

PARALLEL AND DISTRIBUTED DATABASES

Unit Structure:

- 1.1 Architectures for Parallel Databases
- 1.2 Parallel Query Evaluation
 - 1.2.1 Data Partitioning
 - 1.2.2 Parallelizing Sequential Operator Evaluation Code
- 1.3 Parallelizing Individual Operations
 - 1.3.1 Bulk Loading and Scanning
 - 1.3.2 Sorting
 - 1.3.3 Joins
- 1.4 Distributed Databases
 - 1.4.1 Introduction to DBMS
 - 1.4.2 Architecture of DDBs
- 1.5 Storing data in DDBs
 - 1.5.1 Fragmentation
 - 1.5.2 Replication
 - 1.5.3 Distributed catalog management
 - 1.5.4 Distributed query processing
- 1.6 Distributed concurrency control and recovery
 - 1.6.1 Concurrency Control and Recovery in Distributed Databases
 - 1.6.2 Lock management can be distributed across sites in many ways
 - 1.6.3 Distributed Deadlock
 - 1.6.4 Distributed Recovery

A **parallel database system** is one that seeks to improve performance through parallel implementation of various operations such as loading data, building indexes, and evaluating queries.

In a **distributed database system**, data is physically stored across several sites, and each site is typically managed by a DBMS that is capable of running independently of the other sites.

1.1 ARCHITECTURES FOR PARALLEL DATABASES

Three main architectures are proposed for building parallel databases:

1. Shared - memory system, where multiple CPUs are attached to an interconnection network and can access a common region of main memory.
2. Shared - disk system, where each CPU has a private memory and direct access to all disks through an interconnection network.
3. Shared - nothing system, where each CPU has local main memory and disk space, but no two CPUs can access the same storage area; all communication between CPUs is through a network connection.

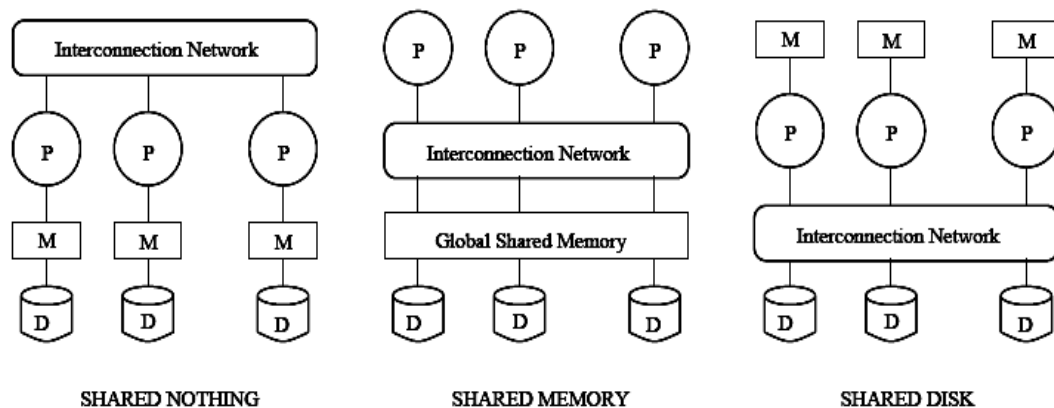


Fig 1.1 Architectures for Parallel Databases

- Scaling the system is an issue with shared memory and shared disk architectures because as more CPUs are added, existing CPUs are slowed down because of the increased contention for memory accesses and network bandwidth.
- **The Shared Nothing Architecture has shown:**
 - a) Linear Speed Up: the time taken to execute operations decreases in proportion to the increase in the number of CPU's and disks
 - b) Linear Scale Up: the performance is sustained if the number of CPU's and disks are increased in proportion to the amount of data.

1.2. PARALLEL QUERY EVALUATION

- Parallel evaluation of a relational query in a DBMS with a shared-nothing architecture is discussed. Parallel execution of a single query has been emphasized.
- A relational query execution plan is a graph of relational algebra operators and the operators in a graph can be executed in parallel. If an operator consumes the output of a second operator, we have pipelined parallelism.
- Each individual operator can also be executed in parallel by partitioning the input data and then working on each partition in parallel and then combining the result of each partition. This approach is called Data Partitioned parallel Evaluation.

1.2.1 Data Partitioning:

Here large datasets are partitioned horizontally across several disk, this enables us to exploit the I/O bandwidth of the disks by reading and writing them in parallel. This can be done in the following ways:

- a. Round Robin Partitioning
 - b. Hash Partitioning
 - c. Range Partitioning
- a. **Round Robin Partitioning** : If there are n processors, the i^{th} tuple is assigned to processor $i \bmod n$
 - b. **Hash Partitioning** : A hash function is applied to (selected fields of) a tuple to determine its processor.

Hash partitioning has the additional virtue that it keeps data evenly distributed even if the data grows and shrinks over time.

- c. **Range Partitioning** : Tuples are sorted (conceptually), and n ranges are chosen for the sort key values so that each range contains roughly the same number of tuples; tuples in range i are assigned to processor i .

Range partitioning can lead to data skew; that is, partitions with widely varying numbers of tuples across partitions or disks. Skew causes processors dealing with large partitions to become performance bottlenecks.

1.2.2 Parallelizing Sequential Operator Evaluation Code:

Input data streams are divided into parallel data streams. The output of these streams are merged as needed to provide as

inputs for a relational operator, and the output may again be split as needed to parallelize subsequent processing.

1.3. PARALLELIZING INDIVIDUAL OPERATIONS

Various operations can be implemented in parallel in a sharednothing architecture.

1.3.1 Bulk Loading and Scanning:

- Pages can be read in parallel while scanning a relation and the retrieved tuples can then be merged, if the relation is partitioned across several disks.
- If a relation has associated indexes, any sorting of data entries required for building the indexes during bulk loading can also be done in parallel.

1.3.2 Sorting:

- Sorting could be done by redistributing all tuples in the relation using range partitioning.
- Ex. Sorting a collection of employee tuples by salary whose values are in a certain range.
- For N processors each processor gets the tuples which lie in range assigned to it. Like processor 1 contains all tuples in range 10 to 20 and so on.
- Each processor has a sorted version of the tuples which can then be combined by traversing and collecting the tuples in the order on the processors (according to the range assigned)
- The problem with range partitioning is data skew which limits the scalability of the parallel sort. One good approach to range partitioning is to obtain a sample of the entire relation by taking samples at each processor that initially contains part of the relation. The (relatively small) sample is sorted and used to identify ranges with equal numbers of tuples. This set of range values, called a splitting vector, is then distributed to all processors and used to range partition the entire relation.

1.3.3 Joins:

- Here we consider how the join operation can be parallelized
- Consider 2 relations A and B to be joined using the age attribute. A and B are initially distributed across several disks in a way that is not useful for join operation

- So we have to decompose the join into a collection of k smaller joins by partitioning both A and B into a collection of k logical partitions.
- If same partitioning function is used for both A and B then the union of k smaller joins will compute to the join of A and B .

1.4 DISTRIBUTED DATABASES

- The abstract idea of a distributed database is that the data should be physically stored at different locations but its distribution and access should be transparent to the user.

1.4.1 Introduction to DBMS:

A Distributed Database should exhibit the following properties:

- 1) **Distributed Data Independence:** - The user should be able to access the database without having the need to know the location of the data.
- 2) **Distributed Transaction Atomicity:** - The concept of atomicity should be distributed for the operation taking place at the distributed sites.

- **Types of Distributed Databases are:-**

- a) Homogeneous Distributed Database is where the data stored across multiple sites is managed by same DBMS software at all the sites.
- b) Heterogeneous Distributed Database is where multiple sites which may be autonomous are under the control of different DBMS software.

1.4.2 Architecture of DDBs :

There are 3 architectures: -

1.4.2.1 Client-Server:

- A Client-Server system has one or more client processes and one or more server processes, and a client process can send a query to any one server process. Clients are responsible for user-interface issues, and servers manage data and execute transactions.
- Thus, a client process could run on a personal computer and send queries to a server running on a mainframe.

- **Advantages: -**

1. Simple to implement because of the centralized server and separation of functionality.
2. Expensive server machines are not underutilized with simple user interactions which are now pushed on to inexpensive client machines.
3. The users can have a familiar and friendly client side user interface rather than unfamiliar and unfriendly server interface

1.4.2.2 Collaborating Server:

- In the client sever architecture a single query cannot be split and executed across multiple servers because the client process would have to be quite complex and intelligent enough to break a query into sub queries to be executed at different sites and then place their results together making the client capabilities overlap with the server. This makes it hard to distinguish between the client and server
- In Collaborating Server **system**, we can have collection of database servers, each capable of running transactions against local data, which cooperatively execute transactions spanning multiple servers.
- When a server receives a query that requires access to data at other servers, it generates appropriate sub queries to be executed by other servers and puts the results together to compute answers to the original query.

1.4.2.3 Middleware:

- Middleware system is as special server, a layer of software that coordinates the execution of queries and transactions across one or more independent database servers.
- The Middleware architecture is designed to allow a single query to span multiple servers, without requiring all database servers to be capable of managing such multi site execution strategies. It is especially attractive when trying to integrate several legacy systems, whose basic capabilities cannot be extended.
- We need just one database server that is capable of managing queries and transactions spanning multiple servers; the remaining servers only need to handle local queries and transactions.

1.5 STORING DATA IN DDBS

Data storage involved 2 concepts

1. Fragmentation
2. Replication

1.5.1 Fragmentation:

- It is the process in which a relation is broken into smaller relations called fragments and possibly stored at different sites.
- It is of 2 types

1. **Horizontal Fragmentation** where the original relation is broken into a number of fragments, where each fragment is a subset of rows.

The union of the horizontal fragments should reproduce the original relation.

2. **Vertical Fragmentation** where the original relation is broken into a number of fragments, where each fragment consists of a subset of columns.

The system often assigns a unique tuple id to each tuple in the original relation so that the fragments when joined again should form a lossless join.

The collection of all vertical fragments should reproduce the original relation.

1.5.2 Replication:

- Replication occurs when we store more than one copy of a relation or its fragment at multiple sites.
- **Advantages:-**
 1. **Increased availability of data:** If a site that contains a replica goes down, we can find the same data at other sites. Similarly, if local copies of remote relations are available, we are less vulnerable to failure of communication links.
 2. **Faster query evaluation:** Queries can execute faster by using a local copy of a relation instead of going to a remote site.

1.5.3 Distributed catalog management :

Naming Object

- It's related to the unique identification of each fragment that has been either partitioned or replicated.

- This can be done by using a global name server that can assign globally unique names.
 - This can be implemented by using the following two fields:-
1. Local name field – locally assigned name by the site where the relation is created. Two objects at different sites can have same local names.
 2. Birth site field – indicates the site at which the relation is created and where information about its fragments and replicas is maintained.

Catalog Structure:

- A centralized system catalog is used to maintain the information about all the transactions in the distributed database but is vulnerable to the failure of the site containing the catalog.
- This could be avoided by maintaining a copy of the global system catalog but it involves broadcast of every change done to a local catalog to all its replicas.
- Another alternative is to maintain a local catalog at every site which keeps track of all the replicas of the relation.

Distributed Data Independence:

- It means that the user should be able to query the database without needing to specify the location of the fragments or replicas of a relation which has to be done by the DBMS
- Users can be enabled to access relations without considering how the relations are distributed as follows:
The *local name* of a relation in the system catalog is a combination of a *user name* and a user-defined *relation name*.
- When a query is fired the DBMS adds the user name to the relation name to get a local name, then adds the user's site-id as the (default) birth site to obtain a global relation name. By looking up the global relation name in the local catalog if it is cached there or in the catalog at the birth site the DBMS can locate replicas of the relation.

1.5.4 Distributed query processing:

- In a distributed system several factors complicate the query processing.
- One of the factors is cost of transferring the data over network.
- This data includes the intermediate files that are transferred to other sites for further processing or the final result files that may have to be transferred to the site where the query result is needed.
- Although these costs may not be very high if the sites are connected via a high local n/w but sometime they become quite significant in other types of network.
- Hence, DDBMS query optimization algorithms consider the goal of reducing the amount of data transfer as an optimization criterion in choosing a distributed query execution strategy.
- Consider an EMPLOYEE relation.
- The size of the employee relation is $100 * 10,000 = 10^6$ bytes
- The size of the department relation is $35 * 100 = 3500$ bytes

EMPLOYEE

Fname	Lname	<u>SSN</u>	Bdate	Add	Gender	Salary	Dnum
-------	-------	------------	-------	-----	--------	--------	------

- 10,000 records
- Each record is 100 bytes
- Fname field is 15 bytes long
- SSN field is 9 bytes long
- Lname field is 15 bytes long
- Dnum field is 4 bytes long

DEPARTMENT

Dname	Dnumber	MGRSSN	MgrStartDate
-------	---------	--------	--------------

- 100 records
- Each record is 35 bytes long
- Dnumber field is 4 bytes long
- Dname field is 10 bytes long
- MGRSSN field is 9 bytes long
- Now consider the following query:
 “For each employee, retrieve the employee name and the name of the department for which the employee works.”

- Using relational algebra this query can be expressed as

$$FNAME, LNAME, DNAME$$

$$(EMPLOYEE * DNO=DNUMBER DEPARTMENT)$$
- If we assume that every employee is related to a department then the result of this query will include 10,000 records.
- Now suppose that each record in the query result is 40 bytes long and the query is submitted at a distinct site which is the result site.
- Then there are 3 strategies for executing this distributed query:
 1. Transfer both the EMPLOYEE and the DEPARTMENT relations to the site 3 that is your result site and perform the join at that site. In this case a total of $1,000,000 + 3500 = 1,003,500$ bytes must be transferred.
 2. Transfer the EMPLOYEE relation to site 2 (site where u have Department relation) and send the result to site 3. the size of the query result is $40 * 10,000 = 400,000$ bytes so $400,000 + 1,000,000 = 1,400,000$ bytes must be transferred.
 3. Transfer the DEPARTEMNT relation to site 1 (site where u have Employee relation) and send the result to site 3. in this case $400,000 + 3500 = 403,500$ bytes must be transferred.

1.5.4.1 Nonjoin Queries in a Distributed DBMS:

- Consider the following two relations:
 Sailors (sid: integer, sname:string, rating: integer, age: real)
 Reserves (sid: integer, bid: integer, day: date, rname: string)
- Now consider the following query:

$$\begin{aligned} & \text{SELECT S.age} \\ & \text{FROM Sailors S} \\ & \text{WHERE S.rating} > 3 \text{ AND S.rating} < 7 \end{aligned}$$
- Now suppose that sailor relation is horizontally fragmented with all the tuples having a rating less than 5 at Shanghais and all the tuples having a rating greater than 5 at Tokyo.
- The DBMS will answer this query by evaluating it both sites and then taking the union of the answer.

1.5.4.2 Joins in a Distributed DBMS:

- Joins of a relation at different sites can be very expensive so now we will consider the evaluation option that must be considered in a distributed environment.

- Suppose that Sailors relation is stored at London and Reserves relation is stored at Paris. Hence we will consider the following strategies for computing the joins for Sailors and Reserves.
- In the next example the time taken to read one page from disk (or to write one page to disk) is denoted as td and the time taken to ship one page (from any site to another site) as ts .

2.5.1 Fetch as needed:

- We can do a page oriented nested loops joins in London with Sailors as the outer join and for each Sailors page we will fetch all Reserves pages form Paris.
- If we cache the fetched Reserves pages in London until the join is complete , pages are fetched only once, but lets assume that Reserves pages are not cached, just to see how bad result we can get:
- To scan Sailors the cost is $500td$ for each Sailors page, plus the cost of scanning and shipping all of Reserves is $1000(td+ts)$. Therefore the total cost is $500td+500000(td+ts)$.
- In addition if the query was not submitted at the London site then we must add the cost of shipping the result to the query site and this cost depends on the size of the result.
- Because sid a key for the Sailors. So, the number of tuples in the result is 100,000 (which is the number of tuples in Reserves) and each tuple is $40+50=90$ bytes long.
- Thus (4000 is the size of the result) $4000/90=44$ result tuples fit on a page and the result size is $100000/44=2273$ pages.

