data can be identified efficiently.

Identification could be costly
Unfortunately, for many source systems, identifying the recently modified data may be difficult or effect operation of the source system.

Very challenging
Change Data Capture is therefore, typically the most challenging technical issue in data extraction.

Two CDC sources

- Modern systems
- Legacy systems

If a data warehouse extracts data from an operational system on a nightly basis, then the data warehouse requires only the data that has changed since the last extraction (that is, the data that has been modified in the past 24 hours).

The changed data resulting from INSERT, UPDATE, and DELETE operations made to user tables is recorded. For some environments, the change data is then stored in a database object called a *change table*, and the changed data is made available to applications in a controlled way.

When it is possible to efficiently identify and extract only the most recently changed data, the extraction process (as well as all downstream operations in the ETL process) can be much more efficient, because it must extract a much smaller volume of data.

Unfortunately, for many source systems, identifying the recently modified data may be difficult or intrusive to the operation of the system. Change Data Capture is typically the most challenging technical issue in data extraction.

18.2 CDC in Modern Systems

- Time Stamps
  - Works if timestamp column present
  - If column not present, add column
  - May not be possible to modify table, so add triggers

- Triggers
  - Create trigger for each source table
  - Following each DML operation trigger performs updates
  - Record DML operations in a log

- Partitioning
  - Table range partitioned, say along date key
  - Easy to identify new data, say last week's data

Several techniques are used for implementing changed data capture in currently used state of the art source systems:

- Timestamps
- Partitioning
- Triggers

These techniques are based upon the characteristics of the source systems, or may require modifications to the source systems. Thus, each of these techniques must be carefully evaluated by the owners of the source system prior to implementation.

Timestamps

The tables in some operational systems have timestamp columns. The timestamp specifies the time and date that a given row was last modified. If the tables in an operational system have columns containing timestamps, then the latest data can easily be identified using the timestamp columns. If the timestamp information is not available in an operational source system, you will not always be able to modify the system to include timestamps. Such modification would require, first, modifying the operational system's tables to include a new timestamp column and then creating a trigger to update the timestamp column following every operation that modifies a given row.

Triggers

Triggers can be created in operational systems to keep track of recently updated records. They can then be used in conjunction with timestamp columns to identify the exact time and date when a given row was last modified. You do this by creating a trigger on each source table that requires change data capture. Following each DML statement that is executed on the source table, this trigger updates the timestamp column with the current

time. Thus, the timestamp column provides the exact time and date when a given row was last modified. Then, whenever any modifications are made to the source table, a record is inserted into the materialized view log indicating which rows were modified

Partitioning

Some source systems might use range partitioning, such that the source tables are partitioned along a date key, which allows for easy identification of new data. For example, if you are extracting from an orders table, and the orders table is partitioned by week, then it is easy to identify the current week's data.

18.3 CDC in Legacy Systems

- Changes recorded in tapes Changes occurred in legacy transaction processing are recorded on the log or journal tapes.

- Changes read and removed form tapes Log or journal tape is read and the update/transaction changes are stripped off for movement into the data warehouse.

- Problems with reading a log/journal tape are many:

  - Contains lot of extraneous data
  - Format is often arcane
  - Often contains addresses instead of data values and keys
  - Sequencing of data in the log tape often has deep and complex implications
  - Log tape varies widely from one DBMS to another.

The log or journal tape is used for backup and recovery purposes and is a normal part of the legacy DBMS infrastructure. CDC uses the legacy log tape as a means of capturing and identifying the update/ transaction changes that have occurred throughout the online day. CDC requires that the log or journal tape be read and the update/transaction changes be stripped out for the purpose of preparing the data for movement into the data warehouse. But reading a log/journal tape is no small matter. There are many obstacles that are in the way:

- the log tape contains lot of extraneous data,
- the log tape format is often arcane,
- the log tape often contains addresses instead of data values and keys,
- the sequencing of data in the log tape often has deep and complex implications,
- the log tape reflects the idiosyncrasies of the DBMS and varies widely from one DBMS to another, etc.

18.4 CDC Advantages

Advantages

1) Immediate.
2) No loss of history
3) Flat files NOT required

} Modern Systems

4) No incremental on-line I/O required for log tape

5) The log tape captures all update processing

6) Log tape processing can be taken off-line.
7) No haste to make waste.

Some of the advantages of CDC are;

1. Database extraction from INSERT, UPDATE, and DELETE operations occurs immediately, at the same time the changes occur to the source tables.

2. In the absence of CDC database extraction is marginal at best for INSERT operations, and problematic for UPDATE and DELETE operations, because the data is no longer in the table.