1.5.4.3 Ship to one site:

- There are three possibilities to compute the result at one site:
 - Ship the Sailors from London to Paris and carry out the join.
 - Ship the Reserves form Paris to London and carry out the join.
 - Ship both i.e. Sailors and Reserves to the site where the query was posed and compute the join.
- And the cost will be:
 - The cost of scanning and shipping Sailors form London to Paris and doing the join at Paris is $500(2td+ ts) + 4500td$.
 - The cost shipping Reserves form Paris to London and then doing the join at London is $1000 (2td + ts) + 4500td$.

2.5.2 Semi joins and bloomjoins:

- Consider the strategy of shipping Reserves from Paris to London and computing the joins at London.
- It may happen that some tuples in Reserves do not join with any tuple in the Sailors, so we could somehow identify the tuples that are guaranteed not to join with any Sailors tuples and we could avoid shipping them.

- **Semijoins:**

- The basic idea of Semijoins can be proceed in three steps:
 - 1) At London compute the projection of Sailors onto the join columns, and ship this projection to Paris.
 - 2) At Paris, compute the natural join of the projection received from the first site with the Reserves relation. The result of this join is called the reduction of Reserves with respect to Sailors because only those Reserves tuples in the reduction will join with tuples in the Sailors relation. Therefore, ship the reduction of Reserves to London, rather than the entire Reserves relation.
 - 3) At London, compute the join of the reduction of Reserves with Sailors.

- **Computing the cost of Sailors and Reserves using Semijoin:**

- Assume we have a projection based on first scanning Sailors and creating a temporary relation with tuples that have only an *sid* field, then sorting the temporary and scanning the sorted temporary to eliminate duplicates.
- If we assume that the size of the *sid* field is 10 bytes, then the cost of projection is 500td for scanning Sailors, plus 100td for creating the temporary, plus 400td for sorting it, plus 100td for the final scan, plus 100td for writing the result into another temporary relation, that is total is 1200td.

$$500td + 100td + 400td + 100td + 100td = 1200td.$$
- The cost of computing the projection and shipping them it to Paris is $1200td + 100ts$.
- The cost of computing the reduction of Reserves is $3 * (100+1000)=3300td$.

- **But what is the size of Reduction?**

- If every sailor holds at least one reservation then the reduction includes every tuple of Reserves and the effort invested in shipping the projection and reducing Reserves is a total waste.

- So because of this observation we can say that Semijoin is especially useful in conjunction with a selection on one of the relations.
- For example if we want to compute the join of Sailors tuples with a rating > 8 of the Reserves relation, then the size of the projection on *sid* for tuples that satisfy the selection would be just 20% of the original projection that is 20 pages.
- **Bloomjoin:**
 - Bloomjoin is quit similar to semijoins.
 - The steps of Bloomjoins are:

Step 1:

The main difference is that a bit-vector is shipped in first step, instead of the projection of Sailors.

A bit-vector (some chosen tuple) of size k is computed by hashing each tuple of Sailors into the range 0 to $k-1$ and setting bit l to 1 if some tuple hashes to l and 0 otherwise.

Step 2:

The reduction of Reserves is computed by hashing each tuple of Reserves (using the *sid* field) into the range 0 to $k-1$, using the same hash function which is used to construct the bit-vector and discard the tuples whose hash values corresponds to 0 bit.

Because no Sailors tuples hash to such an i and no Sailors tuples can join with any Reserves tuple that is not in the reduction.

- Thus the cost of shipping a bit-vector and reducing Reserves using the vector are less than the corresponding costs is Semijoins.

1.5.4.4 Cost-Based Query Optimization:

A query involves several operations and optimizing a query in a distributed database poses some challenges:

- Communication cost must be considered. If we have several copies of a real time then we will have to decide which copy to use.
- If the individual sites are run under the control of different DBMS then the autonomy of each site must be respected while doing global query planning.

1.6 DISTRIBUTED CONCURRENCY CONTROL AND RECOVERY

The main issues with respect to the Distributed transaction are:

- **Distributed Concurrency Control**

- How can deadlocks be detected in a distributed database?
- How can locks for objects stored across several sites be managed?

- **Distributed Recovery**

- When a transaction commits, all its actions across all the sites at which it executes must persist.
- When a transaction aborts none of its actions must be allowed to persist.

1.6.1 Concurrency Control and Recovery in Distributed Databases:

For currency control and recovery purposes, numerous problems arise in a distributed DBMS environment that is not encountered in a centralized DBMS environment.

These includes the following:

Dealing with multiple copies of the data items:

The concurrency control method is responsible for maintaining consistency among these copies. The recovery method is responsible for making a copy consistent with other copies if the site on which he copy is stored fails and recovers later.

Failure of individual sites:

The DBMS should continue to operate with its running sites, if possible when one or the more individual site fall. When a site recovers its local database must be brought up to date with the rest of the sites before it rejoins the system.

Failure of communication links:

The system must be able to deal with failure of one or more of the communication links that connect the sites. An extreme case of this problem is hat network partitioning may occur. This breaks up the sites into two or more partitions where the sites within each partition can communicate only with one another and not with sites in other partitions.

Distributed Commit:

Problems can arise with committing a transactions that is accessing database stored on multiple sites if some sites fail during the commit process. The two-phase commit protocol is often used to deal with this problem.

Distributed Deadlock:

Deadlock may occur among several sites so techniques for dealing with deadlocks must be extended to take this into account.

1.6.2 Lock management can be distributed across sites in many ways:

- **Centralized:** A single site is in charge of handling lock and unlock requests for all objects.
- **Primary copy:** One copy of each object is designates as the primary copy. All requests to lock or unlock a copy of these objects are handled by the lock manager at the site where the primary copy is stored, regardless of where the copy itself is stored.
- **Fully Distributed:** Request to lock or unlock a copy of an object stored at a site are handled by the lock manager at the site where the copy is stored.

Conclusion:

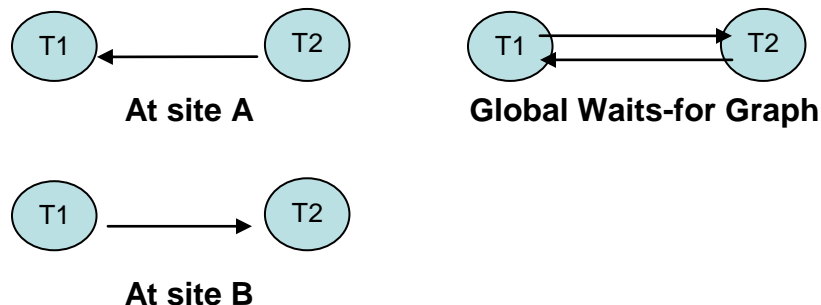
- The Centralized scheme is vulnerable to failure of the single site that controls locking.
- The Primary copy scheme avoids this problem, but in general reading an object requires communication with two sites:
 - The site where the primary copy is resides.
 - The site where the copy to be read resides.
- This problem is avoided in fully distributed scheme because in this scheme the locking is done at the site where the copy to be read resides.

1.6.3 Distributed Deadlock

- One issue that requires special attention when using either primary copy or fully distributed locking is deadlocking detection
- Each site maintains a local waits-for graph and a cycle in a local graph indicates a deadlock.

For example:

- Suppose that we have two sites A and B, both contain copies of objects O1 and O2 and that the read-any write-all technique is used.
- T1 which wants to read O1 and write O2 obtains an S lock on O1 and X lock on O2 at site A, and request for an X lock on O2 at site B.
- T2 which wants to read O2 and write O1 meanwhile obtains an S lock on O2 and an X lock on O1 at site B then request an X lock on O1 at site A.
- As shown in the following figure T2 is waiting for T1 at site A and T1 is waiting for T2 at site B thus we have a Deadlock.



To detect such deadlocks, a distributed deadlock detection algorithm must be used and we have three types of algorithms:

1. Centralized Algorithm:

- It consists of periodically sending all local waits-for graphs to some one site that is responsible for global deadlock detection.
- At this site, the global waits-for graph is generated by combining all local graphs and in the graph the set of nodes is the union of nodes in the local graphs and there is an edge from one node to another if there is such an edge in any of the local graphs.

2. Hierarchical Algorithm:

- This algorithm groups the sites into hierarchies and the sites might be grouped by states, then by country and finally into single group that contain all sites.
- Every node in this hierarchy constructs a waits-for graph that reveals deadlocks involving only sites contained in (the sub tree rooted at) this node.

- Thus, all sites periodically (e.g., every 10 seconds) send their local waits-for graph to the site constructing the waits-for graph for their country.
- The sites constructing waits-for graph at the country level periodically (e.g., every 10 minutes) send the country waits-for graph to site constructing the global waits-for graph.

3. Simple Algorithm:

- If a transaction waits longer than some chosen time-out interval, it is aborted.
- Although this algorithm causes many unnecessary restart but the overhead of the deadlock detection is low.

1.6.4 Distributed Recovery:

Recovery in a distributed DBMS is more complicated than in a centralized DBMS for the following reasons:

- New kinds of failure can arise: failure of communication links and failure of remote site at which a sub transaction is executing.
- Either all sub transactions of a given transaction must commit or none must commit and this property must be guaranteed despite any combination of site and link failures. This guarantee is achieved using a commit protocol.

Normal execution and Commit Protocols:

- During normal execution each site maintains a log and the actions of a sub transaction are logged at the site where it executes.
- The regular logging activity is carried out which means a commit protocol is followed to ensure that all sub transactions of a given transaction either commit or abort uniformly.
- The transaction manager at the site where the transaction originated is called the *Coordinator* for the transaction and the transaction managers where its sub transactions execute are called *Subordinates*.

Two Phase Commit Protocol:

- When the user decides to commit the transaction and the commit command is sent to the coordinator for the transaction.

This initiates the 2PC protocol:

- The coordinator sends a Prepare message to each subordinate.
- When a subordinate receive a Prepare message, it then decides whether to abort or commit its sub transaction. it force-writes an abort or prepares a log record and then sends a NO or Yes message to the coordinator.
- Here we can have two conditions:
 - If the coordinator receives Yes message from all subordinates. It force-writes a commit log record and then sends a *commit* message to all the subordinates.
 - If it receives even one No message or No response from some coordinates for a specified time-out period then it will force-write an abort log record and then sends an *abort* message to all subordinate.
- Here again we can have two conditions:
 - When a subordinate receives an abort message, it force-writes an abort log record sends an *ack* message to coordinator and aborts the sub transaction.
 - When a subordinates receives a commit message, it force-writes a commit log record and sends an ack message to the coordinator and commits the sub transaction.



Unit Structure:

- 2.1 Introduction
- 2.2 Data Marts
- 2.3 Extraction
- 2.4 Transformation
- 2.5 Loading
- 2.6 Metadata
- 2.7 Data Warehousing and ERP
- 2.8 Data Warehousing and CRM
- 2.9 Data Warehousing and KM

2.1 INTRODUCTION:

A **data warehouse** is a repository of an organization's electronically stored data. Data warehouses are designed to facilitate reporting and analysis.

This definition of the data warehouse focuses on data storage. However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system.

A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process"

Subject Oriented:

Data that gives information about a particular subject instead of about a company's ongoing operations.

Integrated:

Data that is gathered into the data warehouse from a variety of sources and merged into a coherent whole.

Time-variant:

All data in the data warehouse is identified with a particular time period.

Non-volatile:

Data is stable in a data warehouse. More data is added but data is never removed. This enables management to gain a consistent picture of the business.

2.2 DATA MARTS:

Data marts are analytical data stores designed to focus on specific business functions for a specific community within an organization. Data marts are often derived from subsets of data in a data warehouse.

Reasons for creating a data mart

- Easy access to frequently needed data
- Creates collective view by a group of users
- Improves end-user response time
- Ease of creation
- Lower cost than implementing a full Data warehouse
- Potential users are more clearly defined than in a full Data warehouse

Data mart issues:

- **data mart functionality**→the capabilities of data marts have increased with the growth in their popularity.
- **data mart size**→the performance deteriorates as data marts grow in size, so need to reduce the size of data marts to gain improvements in performance.
- **data mart load performance**→two critical components: end-user response time and data loading performance→to increment DB updating so that only cells affected by the change are updated and not the entire MDDB structure.
- **users' access to data in multiple marts**→one approach is to replicate data between different data marts or, alternatively, build **virtual data mart**→*it is views of several physical data marts or the corporate data warehouse tailored to meet the requirements of specific groups of users.*
- **data mart internet/intranet access**→it's products sit between a web server and the data analysis product. Internet/intranet offers users low-cost access to data marts and the data WH using web browsers.
- **data mart administration**→organization can not easily perform administration of multiple data marts, giving rise to issues such as data mart versioning, data and meta-data consistency and integrity, enterprise-wide security, and performance tuning. Data mart administrative tools are commercially available.

- **data mart installation**→data marts are becoming increasingly complex to build. Vendors are offering products referred to as "data mart in a box" that provide a low-cost source of data mart tools.

What is the difference between Data Mart and Data warehousing?

Firstly, Data mart represents the *programs, data, software and hardware* of a **specific department**. For example, there is separate data mart for finance, production, marketing and sales department.

None of the data mart resembles with any other data mart. However, it is possible to coordinate the data of various departments. Data mart of a specific department is completely focused on individual needs, requirements and desires. Data in data mart is highly indexed but is not suitable to support huge data as it is designed for a particular department.

Whenever the data mart database is to be designed, the requirements of all users in the department are gathered. Data marts are basically of two types, *independent data mart and dependent data mart*. *Data warehouse is used as a source by a dependent data mart*. Dependent data mart is structurally and architecturally strong. Independent data mart on the other hand is not structurally and architecturally strong. Moreover, they are not useful for organization until you create multiple independent data marts.

Data warehousing is not limited to a department of office. It represents the database of a complete corporate organization. Subject areas of data warehousing includes all corporate subject areas of corporate data model. Data warehousing is neither bounded to have relations between subject areas of departments nor with subject areas of corporation. Detailed data is stored in the database of data warehousing unlike data mart which stores only aggregated or summarized data. Data in the data warehouse is indexed lightly as it has to manage large volume of data. It would be wise to say that there is very little difference in data structure and content of datamarts and datawarehouse.

Data warehouse is built iteratively as it is not possible to create the complete data warehouse and then implement it. Data warehouse is a vast concept that covers various aspects of a corporate world. In contrast, data mart can be easily and quickly designed and implemented for end user's use.

Many people think that they can create data marts for various departments and then integrate them into a data warehouse but this is not possible. Data mart only concentrates on individual department requirements and not on collective need of an entire organization. Moreover, various other factors such as granularity or level of data used, data structure etc. is also different in data mart and data warehousing. Data warehouses are an expensive deal and require a lot of time, investment, planning, managing volumes of data, data integration complexities etc. compared to a data mart.

Data mart is therefore useful for small organizations with very few departments or at places that requires data management for a specific business requirement whereas data warehousing is suitable to support an entire corporate environment.

2.3 EXTRACTION:

Extraction is the operation of extracting data from a source system for further use in a data warehouse environment. 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 data 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.

Since the data coming to the data warehouse may come from different source which commonly are of disparate systems resulting in different data formats, a data warehouse uses three processes to make use of the data. These processes are extraction, transformation and loading (ETL).

Data extraction is a process that involves retrieval of all format and types of data out of unstructured or badly structured data sources. These data will be further used for processing or data migration. Raw data is usually imported into an intermediate extracting system before being processed for data transformation where they will possibly be padded with meta data before being exported to another stage in the data warehouse work flow. The term data extraction is often applied when experimental data is first imported into a computer server from the primary sources such as recording or measuring devices.

During the process of data extraction in a data warehouse, data may be removed from the system source or a copy may be made with the original data being retained in the source system. It

is also practiced in some data extraction implementation to move historical data that accumulates in the operational system to a data warehouse in order to maintain performance and efficiency.

The data extraction process in general is performed within the source system itself. This is can be most appropriate if the extraction is added to a relational database.

2.4 TRANSFORMATION:

The transform stage applies a series of rules or functions to the extracted data from the source to derive the data for loading into the end target. Some data sources will require very little or even no manipulation of data. In other cases, one or more of the following transformation types may be required to meet the business and technical needs of the target database:

- Selecting only certain columns to load (or selecting null columns not to load). For example, if source data has three columns (also called attributes) say roll_no, age and salary then the extraction may take only roll_no and salary. Similarly, extraction mechanism may ignore all those records where salary is not present (salary = null).
- Translating coded values (e.g., if the source system stores 1 for male and 2 for female, but the warehouse stores M for male and F for female), this calls for automated data cleansing; no manual cleansing occurs during ETL
- Deriving a new calculated value (e.g., sale_amount = qty * unit_price)
- Filtering
- Sorting
- Joining data from multiple sources (e.g., lookup, merge)
- Aggregation (for example, rollup - summarizing multiple rows of data - total sales for each store, and for each region, etc.)
- Generating surrogate-key values
- Transposing or pivoting (turning multiple columns into multiple rows or vice versa)
- Splitting a column into multiple columns (e.g., putting a comma-separated list specified as a string in one column as individual values in different columns)
- Disaggregation of repeating columns into a separate detail table (e.g., moving a series of addresses in one record into single addresses in a set of records in a linked *address* table)

- Applying any form of simple or complex data validation. If validation fails, it may result in a full, partial or no rejection of the data, and thus none, some or all the data is handed over to the next step, depending on the rule design and exception handling. Many of the above transformations may result in exceptions, for example, when a code translation parses an unknown code in the extracted data.

2.5 LOADING:

After the data is extracted from appropriate sources and transformed in the required format, it needs to be loaded in the database. Data loading can be performed in one of the following phases :

Initial load:

Data warehouse tables are populated for the very first time in single run.

Incremental loads:

The changes that are made to the database are applied periodically to the database.

Full Refresh:

Contents of one or more tables are erased completely and they are reloaded.

The process of data loading takes up lot of time hence data warehouse needs to be offline during the process of loading.

Technically refresh is much simpler option than update.

Refresh involves periodic replacement of complete data. But if you have to run refresh jobs every day then it will become a tedious job. In this case its better to use data update.

If the number of records to be updated falls between 15 to 20% then use data update. If number of records to be updated is above 25% then use full-refresh.

2.6 METADATA:

Metadata in a DW contains the answers to questions about the data in DW. The metadata component serves as a directory of the contents of your DW. All the answers are kept in metadata repository.

Metadata repository is a general purpose information directory that classifies and manages Business and Technical metadata.

Types of metadata:-

- Operational metadata
- Extraction and transformation metadata
- End-user metadata

1) Operational Metadata:

The data elements selected for the DW are selected from different operational sources hence they have various field lengths and data types. While storing the records you may split records, combine parts of records from different source files etc. When you deliver such information to the end-user you must be able to tie that information back to the original source data.

Operational metadata contains all this information about the operations performed on the data.

2) Extraction and transformation metadata:

It contains data about various source systems from which the data is extracted for example, extraction frequencies, methods, business rules etc. It also contains information about all data transformations that take place in the staging area.

3) End-user metadata:

It is the navigation map of the DW. It enables the end-user to find the information from the DW.

The end-user metadata allows the user to use their own business terminology and look for the information.

Metadata is useful in DW firstly because it acts as glue that connects all parts of the DW and it also provides information about the contents and structures to the developers. It also helps the end-user to recognize the contents of the DW.

2.7 DATA WAREHOUSING AND ERP:

With introduction of “data warehousing”, companies have explored the ways in which they can capture, store and manipulate data for analysis and decision support. At the same time, many companies have been instituting enterprise resource planning (ERP) software to coordinate the common functions of an enterprise. The use of ERP helped to capture required data from the source systems and the existence of a central ERP database has created the opportunity to develop enterprise data warehouses for manipulating that data for analysis.

Since companies usually implement an ERP in addition to their current applications, the problem of data integration from the various sources of data to the data warehouse becomes an issue. Actually, the existence of multiple data sources is a problem with ERP implementation regardless of whether a company plans to develop a data warehouse; this issue must be addressed and resolved at the ERP project initiation phase to avoid serious complications from multiple sources of data for analysis and transaction activity. In data warehouse development, data is usually targeted from the most reliable or stable source system and moved into the data warehouse database as needed. Identification of the correct source system is essential for any data warehouse development, and is even more critical in a data warehouse that includes an ERP along with more traditional transaction systems. Integration of data from multiple sources (ERP database and others) requires rigorous attention to the metadata and business logic that populated the source data elements, so that the “right” source is chosen.

Another troubling issue with ERP data is the need for historical data within the enterprise’s data warehouse. Traditionally, the enterprise data warehouse needs historical data. And traditionally ERP technology does not store historical data, at least not to the extent that is needed in the enterprise data warehouse. When a large amount of historical data starts to stack up in the ERP environment, the data is archived to a remote storage facility.

2.8 DATA WAREHOUSING AND CRM:

If you wish to have your data warehouse more focused on customers, then you will have to make your data warehouse CRM-ready. Your data warehouse must contain all the information of all customers. This will increase data volume tremendously. CRM-ready data warehouse can be achieved by performing cleansing, eliminating duplicates, combining data elements etc.

For example, while taking customer name and address you will have to parse the unstructured data.

2.9 DATA WAREHOUSING AND KM:

Knowledge management means capturing, integrating, organizing and communicating knowledge accumulated by employees. All knowledge is stored in a knowledge repository or knowledge warehouse. A knowledge warehouse contains

unstructured data. Therefore KM should be integrated with Data warehouse in order to retrieve the information in the stored knowledge in a systematic way.



3

PLANNING AND PROJECT MANAGEMENT

Unit Structure:

- 3.1 How is it different?
- 3.2 Life-cycle approach
 - 3.2.1 The development phases
 - 3.2.2 Dimensional analysis
 - 3.2.3 Dimensional Nature of Business Data
- 3.3 Star schema
- 3.4 Snowflake scheme

3.1 HOW IS IT DIFFERENT?

A data warehouse project differs from transaction processing system.

It has Data Acquisition, Data Storage and Information Delivery components as its major functional parts

Data Warehouse Project Different From OLTP System Project Data Warehouse: Distinctive Features and Challenges for Project Management		
DATA ACQUISITION	DATA STORAGE	INFO. DELIVERY
Large number of sources	Storage of large data volumes	Several user types
Many disparate sources	Rapid growth	Queries stretched to limits
Different computing platforms	Need for parallel processing	Multiple query types
Outside sources	Data storage in staging area	Web-enabled
Huge initial load	Multiple index types	Multidimensional analysis
Ongoing data feeds	Several index files	OLAP functionality
Data replication considerations	Storage of newer data types	Metadata management
Difficult data integration	Archival of old data	Interfaces to DSS apps.
Complex data transformations	Compatibility with tools	Feed into Data Mining
Data cleansing	RDBMS & MDDBMS	Multi-vendor tools

Fig: How a data warehouse project is different

3.2 LIFE-CYCLE APPROACH

- A data warehouse project is complex in terms of tasks, technologies and member roles.
- The life cycle approach breaks down the project complexity and removes the confusions with regard to the responsibilities of the project team members.
- The Life cycle approach has to be adapted to the special needs of the data warehouse project.
- It should include iterative tasks going through cycles of refinement.
- Ex –
 1. Identification of data sources begins by listing all the source systems and source data structures
 2. Next Iteration – review the data elements with the users.
 3. Next Iteration – review the data elements with the database administrator
 4. Next Iteration - walking through the data elements one more time to complete the refinements

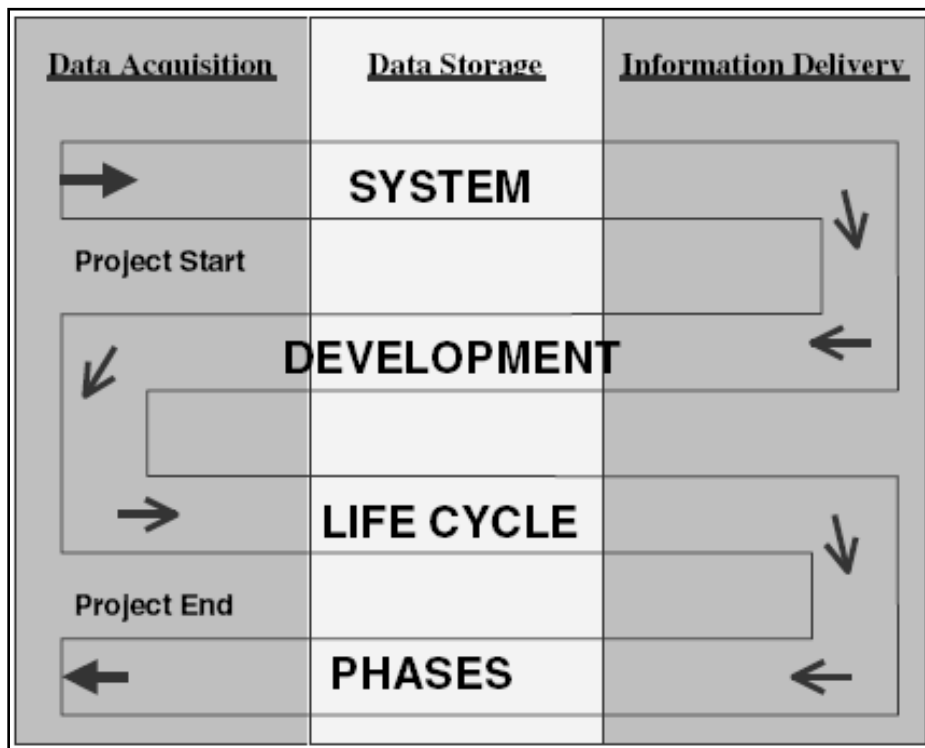


Fig : Data Warehouse functional components and SDLC

- The phases of the Life Cycle wrap around the functional components of the data warehouse i.e. Data Acquisition, Data Storage and Information Delivery components
- Similar to any System Development Life Cycle, the data warehouse project begins with the preparation of a project plan which describes the project, identifies its objectives, lists the assumptions and highlights the critical issues, etc.
- Following diagram is a sample outline of the project plan

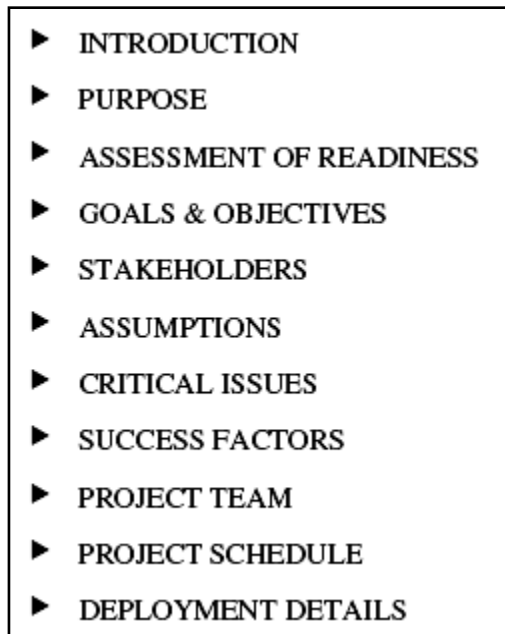
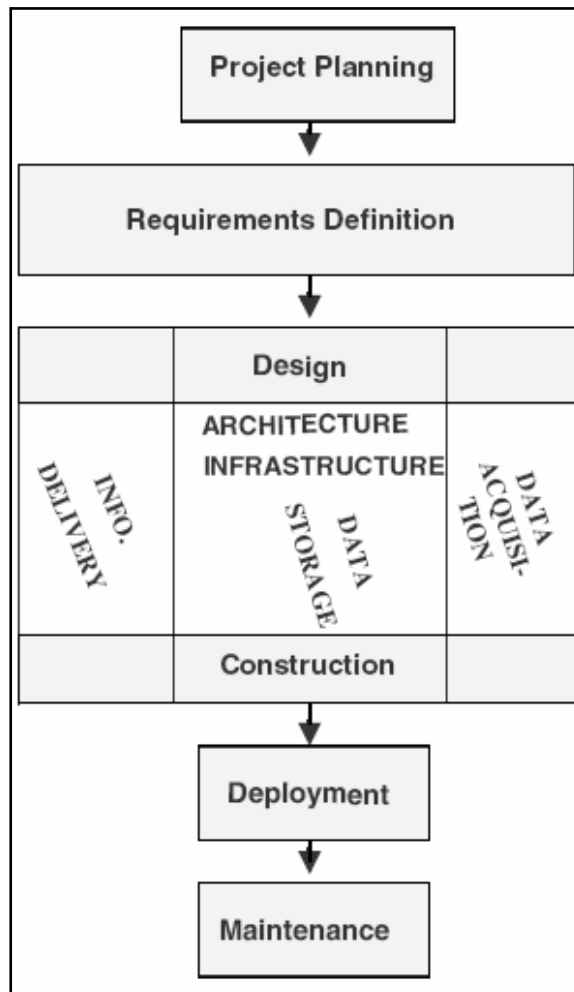


Fig : a sample outline of the project plan

3.2.1 The development phases:

- The Data Acquisition, Data Storage and Information Delivery components form the general architecture of a Data Warehouse
- Proper technical infrastructure should be available to support them hence it should be ensured that the development phases include tasks relating to the three components that define as well as establish the infrastructure to support the architecture of a Data Warehouse.
- The traditional phases of the Life Cycle include
 1. Project Planning
 2. Requirements definition
 3. Design
 4. Construction
 5. Deployment
 6. Maintenance

- The phases may include varying number of tasks if compared. It depends upon which factor is emphasized during the project
ex. Data Quality or Performance



- The three main components of the data warehouse are interleaved within the design and construction phases as shown above that define its architecture and establish its infrastructure.

3.2.2 Dimensional Analysis:

- Building a data warehouse is very different from building an operational system in the requirements gathering phase. Because of this difference, the traditional methods of collecting requirements that work well for operational systems cannot be applied to data warehouses.

The Usage of Information Unpredictable:

- In providing information about the requirements for an operational system, the users are able to give you precise

details of the required functions, information content, and usage patterns. In contrast, for a data warehousing system, the users are generally unable to define their requirements clearly and precisely.

- The users are familiar with operational systems because they use these in their daily work, so they are able to visualize the requirements for other new operational systems, and in other hand, the users cannot relate a data warehouse system to anything they have used before.

Therefore, the whole process of defining requirements for a data warehouse is so vague.

- So now we have to build something the users are unable to define clearly and precisely. This can be done as follows:
1. You may initially collect data on the overall business of the organization.
 2. You may check on the industry's best practices.
 3. You may gather some business rules guiding the day-to-day decision making.

Note: But these are generalities and are not sufficient to determine detailed requirements, because of that managers think of the business in terms of business dimensions.

3.2.3 Dimensional Nature of Business Data:

- The requirements should be collected keeping in mind the various business dimensions the users of the data warehouse may think about.