3. Stages data directly to relational tables; there is no need to use flat files. In the absence of CDC, the entire contents of tables are moved into flat files.

4. Log tape processing requires no extra I/O in the middle of online processing, as does replication processing. (Strictly speaking, log tape processing does require I/O. But the I/O was going to be spent whether the log tape was used for data warehouse refreshment or not. Therefore the marginal amount of extra I/O is nil when it comes to using the log tape for data warehouse refreshment.)

5. Another advantage of CDC is that the log tape captures all update processing. There is no need to go back and redefine parameters when there is a change made either to the data warehouse or to the legacy systems environment. The log tape is as basic and as stable as you can get.

6. The sixth and perhaps the most important reason for using the log tape is that the log tape processing can be taken off line. There is no need for any demands to be made against the native legacy DBMS when the log tape is used.

7. Finally data is able to be integrated and transformed "in-flight". Once the update/transaction data has been pulled from the log tape, the DWH is free to re-sequence, reformat, convert, merge, summarize, etc. There is no unnecessary haste or rush to get the data into the data warehouse with log tape processing. Therefore there is the opportunity to properly integrate the update/ transaction data before passing the data into the data warehouse.

Stated differently, unlike the data warehouse that results from raw transactions being stacked into a data warehouse from replication, with CDC the update/transaction data is able to be properly integrated before passing into the data warehouse.

18.5 Major Transformation Types

- Format revision
- Decoding of fields
- Calculated and derived values

- Splitting of single fields
- Merging of information
- Character set conversion
- Unit of measurement conversion
- Date/Time conversion
- Summarization
- Key restructuring
- Duplication

Irrespective of the variety and complexity of the source operational systems, and regardless of the extent of your data warehouse, you will find that most of your data transformation functions break down into a few basic tasks. We have already discussed them. When we consider a particular set of extracted data structures, you will find that the transformation functions you need to perform on this set may done by doing a combination of the basic tasks discussed. Now let us consider specific types of transformation functions. These are the most common transformation types:

Major Transformation Types

- Format revision
- Decoding of fields
- Calculated and derived values Covered in De-Norm
- Splitting of single fields Covered in issues

Format Revisions: You will come across these quite often. These revisions include changes to the data types and lengths of individual fields 12 digit new NID. In your source systems, product package types may be indicated by codes and names in which the fields are numeric and text data types. Again, the lengths of the package types may vary among the different source systems. It is wise to standardize and change the data type to text to provide values meaningful to the users.

Decoding of Fields: This is also a common type of data transformation. When you deal with multiple source systems, you are bound to have the same data items described by a plethora of field values. The classic example is the coding for gender, with one source system using 1 and 2 for male and female and another system using M and F. Also, many legacy systems are notorious for using cryptic codes to represent business values. What do the codes AC, IN, RE, and SU mean in a customer file? You need to decode all such cryptic codes and change these into values that make sense to the users. Change the codes to Active, Inactive, Regular, and Suspended.

Calculated and Derived Values: What if you want to keep profit margin along with sales and cost amounts in your data warehouse tables? (Briefly touched in de-normalization). The extracted data from the sales system contains sales amounts, sales units, and operating cost estimates by product. You will have to calculate the total cost and the profit margin before data can be stored in the data warehouse. Age and GPA are examples of derived fields.

Splitting of Single Fields: Earlier legacy systems stored names and addresses of customers and employees in large text fields. The first name, middle name, and family name were stored as a large text in a single field. Similarly, some earlier systems stored road/street, sector, and city data together in a single field. You need to store individual components of names and addresses in separate fields in your data warehouse for two reasons. First, you may improve the operating performance by indexing on individual components. Second, your users may need to perform analysis by using individual components such as road, sector, city etc.

Major Transformation Types Imp

- Merging of information
  Not really means combining columns to create one column.
  Info for product coming from different sources merging it into single entity.

- Character set conversion
  For PC architecture converting legacy EBCIDIC to ASCII

- Unit of measurement conversion
  For companies with global branches Km vs. mile or lb vs. Kg

- Date/Time conversion
  November 14, 2005 as 11/14/2005 in US and 14/11/2005 in the British format. This date may be standardized to be written as 14 NOV 2005.

Merging of Information: This is not quite the opposite of splitting of single fields. This type of data transformation does not literally mean the merging of several fields to create a single field of data. For example, information about a product may come from different data sources. The product Code and description may come from one data source. The relevant package types may be found in another data source. The cost data may be from yet another source. In this case, merging of information denotes the combination of the product code, description, package types, and cost into a single entity.