<p>Marketing Vice President</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>How much did my new product generate month by month, in the southern division, by user demographic, by sales office, relative to the previous version, and compared to plan?</p> </div>
<p>Marketing Manager</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Give me sales statistics by products, summarized by product categories, daily, weekly, and monthly, by sale districts, by distribution channels.</p> </div>
<p>Financial Controller</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Show me expenses listing actual vs budget, by months, quarters, and annual, by budget line items, by district, division, summarized for the whole company.</p> </div>

Fig: Managers think in Business Dimensions

- The above figure shows what questions a typical Marketing Vice president, a Marketing Manager and a Financial Controller may ask.

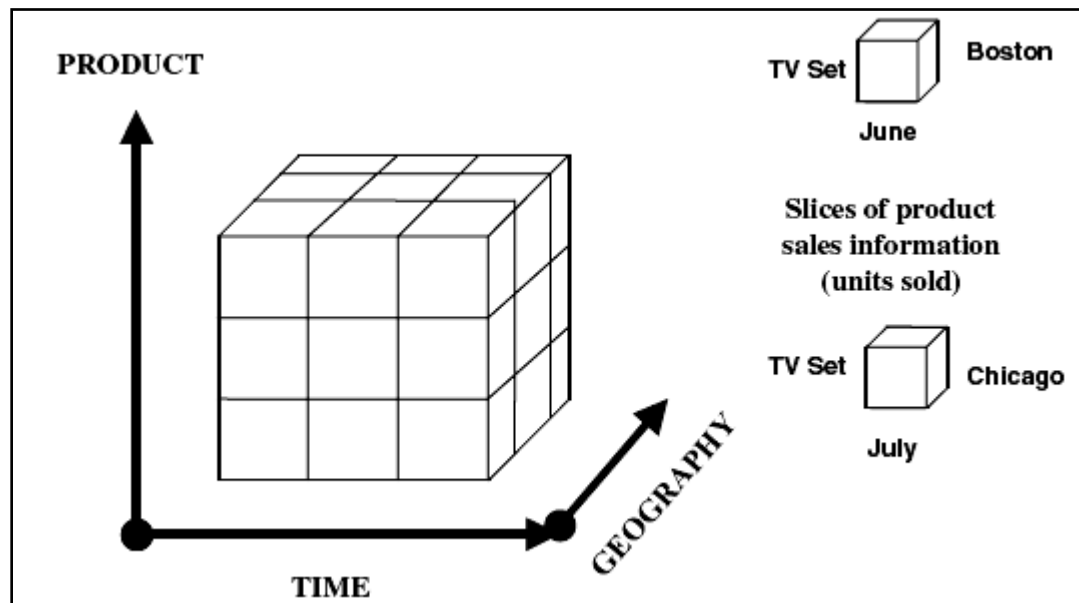


Fig Dimensional Nature of Business Data

- The above figure shows the analysis of the Sales Unit along three business dimensions : Product, Time and Geography.
- In each of the small dimensional cube we can find the sales unit for that particular slice of time, product and geographical division.
- It is observed that the business data unit forms a collection of cubes.
- The business data unit is three dimensional because there are just three dimensions.
- If there are more than three dimensions the concept is extended to multiple dimensions and visualize multidimensional cubes, also called hypercubes.

Examples of Business Dimensions

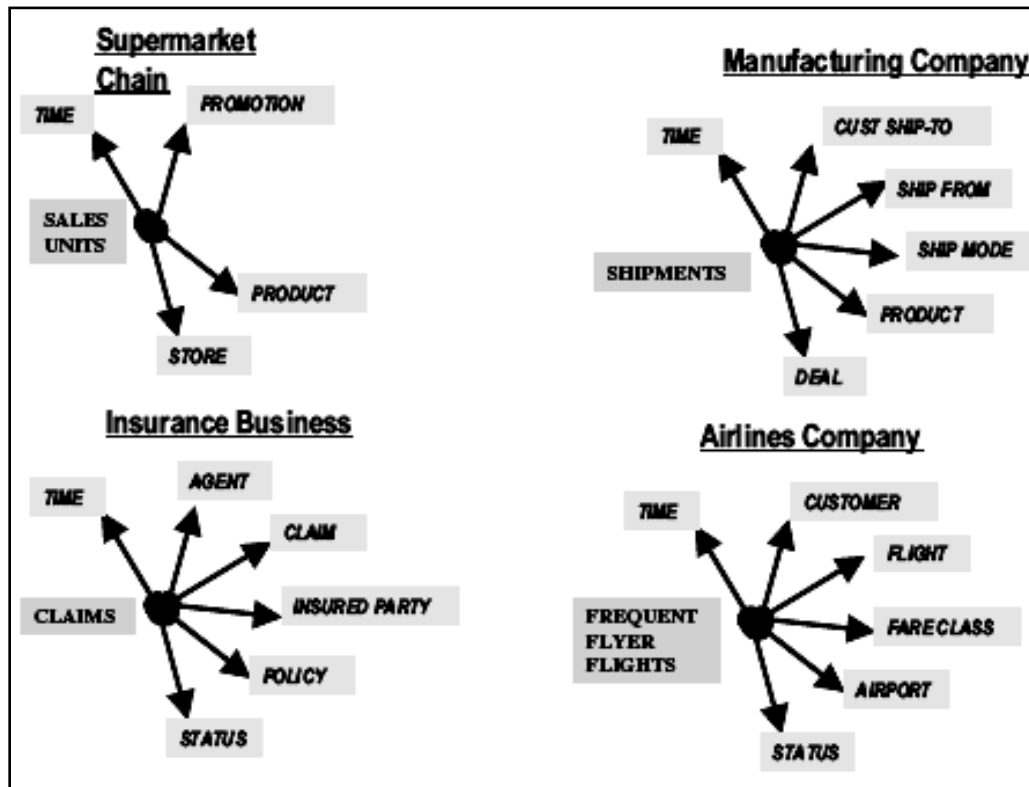


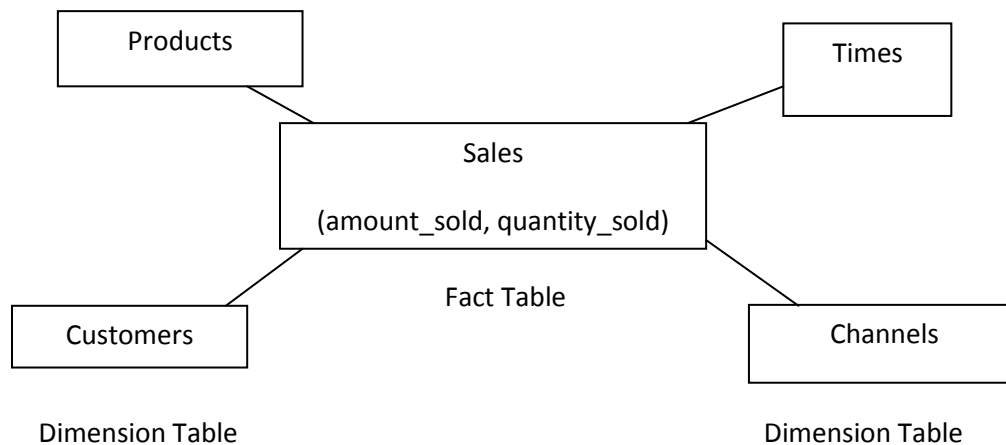
Fig: Examples of Business Dimensions

- For the supermarket chain the Sales Units are analyzed along four business dimension time, promotion, store and product. The Marketing manager for the supermarket would like to know the sales broken down by product, at each store, in time sequence and in relation to the promotions that take place.
- For the Insurance company the claims are analyzed along six business dimensions : time, agent, claim, insured party, policy and status
- What we find from these examples is that the business dimensions are different and relevant to the industry and to the subject for analysis.
- Also the time dimension to be a common dimension in all examples. Almost all business analyses are performed over time.

3.3 STAR SCHEMA

The star schema is the simplest data warehouse schema. It is called a star schema because the diagram resembles a star, with

points radiating from a center. The center of the star consists of one or more fact tables and the points of the star are the dimension tables, as shown in figure.



Fact Tables:

A fact table typically has two types of columns: those that contain numeric facts (often called measurements), and those that are foreign keys to dimension tables. A fact table contains either detail-level facts or facts that have been aggregated.

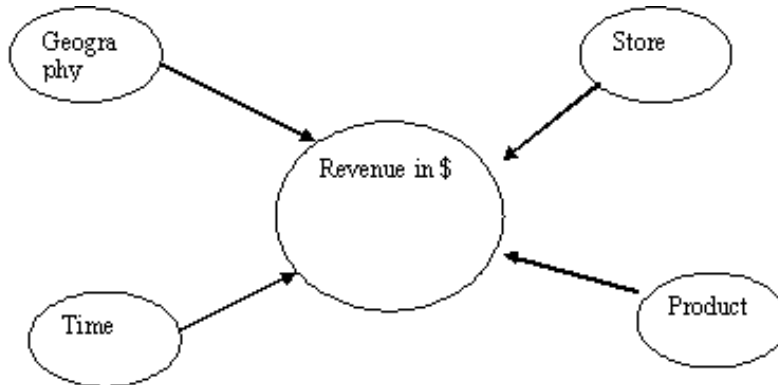
Dimension Tables:

A dimension is a structure, often composed of one or more hierarchies, that categorizes data. Dimensional attributes help to describe the dimensional value. They are normally descriptive, textual values.

Dimension tables are generally small in size as compared to fact table.

-To take an example and understand, assume this schema to be of a retail-chain (like wal-mart or carrefour).

Fact will be revenue (money). Now how do you want to see data is called a dimension.



In above figure, you can see the fact is revenue and there are many dimensions to see the same data. You may want to look at revenue based on time (what was the revenue last quarter?), or you may want to look at revenue based on a certain product (what was the revenue for chocolates?) and so on.

In all these cases, the fact is same, however dimension changes as per the requirement.

Note: In an ideal Star schema, all the hierarchies of a dimension are handled within a single table.

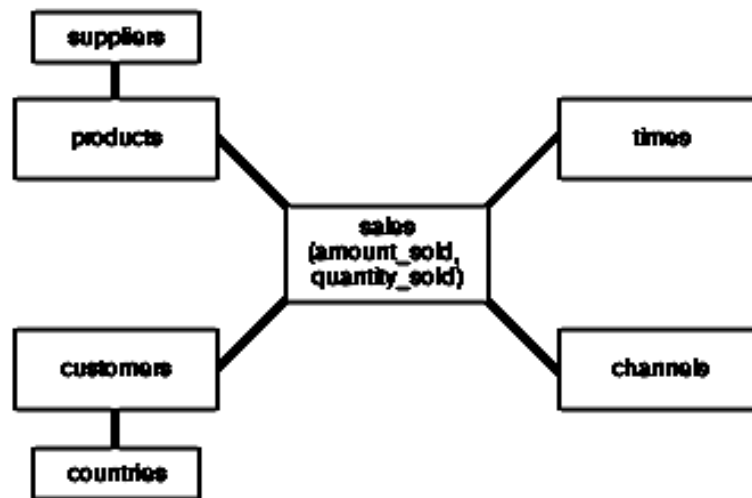
Star Query:

A star query is a join between a fact table and a number of dimension tables. Each dimension table is joined to the fact table using a primary key to foreign key join, but the dimension tables are not joined to each other. The cost-based optimizer recognizes star queries and generates efficient execution plans for them.

3.4 SNOFLAKE SCHEMA

The snowflake schema is a variation of the star schema used in a data warehouse.

The snowflake schema (sometimes called snowflake join schema) is a more complex schema than the star schema because the tables which describe the dimensions are normalized.



Flips of "snowflaking"

In a data warehouse, the fact table in which data values (and its associated indexes) are stored, is typically responsible for 90% or more of the storage requirements, so the benefit here is normally insignificant.

- Normalization of the dimension tables ("snowflaking") can impair the performance of a data warehouse. Whereas conventional databases can be tuned to match the regular pattern of usage, such patterns rarely exist in a data warehouse. Snowflaking will increase the time taken to perform a query, and the design goals of many data warehouse projects is to minimize these response times.

Benefits of "snowflaking"

- If a dimension is very sparse (i.e. most of the possible values for the dimension have no data) and/or a dimension has a very long list of attributes which may be used in a query, the dimension table may occupy a significant proportion of the database and snowflaking may be appropriate.
- A multidimensional view is sometimes added to an existing transactional database to aid reporting. In this case, the tables which describe the dimensions will already exist and will typically be normalised. A snowflake schema will hence be easier to implement.
- A snowflake schema can sometimes reflect the way in which users think about data. Users may prefer to generate queries using a star schema in some cases, although this may or may not be reflected in the underlying organisation of the database.
- Some users may wish to submit queries to the database which,

using conventional multidimensional reporting tools, cannot be expressed within a simple star schema. This is particularly common in data mining of customer databases, where a common requirement is to locate common factors between customers who bought products meeting complex criteria. Some snowflaking would typically be required to permit simple query tools such as Cognos Powerplay to form such a query, especially if provision for these forms of query weren't anticipated when the data warehouse was first designed.

In practice, many data warehouses will normalize some dimensions and not others, and hence use a combination of snowflake and classic star schema.



4

OLAP

Unit Structure:

- 4.1 Introduction to OLAP
- 4.2 OLAP Architecture
- 4.3 Relational OLAP
 - 4.3.1 ROLAP Architecture
- 4.4 ROLAP vs MOLAP
 - 4.4.1 MOLAP
 - 4.4.2 ROLAP
- 4.5 Major Features and Functions

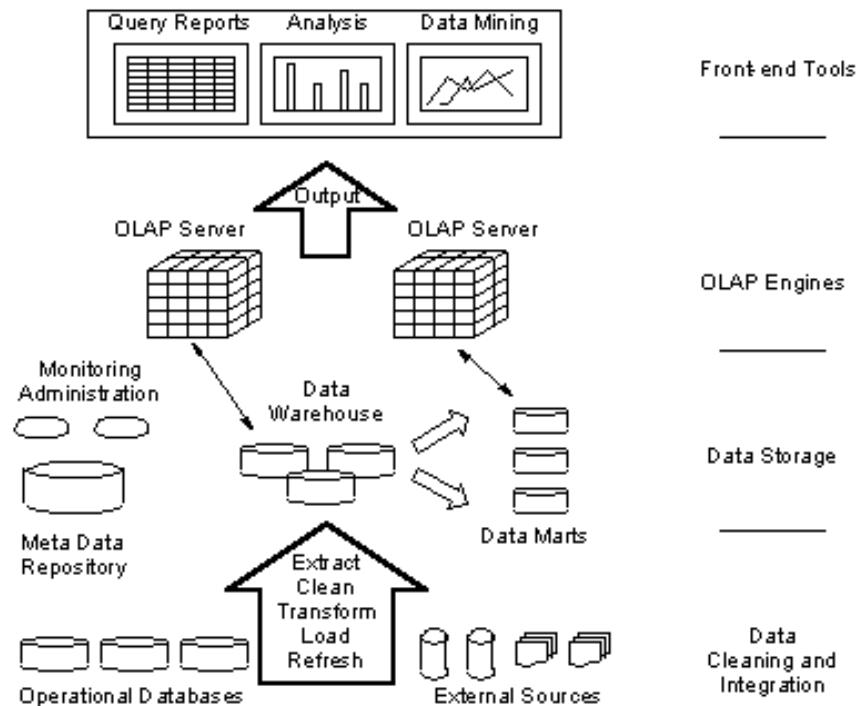
4.1 INTRODUCTION TO OLAP:

OLAP means Online Analytical Processing which is used for processing of data so as to manipulate it for analysis. OLAP is used as a vehicle for carrying out analysis in data warehouse which is best for analysis. In due course we will be able to understand need of OLAP, major functions and features of OLAP, different models of OLAP so that one can understand which one to apply for their own computing environment and various tools of OLAP.

4.2 OLAP ARCHITECTURE:

The classic definition of the data warehouse was provided by W. H. Inmon when he described it as a "subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process."

In general, data warehouses can be seen as three-tiered data models. As shown in the figure below, information is first extracted from operational sources and then cleaned, transformed and loaded into the data warehouse. This step has received a considerable amount of attention from OLAP vendors. Often, the production data resides in a collection of remote, heterogeneous repositories and must undergo considerable *massaging* before it can be integrated into a single clean store.



OLAP ARCHITECTURE

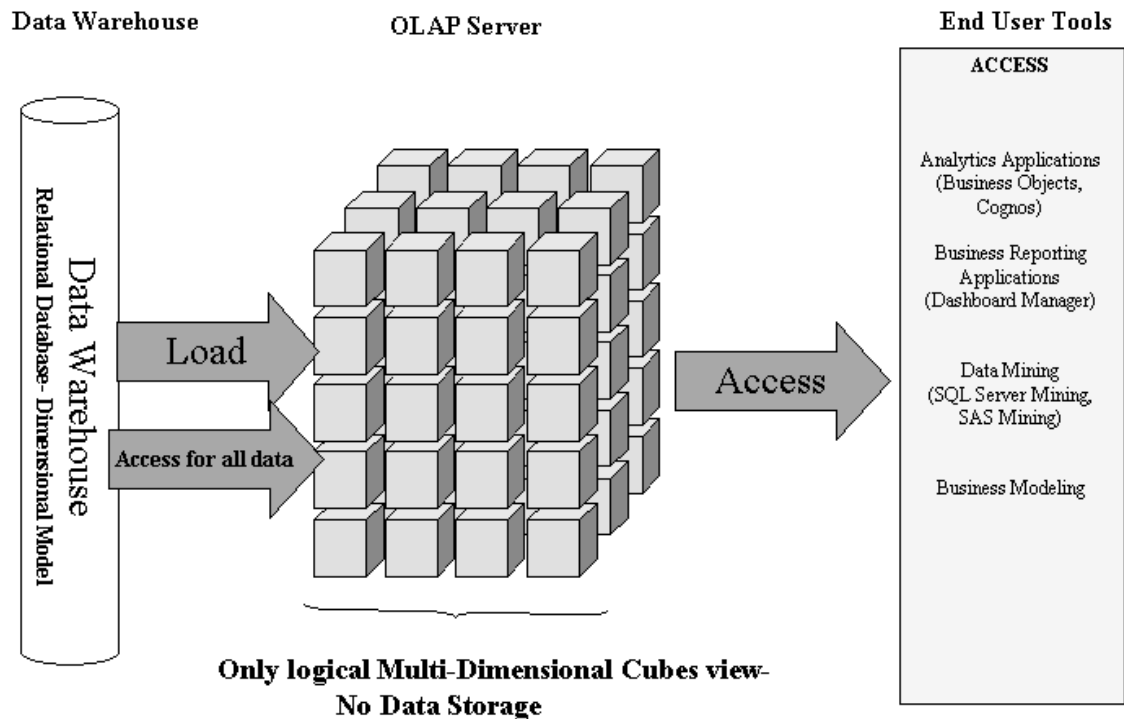
Once the data has been selected from the remote sources, it is placed into the data warehouse, which is almost always a relational data base.

The data warehouse itself may be built as a massive enterprise-wide entity and/or a collection of data marts each containing some subset of the corporate data. Now it is the job of OLAP server at level three to supply analytical functionality for the DSS system. There are two forms of OLAP servers, known as ROLAP and MOLAP. Finally in the top tier, we find that front end tools provide a convenient, user friendly interface to the knowledge workers who will exploit the system.

4.3 RELATIONAL OLAP

ROLAP is an abbreviation of Relational OLAP. It is an alternative to the MOLAP (Multidimensional OLAP) technology. It is one of the forms of OLAP server. While both ROLAP and MOLAP analytic tools are designed to allow analysis of data through the use of a multidimensional data model, there is a slight difference among them. ROLAP does not require the pre-computation and storage of information. Instead, ROLAP tools access the data in a relational database and generate SQL queries to calculate information at the appropriate level when an end user requests it. With ROLAP, it is possible to create additional database tables which summarize the data at any desired combination of dimensions.

While ROLAP uses a relational database source, generally the database must be carefully designed for ROLAP use. A database which was designed for OLTP will not function well as a ROLAP database. Therefore, ROLAP still involves creating an additional copy of the data. However, since it is a database, a variety of technologies can be used to populate the database.



ROLAP Architecture

4.3.1 ROLAP Architecture:

This OLAP stores data in relational format, while presenting it in Multi-dimensional format. ROLAP architecture provides a multi-dimensional view of data to the user, but stored the data in relational database format. ROLAP architecture works with the concept that relational storage of data can give much higher level of scalability, can imbibe as many dimensions as possible etc, as needed, and can provide faster response time due to indexing and other features. There is no database in ROLAP server like that of a MOLAP server. ROLAP server contains analytics objects, which create dynamic aggregation of the RDBMS data in Data Warehouse and present it to the user.

4.4 ROLAP vs MOLAP

One of the most important elements of OLAP environments is their dependence upon multi-dimensional data values. Data warehouses represent subject-oriented records rather than transaction-oriented ones. As such, aggregated values can be viewed as existing within a logical *cube*, where the user is free to index the cube on one or more dimensional axes. This type of

conceptual representation of the data gives rise to what is known as data cube.

4.4.1 MOLAP:

MOLAP is the more traditional way of OLAP analysis. Here, data is stored in a multidimensional cube. The storage is not in the relational database, but in named formats.

Advantages:

- ✓ Excellent Performance: MOLAP cubes are built for fast data retrieval, and are optimal for slicing and dicing operations.
- ✓ Can perform complex calculations: All calculations have been pre-generated when the cube is created. Hence, complex calculations are not only solvable, but they return quickly.

Disadvantages:

- ✓ Limited in the amount of data it can handle: Because all calculations are performed when the cube is built, it is not possible to include a large amount of data in the cube itself. This is not to say that the data in the cube cannot be derived from a large amount of data. Indeed, this is possible. But in this case, only summary-level information will be included in the cube itself.
- ✓ Requires additional investment: Cube technology are often proprietary and do not already exist in the organization. Therefore, to adopt MOLAP technology, chances are additional investments in human and capital resources are needed.

4.4.2 ROLAP:

ROLAP depends on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement.

Advantages:

- ✓ Can handle large amounts of data: The data size limitation of ROLAP technology is the limitation on data size of the underlying relational database. In other words, ROLAP itself places no limitation on data amount.
- ✓ Can influence functionalities inherent in the relational database: Often, relational database already comes with a host of functionalities. ROLAP technologies, since they sit on

top of the relational database, can therefore leverage these functionalities.

Disadvantages:

- ✓ Performance can be slow: Because each ROLAP report is essentially a SQL query (or multiple SQL queries) in the relational database, the query time can be long if the underlying data size is large.
- ✓ Limited by SQL functionalities: Because ROLAP technology mainly relies on generating SQL statements to query the relational database, and SQL statements do not fit all needs (for example, it is difficult to perform complex calculations using SQL), ROLAP technologies are therefore traditionally limited by what SQL can do. ROLAP vendors have mitigated this risk by building into the tool out-of-the-box complex functions as well as the ability to allow users to define their own functions.

Web Based OLAP:

In the hope that an enterprise will get useful information for making strategic decisions for the benefit of organisation, large amounts of time and money is given for building data warehouse. The Extension of OLAP capabilities to large group of analysts are used so as to increase the value potential of data warehouse. Web-Enabled data warehouses allow not only groups which are inside the enterprise but also the groups outside the enterprise so that OLAP functionalities can be extended to more than a selected group of analysts. Web-based OLAP emerged in the year 1997. WOLAP allow users of Web browsers to access and analyze data warehouse data. WOLAP system is an OLAP system which operates on client/server model. Certain limitations to Web-based operational limits client functionality to what web browsers can be expected to do. Two possibilities are :

- The analysis can be done fully on the server side which in turn will return the result converted to HTML and sent to the client for display.
- The analysis programs can be written as Java applets, Java script code etc.

Both the approaches are independent of client architecture. WOLAP eliminates the need to install a software package on the user's computer, with the attendant issues of administration, upgrades etc. Though the abbreviations ROLAP and WOLAP feel like as if they are alternatives of OLAP but that is not the case. ROLAP refers to the nature of underlying database, as over here it is Relational. Whereas WOLAP refers to the way in which analysis software is structured and executed, as over here it is via the Web

or Web like Intranet. WOLAP applications are designed to work on any kind of database.

4.5 MAJOR FEATURES AND FUNCTIONS

OLAP is much more than an information delivery system for data warehouse. OLAP focus is on the end-users analytical requirements. Goal of OLAP is to support ad-hoc querying for the business analyst. OLAP features generally include Calculations and modeling across dimensions, trend analysis, slicing subsets, drill-down, drill-through and rotation. You will gain

Functional requirements for an OLAP tool:

- Fast Access and Calculations
- Powerful Analytical Capabilities
- Flexibility
- Multi-End-User Support

Drill-Down and Roll-Up:

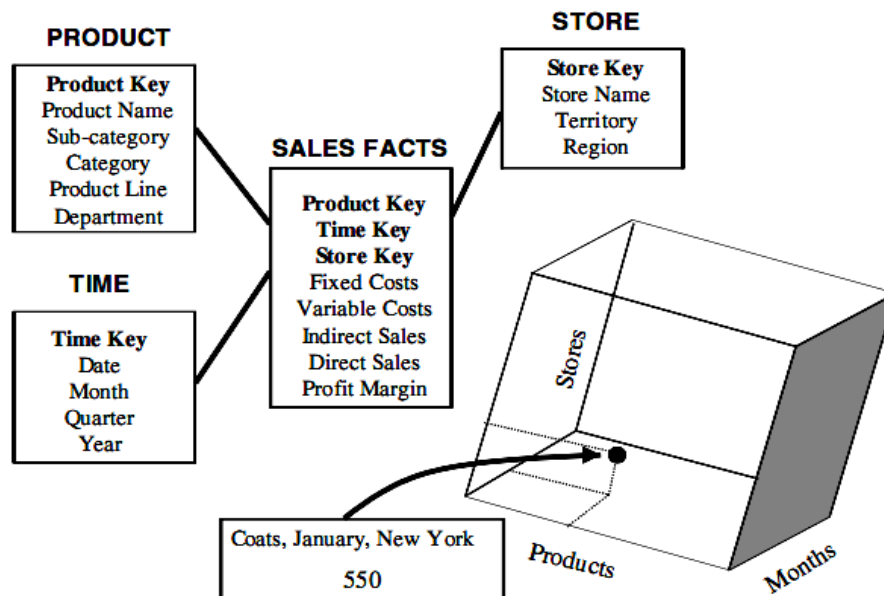
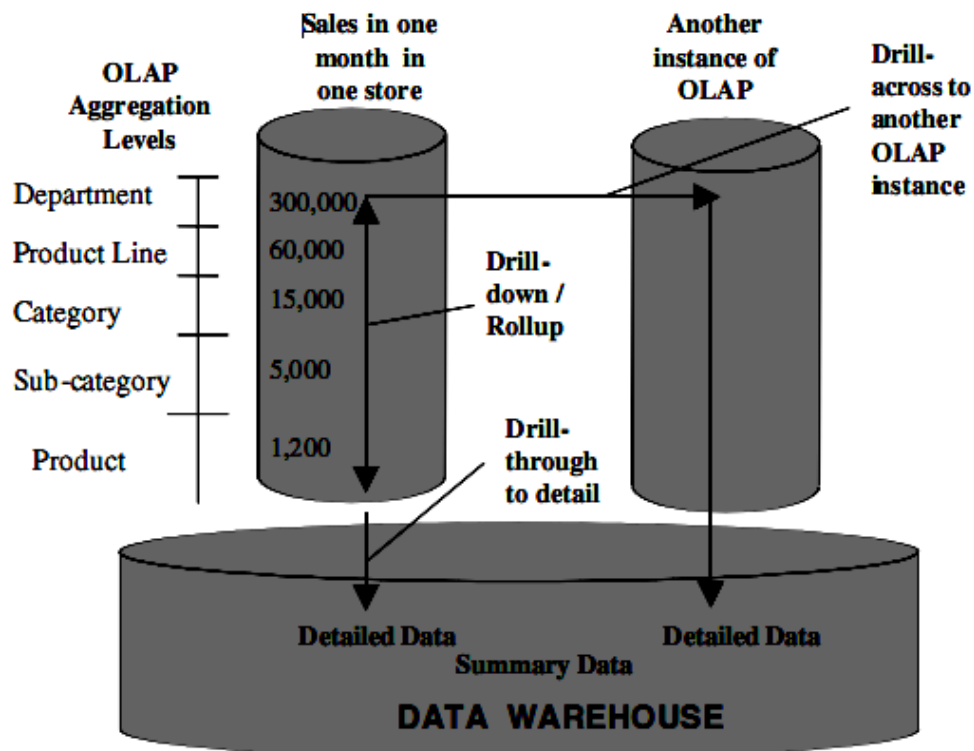


Figure 15-5 Simple STAR schema.

Fig: Referred from Data Warehousing fundamentals by Paulraj Ponniah

In the above figure, the attributes of the product dimension: product name, subcategory, category, product line and department denote an ascending hierarchical sequence from product name to department. A department includes product lines, a product line includes categories, a category includes subcategories, and each subcategory consists of products with individual product names. In an OLAP system, these attributes are known as hierarchies of the product dimension.

OLAP systems provide drill-down and roll-up capabilities.



In the above diagram, these capabilities are explained with the help of product dimension hierarchies. It explains the rolling up to higher hierarchical levels of an aggregation and then drilling down to lower levels of details. The sales numbers shown are sales for one particular store in one specific month at the above levels of aggregation. If noted down properly, the sale numbers as you go down the hierarchy are for single department, single product line, single category etc. Drill down capability of OLAP system not only allows us to get the lower level breakdown of sales but also shows the drill across to another OLAP summarization using a different set of hierarchies of other dimensions. Roll-up, drill-down, drill-across and drill-through are some of the useful features of OLAP systems for supporting multidimensional analysis.

Slice and Dice or Rotation:

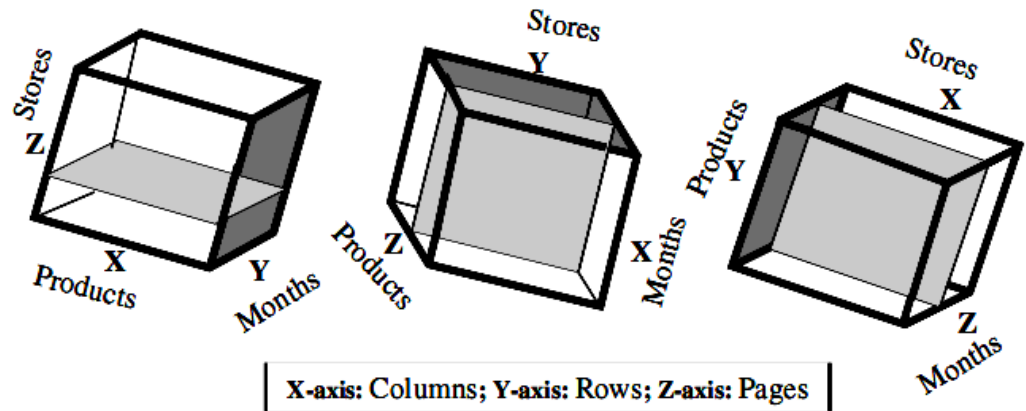
Store: New York

Products

PAGES: STORE dimensionCOLUMNS: PRODUCT dimension

ROWS: TIME dimension Months		Hats	Coats	Jackets	Dresses	Shirts	Slacks
	Jan	200	550	350	500	520	490
	Feb	210	480	390	510	530	500
	Mar	190	480	380	480	500	470
	Apr	190	430	350	490	510	480
	May	160	530	320	530	550	520
	Jun	150	450	310	540	560	330
	Jul	130	480	270	550	570	250
	Aug	140	570	250	650	670	230
	Sep	160	470	240	630	650	210
	Oct	170	480	260	610	630	250
	Nov	180	520	280	680	700	260
	Dec	200	560	320	750	770	310

In the above diagram, each row represents month, each column represents product and each page represents stores where each page represents the sales for one store. The data model with these data elements represented by its primary edges corresponds to a physical cube. The page displayed is a slice of the cube means two-dimensional plane of the cube. In the above figure, it displays a page for new York store that is parallel to the product and time axes.