Character Set Conversion: This type of data transformation relates to the conversion of character sets to an agreed standard character set for textual data in the data warehouse. If you have mainframe legacy systems as source systems, the source data from these systems will be in EBCDIC characters. If PC-based architecture is the choice for your data warehouse, then you must convert the mainframe EBCDIC format to the ASCII format. When your source data is on other types of hardware and operating systems, you are faced with similar character set conversions.

Conversion of Units of Measurements: Many companies today have global branches. Measurements in many European countries are in metric units. If your company has overseas operations, you may have to convert the metrics so that the numbers may all be in one standard unit of measurement.

Date/Time Conversion: This type relates to representation of date and time in standard formats. For example, the American and the British date formats may be standardized to an international format. The date of November 14, 2005 is written as 11/14/2005 in the

U.S. format and as 14/11/2005 in the British format. This date may be standardized to be written as 14 NOV 2005.

Major Transformation Types

- Aggregation & Summarization

Adding like values

- How they are different?

Summarization with calculation across business dimension is aggregation. Example Monthly compensation = monthly sale + bonus

- Why both are required?
  - Grain mismatch (don't require, don't have space)
  - Data Marts requiring low detail
  - Detail losing its utility

Data stored in data warehouses is usually summarized and aggregated at some level because of the vast size of most data warehouses coupled with the analytical processing that occurs on the warehouse data. Although summarization and aggregation are sometimes used interchangeably, you will find a subtle difference between the two.

Summarization is the addition of like values along one or more business dimensions. An example of summarization is adding up detail revenue values by day to arrive at weekly totals (or by week to arrive at monthly totals, by month to arrive at quarterly totals, and so on).

Aggregation refers to a summarization coupled with a calculation across different business elements. An example of aggregation is the addition of bimonthly salary to monthly commission and bonus to arrive at monthly employee compensation values.

Depending on the data requirements of the warehouse, both summarization and aggregation can be deployed during data transformation. Summarization and aggregation are typically used for the following reasons:

- They are required when the lowest level of detail stored in the data warehouse is at a higher level than the detail arriving from the source. This situation occurs when data warehouse queries do not require the lowest level of detail or sometimes when sufficient disk space is not available to store all the data for the time frame required by the data warehouse.

- They can be used to populate data marts from the data warehouse where the data mart does not require the same level of detail as is stored in the warehouse.

- They can be used to roll up detail values when the detail is removed from the warehouse because it is not being used or because it has aged past its useful life in the data warehouse.

Major Transformation Types

- **Key restructuring (inherent meaning at source)**

92	42	4979	234
Country_Code	**City_Code**	**Post_Code**	**Product_Code**

- **i.e. 92424979234 changed to 12345678**

- **Removing duplication**
  - **Incorrect or missing value**
  - **Inconsistent naming convention ONE vs. 1**
  - **Incomplete information**
  - **Physically moved, but address not changed**
  - **Misspelling or falsification of names**

Table-18.1: Major Transformation Types

Key Restructuring: While extracting data from your input sources, look at the primary keys of the extracted records. You will have to come up with keys for the fact and dimension tables based on the keys in the extracted records. In the example shown in the Table 18.1, the product code in this organization is structured to have inherent meaning. If you use this product code as the primary key, there would be problems. If the product is moved to another warehouse, the warehouse part of the product key will have to be changed. This is a typical problem with legacy systems. When choosing keys for your data warehouse database tables, avoid such keys with built-in meanings. Transform such keys into generic keys generated by the system itself. This is called key restructuring.

Deduplication: In a normal client database some clients may be represented by several records for various reasons (i) incorrect or missing data values because of data entry errors (ii) inconsistent value naming conventions because of different entry formats and use of abbreviations such as ONE vs. 1 (iii) incomplete information because data is not captured or available (iv) clients do not notify change of address and (v) clients misspell their names or give false address or incorrect information about themselves. As a result we encounter situations where several records may refer to the same real world entity while not being syntactically equivalent.

In your data warehouse, you want to keep a single record for one customer and link all the duplicates in the source systems to this single record. This process is called deduplication of the customer file. Employee files and, sometimes, product master files have this kind of duplication problem.

Data content defects

- Domain value redundancy
- Non-standard data formats
- Non-atomic data values
- Multipurpose data fields
- Embedded meanings
- Inconsistent data values
- Data quality contamination

These are just some of the data content defects, there are and can be many more. We will discuss them in detail in the next lecture.

Masters

Subscribe to masters