Store: New York				Product: Hats				Month: January			
	Hats	Coats	Jackets		Jan	Feb	Mar		New York	Boston	San Jose
Jan	200	550	350	New York	200	210	190	Hats	200	210	130
Feb	210	480	390	Boston	210	250	240	Coats	550	500	200
Mar	190	480	380	San Jose	130	90	70	Jackets	350	400	100

In the above example, only 3 stores, 3 months and 3 products are chosen for illustration, wherein the first part from left shows the alignment of the cube. Initially the product is along X axis, months is along Y axis and store is along Z axis. Rotate the cube such that products are along Z-axis and months are along X-axis and stores are along the Y-axis. The portion of the cube or say slice is also rotating. Now the page displays months as the columns and products as the rows. The display page represents the sales of one product: hats. This way we can perform many rotations with the slice available to us. That is different versions of the slices on the cube we are able to see, this is the advantage of Slice and Dice or Rotation.

Bitmap Indexes:

The collection of bit vectors for one column is known as bitmap index for that column.

For eg: Consider a table that describes employees

Employees(empid:integer, name:string, gender: boolean, rating: integer)

The rating value is an integer in the range 1 to 10, and only two values are recorded for gender. Columns with few possible values are called sparse. We can exploit sparsity to construct a new kind of index that greatly speeds up queries to these columns.

The idea is to record values for sparse columns as a sequence of bits, one for each possible value. If we consider the gender values for all rows in the Employees table, we can treat this as a collection of two bit vectors, one of which has the associated value M for Male and the other the associated value F for Female. Each bit vector has one bit per row in the Employees table, indicating whether the value in that row is the value associated with the bit vector. The collection of these bit vectors is known as bitmap index.



5

DATA MINING

Unit Structure:

- 5.1 Introduction
- 5.2 Data Mining Algorithms
 - 5.2.1 Clustering
 - 5.2.2 Classification
 - 5.2.3 Association Rules
- 5.3 KDD Process:
- 5.4 Decision Trees:
- 5.5 Neural Networks
- 5.6 Search Engines
 - 5.6.1 Characteristics
 - 5.6.2 Functionality
 - 5.6.3 Architecture

- 5.6.4 Ranking of web pages
- 5.6.5 The search engine industry
- 5.6.6 The enterprise search
- 5.7 Case Study

5.1 INTRODUCTION:

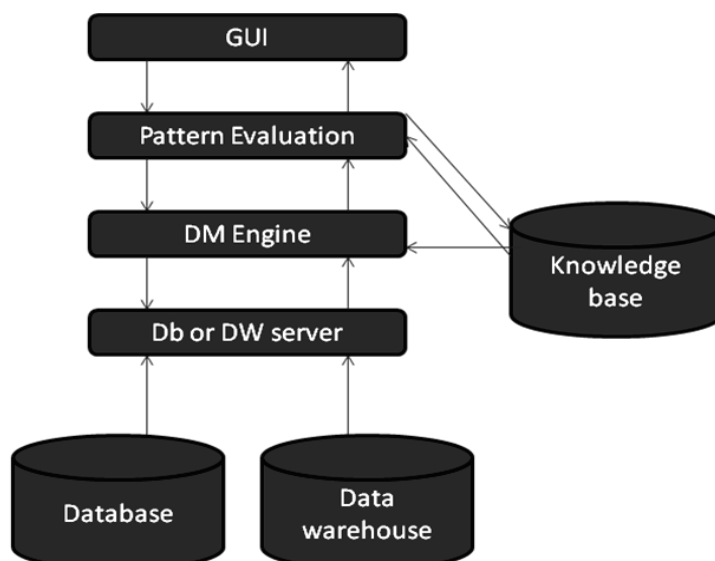
Data mining is often defined as finding hidden information in a db. Through Data Mining we don't get a subset of data stored in db, instead, we get analysis of contents of db.

Data mining is sorting through data to identify patterns and establish relationships.

=> **Data mining parameters include:**

- * **Association** - looking for patterns where one event is connected to another event
- * **Sequence or path analysis** - looking for patterns where one event leads to another event
- * **Classification** - looking for new patterns
- * **Clustering** - finding and visually documenting groups of facts not previously known
- **Forecasting** - discovering patterns in data that can lead to reasonable predictions about the future (This area of data mining is known as predictive analytics.)

DATA MINING ARCHITECTURE



- 1) **Db or DW**: Information repositories. Data cleaning and data integration techniques are performed here.
- 2) **Db or DW server** : Responsible for fetching relevant data, based on user's request.
- 3) **Knowledge base**: This is domain knowledge that is used to guide the search.
- 4) **DM engine** : Consists of set of functional modules for tasks such as cluster analysis, association etc...
- 5) **Pattern evaluation** : Includes interestingness measures and interacts with DM modules so as to focus the search towards interesting patterns .
- 6) **GUI** : This module is responsible for communication between user and DM system.

5.2 DATA MINING ALGORITHMS

5.2.1 CLUSTERING :

Here we partition the available data into groups such that records in one group are similar to each other and records in different groups are dissimilar to each other. Each group is called a cluster. Similarity between records is measured by distance function. The clustering algorithm's output consists of summarized representation of each cluster.

Clustering algorithm called BRICH handles very large databases. Two parameters are set for BRICH algorithm, namely, threshold on the amount of main memory available and initial threshold for radius(ϵ) of any cluster.

The algorithm reads records from database and sequentially processes them as follows:

- 1) Compute the distance between record r and each of the existing cluster centers. Let I be the cluster index such that the distance between r and C_i is the smallest. (C_i - Center of cluster).
- 2) Compute the value of new radius R'_i of the i th cluster under the assumption that r is inserted into it. If $R'_i \leq \epsilon$, then the i th cluster remains compact and we assign r to the i th cluster by updating its center and setting its radius to R'_i . If $R'_i > \epsilon$, then the i th cluster is no longer be compact and if we insert r into it. Therefore we start with new cluster containing only the record r .

5.2.2 CLASSIFICATION:

Classification is a data mining (machine learning) technique used to predict group membership for data instances. Unlike clustering, a classification analysis requires that the end-user/analyst know ahead of time how classes are defined.

For example, classes can be defined to represent the likelihood that a customer defaults on a loan (Yes/No). It is necessary that each record in the dataset used to build the classifier already have a value for the attribute used to define classes. Because each record has a value for the attribute used to define the classes, and because the end-user decides on the attribute to use, classification is much less exploratory than clustering. The objective of a classifier is not to explore the data to discover interesting segments, but rather to decide how new records should be classified -- i.e. is this new customer likely to default on the loan?

Classification routines in data mining also use a variety of algorithms -- and the particular algorithm used can affect the way records are classified.

5.2.3 ASSOCIATION RULES:

In data mining, **association rule learning** is a popular and well researched method for discovering interesting relations between variables in large databases. Based on the concept of strong rules, association rules for discovering regularities between products in large scale transaction data recorded by point-of-sale (POS) systems in supermarkets are introduced. For example, the rule {pen} => {ink} can be read as "If a pen is purchased, it is likely that ink is also purchased in that transaction. . Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. Association rules are employed today in many application areas including Web usage mining, intrusion detection and bioinformatics.

5.3 KDD PROCESS

The Knowledge discovery and data mining can roughly be described in 4 steps:

1) Data selection:

The target subset of data and attributes of interest are identified by examining the entire raw dataset.

2) Data Cleaning :

Noise and outliers are removed and some new fields are created by combining existing fields

3) Data Mining :

Apply data mining algorithms to extract interesting patterns

4) Evaluation:

The patterns are presented to the end user in an understandable form, for example, through visualization.

KDD Process can be described in detail as follows:

1. Developing an understanding of
 - the application domain
 - the relevant prior knowledge
 - the goals of the end-user
2. Creating a target data set: selecting a data set, or focusing on a subset of variables, or data samples, on which discovery is to be performed.
3. Data cleaning and preprocessing.
 - Removal of noise or outliers.
 - Collecting necessary information to model or account for noise.
 - Strategies for handling missing data fields.
 - Accounting for time sequence information and known changes.
4. Data reduction and projection.
 - Finding useful features to represent the data depending on the goal of the task.
 - Using dimensionality reduction or transformation methods to reduce the effective number of variables under consideration or to find invariant representations for the data.
5. Choosing the data mining task.
 - Deciding whether the goal of the KDD process is classification, regression, clustering, etc.
6. Choosing the data mining algorithm(s).
 - Selecting method(s) to be used for searching for patterns in the data.
 - Deciding which models and parameters may be appropriate.

- Matching a particular data mining method with the overall criteria of the KDD process.
7. Data mining.
 - Searching for patterns of interest in a particular representational form or a set of such representations as classification rules or trees, regression, clustering, and so forth.
 8. Interpreting mined patterns.
 9. Consolidating discovered knowledge.

5.4 DECISION TREES

A decision tree is a graphical representation of a collection of classification rules. The tree directs the record from the root to a leaf. Each internal node of the tree is labeled with a predictor attribute called as splitting attribute because the data is split based on condition over this attribute.

The algorithm is as follows:

Input: node n , partition D , split selection method S .

Output: decision tree for D rooted at node n .

Algorithm:

- 1) Apply S to D to find splitting criteria.
- 2) If (a good splitting criteria is found)
- 3) Create two child nodes n_1 and n_2 of n .
- 4) Partition D into D_1 and D_2 .
- 5) BuildTree(n_1, D_1, S)
- 6) BuildTree(n_2, D_2, S)
- 7) endIf

Decision tree advantages:

Decision trees have various advantages:

- **Simple to understand and interpret.** People are able to understand decision tree models after a brief explanation.
- **Requires little data preparation.** Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- **Able to handle both numerical and categorical data.** Other techniques are usually specialized in analyzing datasets that have only one type of variable. Ex: relation

rules can be used only with nominal variables while neural networks can be used only with numerical variables.

- **Use a white box model.** If a given situation is observable in a model the explanation for the condition is easily explained by Boolean logic. An example of a black box model is an artificial neural network since the explanation for the results is difficult to understand.
- **Possible to validate a model using statistical tests.** That makes it possible to account for the reliability of the model.
- **Robust.** Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
- **Perform well with large data in a short time.** Large amounts of data can be analysed using personal computers in a time short enough to enable stakeholders to take decisions based on its analysis.

5.5 NEURAL NETWORKS

Neural Network is information processing system graph the processing system and the various algorithms that access that graph.

Neural Network is a structured graph with many nodes (processing elements) and arcs (interconnections) between them.

If compared with human brain, nodes are like individual neurons and arcs are the interconnections between neurons.

NN a directed with 3 layers namely: input layer, hidden layer and output layer.

The development of NNs from neural biological generalizations has required some basic assumptions which we list below:

1. "Information processing occurs at many simple elements called neurons [also referred to as units, cells, or nodes].
2. Signals are passed between neurons over connection links.
3. Each connection link has an associated weight, which, in a typical neural net, is the signal transmitted.
4. Each neuron applies an activation function (usually nonlinear) to its net input (weighted input signals) to determine its output signal."

The tasks to which artificial neural networks are applied tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind signal separation and compression.

Application areas of ANNs include system identification and control (vehicle control, process control), game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition, etc.), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications, data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.

- Moreover, some brain diseases, e.g. Alzheimer, are apparently, and essentially, diseases of the brain's natural NN by damaging necessary prerequisites for the functioning of the mutual interconnections between neurons.

5.6 SEARCH ENGINES

5.6.1 Characteristics:

One of the things that will make you a better web searcher is understanding what you need to type in order to tell the search engine exactly what you are looking for.

There are two categories of search operators: Boolean and **non-Boolean** operators. An operator is a word or symbol that you type in that gives the search engine directions to help it know what to search for. Using these operators can narrow or widen your search, helping you find web sites that may be useful to you. You will have to check out the individual search engines to find out which operators work.

Boolean Searching:

AND means the search results must have both terms – often it is typed in UPPER CASE, but not always AND decreases the number of web sites that will be found, narrowing in on specific topics.

Example: pollution AND water will look for web sites about both pollution and water **OR** means the search results can have just one of the terms OR increases the number of web sites that will be found, broadening your search.

Example: pollution OR water will look for web sites that mention **either** pollution or water **NOT** means any result containing the second term will be excluded NOT decreases the number of web sites that will be found, narrowing your search.

Example: pollution NOT water will look for web sites about pollution that do not involve water.

Warning: Be very careful about using **NOT**. If the web site mentions water even once, it may be left out. This could rule out some very useful web sites. Likewise, you must be careful when using **OR**. You can end up with a huge number of sites to sort through.

Non-Boolean Searching:

+ works like AND, making the term required in the search results the + should be placed directly in front of the search term without any spaces.

Example: pollution +water will look for web sites about pollution that also mention water.

- works like NOT and means to exclude the term the - should be placed directly in front of the search term without any spaces.

Example: pollution -water will also look for web sites about pollution that do not involve water.

" " placed around terms means the search engine will look for the exact phrase.

Example: "water pollution" will look for that exact phrase. This can make your search very specific.

Warning: Just as with the Boolean terms, you must be careful when using **-** that you do not eliminate web sites that might mention the term you do not want, but are not really about that term. **- water** may eliminate web sites that are about air pollution but mention water pollution as well.

How to find out which search operators to use:

Different web sites and search engines use different systems, and you will need to learn which search terms work best with your particular search engine.

Many web sites and search engines will tell you which to use if you click on buttons such as HELP or ADVANCED SEARCH or POWER SEARCH or SEARCH TIPS. They may also offer other search limit options such as letting you pick dates or languages. Explore your favorite search engines to find out what they can do!

5.6.2 Functionality:

The main functionalities of search engines are :

- Keyword Searching
- Refining Your Search
- Relevancy Ranking
- Meta Tags
- Concept-based Searching

Search engines use automated software programs known as spiders or bots to survey the Web and build their databases. Web documents are retrieved by these programs and analyzed. Data collected from each web page are then added to the search engine index. When you enter a query at a search engine site, your input is checked against the search engine's index of all the web pages it has analyzed. The best urls are then returned to you as hits, ranked in order with the best results at the top.

Keyword Searching:

This is the most common form of text search on the Web. Most search engines do their text query and retrieval using keywords.

What is a keyword, exactly? It can simply be any word on a webpage. For example, I used the word "simply" in the previous sentence, making it one of the keywords for this particular webpage in some search engine's index. However, since the word "simply" has nothing to do with the subject of this webpage (i.e., how search engines work), it is not a very useful keyword. Useful keywords and key phrases for this page would be "search," "search engines," "search engine methods," "how search engines work," "ranking" "relevancy," "search engine tutorials," etc. Those keywords would actually tell a user something about the subject and content of this page.

Unless the author of the Web document specifies the keywords for her document (this is possible by using [meta tags](#)), it's up to the search engine to determine them. Essentially, this means that search engines pull out and index words that appear to be significant. Since engines are software programs, not rational human beings, they work according to rules established by their creators for what words are *usually* important in a broad range of documents. The title of a page, for example, usually gives useful information about the subject of the page (if it doesn't, it should!). Words that are mentioned towards the beginning of a document (think of the "topic sentence" in a high school essay, where you lay out the subject you intend to discuss) are given more weight by most search engines. The same goes for words that are repeated several times throughout the document.

Some search engines index every word on every page. Others index only part of the document.

Full-text indexing systems generally pick up every word in the text except commonly occurring stop words such as "a," "an," "the," "is," "and," "or," and "www." Some of the search engines discriminate upper case from lower case; others store all words without reference to capitalization.

The Problem With Keyword Searching:

Keyword searches have a tough time distinguishing between words that are spelled the same way, but mean something different (i.e. hard cider, a hard stone, a hard exam, and the hard drive on your computer). This often results in hits that are completely irrelevant to your query. Some search engines also have trouble with so-called stemming -- i.e., if you enter the word "big," should they return a hit on the word, "bigger?" What about singular and plural words? What about verb tenses that differ from the word you entered by only an "s," or an "ed"?

Search engines also cannot return hits on keywords that mean the same, but are not actually entered in your query. A query on heart disease would not return a document that used the word "cardiac" instead of "heart."

Refining Your Search

Most sites offer two different types of searches--"basic" and "refined" or "advanced." In a "basic" search, you just enter a keyword without sifting through any pulldown menus of additional options. Depending on the engine, though, "basic" searches can be quite complex.

Advanced search refining options differ from one search engine to another, but some of the possibilities include the ability to search on more than one word, to give more weight to one search term than you give to another, and to exclude words that might be likely to muddy the results. You might also be able to search on proper names, on phrases, and on words that are found within a certain proximity to other search terms.

Some search engines also allow you to specify what form you'd like your results to appear in, and whether you wish to restrict your search to certain fields on the internet (i.e., usenet or the Web) or to specific parts of Web documents (i.e., the title or URL).

Many, but not all search engines allow you to use so-called Boolean operators to refine your search. These are the logical terms AND, OR, NOT, and the so-called proximal locators, NEAR and FOLLOWED BY.

Boolean AND means that all the terms you specify must appear in the documents, i.e., "heart" AND "attack." You might use this if you wanted to exclude common hits that would be irrelevant to your query.

Boolean OR means that at least one of the terms you specify must appear in the documents, i.e., bronchitis, acute OR chronic. You might use this if you didn't want to rule out too much.

Boolean NOT means that at least one of the terms you specify must not appear in the documents. You might use this if you anticipated results that would be totally off-base, i.e., nirvana AND Buddhism, NOT Cobain.

Not quite Boolean + and - Some search engines use the characters + and - instead of Boolean operators to include and exclude terms.

NEAR means that the terms you enter should be within a certain number of words of each other. FOLLOWED BY means that one term must directly follow the other. ADJ, for adjacent, serves the same function. A search engine that will allow you to search on phrases uses, essentially, the same method (i.e., determining adjacency of keywords).

Phrases: The ability to query on phrases is very important in a search engine. Those that allow it usually require that you enclose the phrase in quotation marks, i.e., "space the final frontier."

Capitalization: This is essential for searching on proper names of people, companies or products. Unfortunately, many words in

English are used both as proper and common nouns--Bill, bill, Gates, gates, Oracle, oracle, Lotus, lotus, Digital, digital--the list is endless.

All the search engines have different methods of refining queries.

The best way to learn them is to read the help files on the search engine sites and practice!

Relevancy Rankings

Most of the search engines return results with confidence or relevancy rankings. In other words, they list the hits according to how closely they think the results match the query. However, these lists often leave users shaking their heads in confusion, since, to the user, the results may seem completely irrelevant.

Why does this happen? Basically it's because search engine technology has not yet reached the point where humans and computers understand each other well enough to communicate clearly.

Most search engines use search term frequency as a primary way of determining whether a document is relevant. If you're researching diabetes and the word "diabetes" appears multiple times in a Web document, it's reasonable to assume that the document will contain useful information. Therefore, a document that repeats the word "diabetes" over and over is likely to turn up near the top of your list.

If your keyword is a common one, or if it has multiple other meanings, you could end up with a lot of irrelevant hits. And if your keyword is a subject about which you desire information, you don't need to see it repeated over and over--it's the information *about* that word that you're interested in, not the word itself.

Some search engines consider both the frequency and the positioning of keywords to determine relevancy, reasoning that if the keywords appear early in the document, or in the headers, this increases the likelihood that the document is on target. For example, one method is to rank hits according to how many times your keywords appear and in which fields they appear (i.e., in headers, titles or plain text). Another method is to determine which documents are most frequently linked to other documents on the Web. The reasoning here is that if other folks consider certain pages important, you should, too.

If you use the advanced query form on AltaVista, you can assign relevance weights to your query terms before conducting a

search. Although this takes some practice, it essentially allows you to have a stronger say in what results you will get back.

As far as the user is concerned, relevancy ranking is critical, and becomes more so as the sheer volume of information on the Web grows. Most of us don't have the time to sift through scores of hits to determine which hyperlinks we should actually explore. The more clearly relevant the results are, the more we're likely to value the search engine.

Information On Meta Tags

Some search engines are now indexing Web documents by the meta tags in the documents' HTML (at the beginning of the document in the so-called "head" tag). What this means is that the Web page author can have some influence over which keywords are used to index the document, and even in the description of the document that appears when it comes up as a search engine hit. This is obviously very important if you are trying to draw people to your website based on how your site ranks in search engines hit lists.

There is no perfect way to ensure that you'll receive a high ranking. Even if you do get a great ranking, there's no assurance that you'll keep it for long. For example, at one period a page from the Spider's Apprentice was the number- one-ranked result on Altavista for the phrase "how search engines work." A few months later, however, it had dropped lower in the listings.

There is a lot of conflicting information out there on meta-tagging. If you're confused it may be because different search engines look at meta tags in different ways. Some rely heavily on meta tags, others don't use them at all. The general opinion seems to be that meta tags are less useful than they were a few years ago, largely because of the high rate of spamdexing (web authors using false and misleading keywords in the meta tags).

Note: Google, currently the most popular search engine, does not index the keyword metatags. Be aware of this if you are optimizing your webpages for the Google engine.

It seems to be generally agreed that the "title" and the "description" meta tags are important to write effectively, since several major search engines use them in their indices. Use relevant keywords in your title, and vary the titles on the different pages that make up your website, in order to target as many keywords as possible. As for the "description" meta tag, some search engines will use it as their short summary of your url, so

make sure your description is one that will entice surfers to your site.

Note: The "description" meta tag is generally held to be the most valuable, and the most likely to be indexed, so pay special attention to this one.

In the keyword tag, list a few synonyms for keywords, or foreign translations of keywords (if you anticipate traffic from foreign surfers). Make sure the keywords refer to, or are directly related to, the subject or material on the page. Do NOT use false or misleading keywords in an attempt to gain a higher ranking for your pages.

The "keyword" meta tag has been abused by some webmasters. For example, a recent ploy has been to put such words "sex" or "mp3" into keyword meta tags, in hopes of luring searchers to one's website by using popular keywords.

The search engines are aware of such deceptive tactics, and have devised various methods to circumvent them, so be careful. Use keywords that are appropriate to your subject, and make sure they appear in the top paragraphs of actual text on your webpage. Many search engine algorithms score the words that appear towards the top of your document more highly than the words that appear towards the bottom. Words that appear in HTML header tags (H1, H2, H3, etc) are also given more weight by some search engines. It sometimes helps to give your page a file name that makes use of one of your prime keywords, and to include keywords in the "alt" image tags.

One thing you should *not* do is use some other company's trademarks in your meta tags. Some website owners have been sued for trademark violations because they've used other company names in the meta tags. I have, in fact, testified as an expert witness in such cases. You do not want the expense of being sued!

Remember that all the major search engines have slightly different policies. If you're designing a website and meta-tagging your documents, we recommend that you take the time to check out what the major search engines say in their help files about how they each use meta tags. You might want to optimize your meta tags for the search engines you believe are sending the most traffic to your site.

Concept-based searching (***The following information is out-dated, but might have historical interest for researchers***)

Excite used to be the best-known general-purpose search engine site on the Web that relies on concept-based searching. It is now effectively extinct.

Unlike keyword search systems, concept-based search systems try to determine what you mean, not just what you say. In the best circumstances, a concept-based search returns hits on documents that are "about" the subject/theme you're exploring, even if the words in the document don't precisely match the words you enter into the query.

How did this method work? There are various methods of building clustering systems, some of which are highly complex, relying on sophisticated linguistic and artificial intelligence theory that we won't even attempt to go into here. Excite used to a numerical approach. Excite's software determines meaning by calculating the frequency with which certain important words appear. When several words or phrases that are tagged to signal a particular concept appear close to each other in a text, the search engine concludes, by statistical analysis, that the piece is "about" a certain subject.

For example, the word heart, when used in the medical/health context, would be likely to appear with such words as coronary, artery, lung, stroke, cholesterol, pump, blood, attack, and arteriosclerosis. If the word heart appears in a document with others words such as flowers, candy, love, passion, and valentine, a very different context is established, and a concept-oriented search engine returns hits on the subject of romance.

5.6.3 Architecture:

Most of Google is implemented in C or C++ for efficiency and can run in either Solaris or Linux.

In Google, the web crawling (downloading of web pages) is done by several distributed crawlers. There is a URLserver that sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the storeserver. The storeserver then compresses and stores the web pages into a repository. Every web page has an associated ID number called a docID which is assigned whenever a new URL is parsed out of a web page. The indexing function is performed by the indexer and the sorter. The indexer performs a number of functions. It reads the repository, uncompresses the documents, and parses them. Each document is converted into a set of word occurrences called hits. The hits record the word, position in document, an approximation of font size, and capitalization. The indexer distributes these hits into a set of "barrels", creating a partially sorted forward index. The indexer performs another important function. It parses out all the links in every web page and stores important information about them in an anchors file. This file contains enough information to determine where each link points from and to, and the text of the link.

The URLresolver reads the anchors file and converts relative URLs into absolute URLs and in turn into docIDs. It puts the anchor text into the forward index, associated with the docID that the anchor points to. It also generates a database of links which are pairs of docIDs. The links database is used to compute PageRanks for all the documents.

The sorter takes the barrels, which are sorted by docID and resorts them by wordID to generate the inverted index. This is done in place so that little temporary space is needed for this operation. The sorter also produces a list of wordIDs and offsets into the inverted index. A program called DumpLexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. The searcher is run by a web server and uses the lexicon built by DumpLexicon together with the inverted index and the PageRanks to answer queries.

5.6.4 Ranking of web pages:

Early search engines, such as the original Excite or Altavista search engines, ranked pages based on page content and formatting and based on the content of metadata tags. Special attention was paid to bold text or to keywords and summaries in the head of the HTML document.

For several years, many webmasters packed their pages with bogus material to trick the search engines into giving them a high ranking. Web pages containing a single word repeated 10,000 times, but displayed white on white so the user would not see it and the search engine would, were very common! As a result, search engines no longer trust the content of your page to determine its rank, only its relevance to a particular topic. There is very little to be gained by setting keyword and summary metadata in your pages. Of course you still should be sure to state the purpose of your page in all of the likely ways people may search for it, because search engines still must use the content of your page to determine its *relevance*. If your page does not have the keyword the user is searching for, it will not come up on that search at all.

Things changed for the better when Google arrived. Google ranks by how frequently other sites link to you, not by what you do or don't say on your site, although they do of course use what is on your site to determine whether your page is of relevance. When Google arrived, the Google rank of a site was essentially based on its true popularity, as measured by the number of sites that thought it worthwhile to link to it.

Google's techniques are ever-changing, and it is expected that they will find ways to discount the effect of most such links;

websites must distinguish them well enough from actual content that users are not annoyed or offended, and there is a likelihood that Google will be able to adapt and take placement on the page and other factors into account when evaluating the importance of links.

Other modern search engines use their own techniques which are not made known to the public.

5.6.5 The search engine industry:

For webmasters, the major search engines are the most important places to be listed, because they can potentially generate so much traffic.

For searchers, well-known, commercially-backed search engines generally mean more dependable results. These search engines are more likely to be well-maintained and upgraded when necessary, to keep pace with the growing web.

Google

<http://www.google.com>

Voted four times Most Outstanding Search Engine by Search Engine Watch readers, Google has a well-deserved reputation as the top choice for those searching the web. The crawler-based service provides both comprehensive coverage of the web along with great relevancy. It's highly recommended as a first stop in your hunt for whatever you are looking for.

Google provides the option to find more than web pages, however. Using on the top of the search box on the Google home page, you can easily seek out images from across the web, discussions that are taking place on Usenet newsgroups, locate news information or perform product searching. Using the More link provides access to human-compiled information from the Open Directory, catalog searching and other services.

Google is also known for the wide range of features it offers, such as cached links that let you "resurrect" dead pages or see older versions of recently changed ones. It offers excellent spell checking, easy access to dictionary definitions, integration of stock quotes, street maps, telephone numbers and more. In addition to Google's unpaid editorial results, the company also operates its own advertising programs. The cost-per-click AdWords program places ads on Google was originally a Stanford University project by students Larry Page and Sergey Brin called BackRub. By 1998, the name had been changed to Google, and the project jumped off

campus and became the private company Google. It remains privately held today.

Yahoo

<http://www.yahoo.com>

Launched in 1994, Yahoo is the web's oldest "directory," a place where human editors organize web sites into categories. However, in October 2002, Yahoo made a giant shift to crawler-based listings for its main results. These came from Google until February 2004. Now, Yahoo uses its own search technology. Learn more in this recent review from our SearchDay newsletter, which also provides some updated submission details.

In addition to excellent search results, you can use tabs above the search box on the Yahoo home page to seek images, Yellow Page listings or use Yahoo's excellent shopping search engine. Or visit the Yahoo Search home page, where even more specialized search options are offered.

It's also possible to do a pure search of just the human-compiled Yahoo Directory, which is how the old or "classic" Yahoo used to work. To do this, search from the Yahoo Directory home page, as opposed to the regular Yahoo.com home page. Then you'll get both directory category links ("Related Directory Categories") and "Directory Results," which are the top web site matches drawn from all categories of the Yahoo Directory.

Ask

<http://www.ask.com>

Ask Jeeves initially gained fame in 1998 and 1999 as being the "natural language" search engine that let you search by asking questions and responded with what seemed to be the right answer to everything.

In reality, technology wasn't what made Ask Jeeves perform so well. Behind the scenes, the company at one point had about 100 editors who monitored search logs. They then went out onto the web and located what seemed to be the best sites to match the most popular queries.

In 1999, Ask acquired Direct Hit, which had developed the world's first "click popularity" search technology. Then, in 2001, Ask acquired Teoma's unique index and search relevancy technology. Teoma was based upon the clustering concept of subject-specific popularity.

Today, Ask depends on crawler-based technology to provide results to its users. These results come from the Teoma algorithm, now known as ExpertRank.

AllTheWeb.com

<http://www.alltheweb.com>

Powered by Yahoo, you may find AllTheWeb a lighter, more customizable and pleasant "pure search" experience than you get at Yahoo itself. The focus is on web search, but news, picture, video, MP3 and FTP search are also offered.

AllTheWeb.com was previously owned by a company called FAST and used as a showcase for that company's web search technology. That's why you sometimes may sometimes hear AllTheWeb.com also referred to as FAST or FAST Search. However, the search engine was purchased by search provider Overture (see below) in late April 2003, then later become Yahoo's property when Yahoo bought Overture. It no longer has a connection with FAST.

AOL Search

<http://aolsearch.aol.com> (internal)

[http://search.aol.com/\(external\)](http://search.aol.com/(external))

AOL Search provides users with editorial listings that come Google's crawler-based index. Indeed, the same search on Google and AOL Search will come up with very similar matches. So, why would you use AOL Search? Primarily because you are an AOL user. The "internal" version of AOL Search provides links to content only available within the AOL online service. In this way, you can search AOL and the entire web at the same time. The "external" version lacks these links. Why wouldn't you use AOL Search? If you like Google, many of Google's features such as "cached" pages are not offered by AOL Search.

Search Engine Watch members have access to the How AOL Search Works page, which provides in-depth coverage of how AOL Search operates and why there may be subtle differences between it and Google.

5.6.6 The enterprise search:

Companies like most others, have a lot of information in a lot of different formats. To help cut through the clutter, Enterprise Search provides an interactive, visual search experience. Visual cues help people find information quickly, while refiners let them drill into the results and discover insights.

People are the company's key asset. The more quickly and easily they can find each other, the more they can share ideas and expertise to solve problems, improve processes, and foster innovation.

People use search in many different ways. With Enterprise Search, everyone's search needs can be met on a single platform. Addition of own vocabulary, tuning relevance, and using each person's specific information to deliver a great search experience are the effects of a Enterprise search engine.

It delivers powerful intranet and people search, right out of the box. On the whole, it is a unique combination of relevance, refinement, and people provides a highly personalized and effective search experience. It Combines the power of FAST with the simplicity and delivers an exceptional intranet and people search experience and a platform for building custom search-driven applications.

With the new features such as refinement panels one can narrow the results of search and navigate to the right content faster. It also helps in capturing knowledge not found in documents by locating the right people and expertise for any business need.

5.7 CASE STUDY

The Anatomy of a Large-Scale
Hypertextual Web Search Engine
Sergey Brin and Lawrence Page
Computer Science Department,
Stanford University, Stanford, CA 94305, USA
sergey@cs.stanford.edu and page@cs.stanford.edu

Abstract:

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at <http://google.stanford.edu/> To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been done on them.

Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago.

This paper provides an in-depth description of our large-scale web search engine -- the first such detailed public description we know of to date. Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical largescale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

1. Introduction

(Note: There are two versions of this paper -- a longer full version and a shorter printed version. The full version is available on the web and the conference CDROM.)

The web creates new challenges for information retrieval. The amount of information on the web is growing rapidly, as well as the number of new users inexperienced in the art of web research. People are likely to surf the web using its link graph, often starting with high quality human maintained indices such as Yahoo! or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines. We have built a large-scale search engine which addresses many of the problems of existing systems. It makes especially heavy use of the additional structure present in hypertext to provide much higher quality search results. We chose our system name, Google, because it is a common spelling of googol, or 10^{100} and fits well with our goal of building very large-scale search engines.

1.1 Web Search Engines -- Scaling Up: 1994 – 2000:

Search engine technology has had to scale dramatically to keep up with the growth of the web. In 1994, one of the first web search engines, the World Wide Web Worm (WWWW) [McBryan 94] had an index of 110,000 web pages and web accessible documents. As of November, 1997, the top search engines claim to index from 2 million (WebCrawler) to 100 million web documents (from Search Engine Watch). It is foreseeable that by the year 2000, a comprehensive index of the Web will contain over a billion documents. At the same time, the number of queries search engines handle has grown incredibly too. In March and April 1994, the World Wide Web Worm received an average of about 1500 queries per day. In November 1997, Altavista claimed it handled roughly 20 million queries per day.

With the increasing number of users on the web, and automated systems which query search engines, it is likely that top search engines will handle hundreds of millions of queries per day by the year 2000. The goal of our system is to address many of the problems, both in quality and scalability, introduced by scaling search engine technology to such extraordinary numbers.

1.2. Google: Scaling with the Web:

Creating a search engine which scales even to today's web presents many challenges. Fast crawling technology is needed to gather the web documents and keep them up to date. Storage space must be used efficiently to store indices and, optionally, the documents themselves. The indexing system must process hundreds of gigabytes of data efficiently. Queries must be handled quickly, at a rate of hundreds to thousands per second.

These tasks are becoming increasingly difficult as the Web grows. However, hardware performance and cost have improved dramatically to partially offset the difficulty. There are, however, several notable exceptions to this progress such as disk seek time and operating system robustness. In designing Google, we have considered both the rate of growth of the Web and technological changes. Google is designed to scale well to extremely large data sets. It makes efficient use of storage space to store the index. Its data structures are optimized for fast and efficient access (see section <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm>) 2 hardware performance and cost have improved dramatically to partially offset the difficulty. There are, however, several notable exceptions to this progress such as disk seek time and operating system robustness. In designing Google, we have considered both the rate of growth of the Web and technological changes. Google is designed to scale well to extremely large data sets. It makes efficient use of storage space to store the index. Its data structures are optimized for fast and efficient access (see

section 4.2). Further, we expect that the cost to index and store text or HTML will eventually decline relative to the amount that will be available (see Appendix B). This will result in favorable scaling properties for centralized systems like Google.

1.3 Design Goals:

1.3.1 Improved Search Quality:

Our main goal is to improve the quality of web search engines. In 1994, some people believed that a complete search index would make it possible to find anything easily. According to Best of the Web 1994 -- Navigators, "The best navigation service should make it easy to find almost anything on the Web (once all the data is entered)." However, the Web of 1997 is quite different. Anyone who has used a search engine recently, can readily testify that the completeness of the index is not the only factor in the quality of search results. "Junk results" often wash out any results that a user is interested in. In fact, as of November 1997, only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results). One of the main causes of this problem is that the number of documents in the indices has been increasing by many orders of magnitude, but the user's ability to look at documents has not.

People are still only willing to look at the first few tens of results. Because of this, as the collection size grows, we need tools that have very high precision (number of relevant documents returned, say in the top tens of results). Indeed, we want our notion of "relevant" to only include the very best documents since there may be tens of thousands of slightly relevant documents. This very high precision is important even at the expense of recall (the total number of relevant documents the system is able to return). There is quite a bit of recent optimism that the use of more hypertextual information can help improve search and other applications [Marchiori 97] [Spertus 97] [Weiss 96] [Kleinberg 98]. In particular, link structure [Page 98] and link text provide a lot of information for making relevance judgments and quality filtering. Google makes use of both link structure and anchor text (see Sections 2.1 and 2.2).

1.3.2 Academic Search Engine Research:

Aside from tremendous growth, the Web has also become increasingly commercial over time. In 1993, 1.5% of web servers were on .com domains. This number grew to over 60% in 1997. At the same time, search engines have migrated from the academic domain to the commercial. Up until now most search engine development has gone on at companies with little publication of technical details.

This causes search engine technology to remain largely a black art and to be advertising oriented (see Appendix A). With Google, we have a strong goal to push more development and understanding into the academic realm.

Another important design goal was to build systems that reasonable numbers of people can actually use. Usage was important to us because we think some of the most interesting research will involve leveraging the vast amount of usage data that is available from modern web systems. For example, there are many tens of millions of searches performed every day. However, it is very difficult to get this <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 3 Another important design goal was to build systems that reasonable numbers of people can actually use. Usage was important to us because we think some of the most interesting research will involve leveraging the vast amount of usage data that is available from modern web systems. For example, there are many tens of millions of searches performed every day. However, it is very difficult to get this data, mainly because it is considered commercially valuable.

Our final design goal was to build an architecture that can support novel research activities on large-scale web data. To support novel research uses, Google stores all of the actual documents it crawls in compressed form. One of our main goals in designing Google was to set up an environment where other researchers can come in quickly, process large chunks of the web, and produce interesting results that would have been very difficult to produce otherwise. In the short time the system has been up, there have already been several papers using databases generated by Google, and many others are underway. Another goal we have is to set up a Spacelab-like environment where researchers or even students can propose and do interesting experiments on our large-scale web data.

2. System Features:

The Google search engine has two important features that help it produce high precision results. First, it makes use of the link structure of the Web to calculate a quality ranking for each web page. This ranking is called PageRank and is described in detail in [Page 98]. Second, Google utilizes link to improve search results.

2.1 PageRank: Bringing Order to the Web:

The citation (link) graph of the web is an important resource that has largely gone unused in existing web search engines. We have created maps containing as many as 518 million of these hyperlinks, a significant sample of the total. These maps allow rapid calculation of a web page's "PageRank", an objective measure of its citation importance that corresponds well with people's subjective idea of importance. Because of this correspondence, PageRank is an excellent way to prioritize the results of web keyword searches. For most popular subjects, a simple text matching search that is restricted to web page titles performs admirably when PageRank prioritizes the results (demo available at google.stanford.edu). For the type of full text searches in the main Google system, PageRank also helps a great deal.

2.1.1 Description of PageRank Calculation:

Academic citation literature has been applied to the web, largely by counting citations or backlinks to a given page. This gives some approximation of a page's importance or quality. PageRank extends this idea by not counting links from all pages equally, and by normalizing by the number of links on a page. PageRank is defined as follows:

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 4

We assume page A has pages $T_1 \dots T_n$ which point to it (i.e., are citations). The parameter d is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details about d in the next section. Also $C(A)$ is defined as the number of links going out of page A. The PageRank of a page A is given as follows:

$$PR(A) = (1-d) + d (PR(T_1)/C(T_1) + \dots + PR(T_n)/C(T_n))$$

Note that the PageRanks form a probability distribution over web pages, so the sum of all web pages' PageRanks will be one.

PageRank or $PR(A)$ can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web. Also, a PageRank for 26 million web pages can be computed in a few hours on a medium size workstation. There are many other details which are beyond the scope of this paper.

2.1.2 Intuitive Justification:

PageRank can be thought of as a model of user behavior. We assume there is a "random surfer" who is given a web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page. The probability that the random surfer visits a page is its PageRank. And, the d damping factor is the probability at each page the "random surfer" will get bored and request another random page. One important variation is to only add the damping factor d to a single page, or a group of pages. This allows for personalization and can make it nearly impossible to deliberately mislead the system in order to get a higher ranking.

Another intuitive justification is that a page can have a high PageRank if there are many pages that point to it, or if there are some pages that point to it and have a high PageRank. Intuitively, pages that are well cited from many places around the web are worth looking at. Also, pages that have perhaps only one citation from something like the Yahoo! homepage are also generally worth looking at. If a page was not high quality, or was a broken link, it is quite likely that Yahoo's homepage would not link to it. PageRank handles both these cases and everything in between by recursively propagating weights through the link structure of the web.

2.2 Anchor Text:

The text of links is treated in a special way in our search engine. Most search engines associate the text of a link with the page that the link is on. In addition, we associate it with the page the link points to. This has several advantages. First, anchors often provide more accurate descriptions of web pages than the pages themselves. Second, anchors may exist for documents which cannot be indexed by a text-based search engine, such as images, programs, and databases. This makes it possible to return web pages which have not actually been crawled.

Note that pages that have not been crawled can cause problems, since they are never checked for validity before being returned to the user. In this case, the search engine can even return a page that never actually existed, but had hyperlinks pointing to it. However, it is possible to sort the results, so that this particular <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> engines associate the text of a link with the page that the link is on. In addition, we associate it with the page the link points to. This has several advantages. First, anchors often provide more accurate descriptions of web pages than the pages themselves. Second, anchors may exist for documents which cannot be indexed by a text-based search engine, such as images, programs, and databases. This makes it possible to return web pages which have not actually been crawled. Note that pages that

have not been crawled can cause problems, since they are never checked for validity before being returned to the user. In this case, the search engine can even return a page that never actually existed, but had hyperlinks pointing to it. However, it is possible to sort the results, so that this particular problem rarely happens.

This idea of propagating anchor text to the page it refers to was implemented in the World Wide Web Worm [McBryan 94] especially because it helps search nontext information, and expands the search coverage with fewer downloaded documents. We use anchor propagation mostly because anchor text can help provide better quality results. Using anchor text efficiently is technically difficult because of the large amounts of data which must be processed. In our current crawl of 24 million pages, we had over 259 million anchors which we indexed.

2.3 Other Features:

Aside from PageRank and the use of anchor text, Google has several other features. First, it has location information for all hits and so it makes extensive use of proximity in search. Second, Google keeps track of some visual presentation details such as font size of words. Words in a larger or bolder font are weighted higher than other words. Third, full raw HTML of pages is available in a repository.

3. Related Work:

Search research on the web has a short and concise history. The World Wide Web Worm (WWW) [McBryan 94] was one of the first web search engines. It was subsequently followed by several other academic search engines, many of which are now public companies. Compared to the growth of the Web and the importance of search engines there are precious few documents about recent search engines [Pinkerton 94].

According to Michael Mauldin (chief scientist, Lycos Inc) [Mauldin], "the various services (including Lycos) closely guard the details of these databases". However, there has been a fair amount of work on specific features of search engines. Especially well represented is work which can get results by post-processing the results of existing commercial search engines, or produce small scale "individualized" search engines. Finally, there has been a lot of research on information retrieval systems, especially on well controlled collections.

In the next two sections, we discuss some areas where this research needs to be extended to work better on the web.

3.1 Information Retrieval

Work in information retrieval systems goes back many years and is well developed [Witten 94]. However, most of the research on information retrieval systems is on small well controlled homogeneous collections such as collections of scientific papers or news stories on a related topic. Indeed, the primary benchmark for information retrieval, the Text Retrieval Conference [TREC 96], uses a fairly small, well controlled collection for their benchmarks. The "Very Large Corpus" benchmark is only 20GB compared to the 147GB from our crawl of 24 million web pages. Things that work well on TREC often do not produce good results on the web. For example, the standard vector space model tries to return the document

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 6].

However, most of the research on information retrieval systems is on small well controlled homogeneous collections such as collections of scientific papers or news stories on a related topic. Indeed, the primary benchmark for information retrieval, the Text Retrieval Conference [TREC 96], uses a fairly small, well controlled collection for their benchmarks. The "Very Large Corpus" benchmark is only 20GB compared to the 147GB from our crawl of 24 million web pages. Things that work well on TREC often do not produce good results on the web. For example, the standard vector space model tries to return the document that most closely approximates the query, given that both query and document are vectors defined by their word occurrence.

On the web, this strategy often returns very short documents that are the query plus a few words. For example, we have seen a major search engine return a page containing only "Bill Clinton Sucks" and picture from a "Bill Clinton" query. Some argue that on the web, users should specify more accurately what they want and add more words to their query. We disagree vehemently with this position. If a user issues a query like "Bill Clinton" they should get reasonable results since there is a enormous amount of high quality information available on this topic. Given examples like these, we believe that the standard information retrieval work needs to be extended to deal effectively with the web.

3.2 Differences Between the Web and Well Controlled Collections:

The web is a vast collection of completely uncontrolled heterogeneous documents. Documents on the web have extreme variation internal to the documents, and also in the external meta information that might be available. For example, documents differ internally in their language (both human and programming), vocabulary (email addresses, links, zip codes, phone numbers,

product numbers), type or format (text, HTML, PDF, images, sounds), and may even be machine generated (log files or output from a database). On the other hand, we define external meta information as information that can be inferred about a document, but is not contained within it.

Examples of external meta information include things like reputation of the source, update frequency, quality, popularity or usage, and citations. Not only are the possible sources of external meta information varied, but the things that are being measured vary many orders of magnitude as well. For example, compare the usage information from a major homepage, like Yahoo's which currently receives millions of page views every day with an obscure historical article which might receive one view every ten years.

Clearly, these two items must be treated very differently by a search engine. Another big difference between the web and traditional well controlled collections is that there is virtually no control over what people can put on the web.

Couple this flexibility to publish anything with the enormous influence of search engines to route traffic and companies which deliberately manipulating search engines for profit become a serious problem. This problem that has not been addressed in traditional closed information retrieval systems. Also, it is interesting to note that metadata efforts have largely failed with web search engines, because any text on the page which is not directly represented to the user is abused to manipulate search engines. There are even numerous companies which specialize in manipulating search engines for profit.

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 7 traditional closed information retrieval systems. Also, it is interesting to note that metadata efforts have largely failed with web search engines, because any text on the page which is not directly represented to the user is abused to manipulate search engines. There are even numerous companies which specialize in manipulating search engines for profit.

4. System Anatomy

First, we will provide a high level discussion of the architecture. Then, there is some in-depth descriptions of important data structures. Finally, the major applications:

Crawling, indexing, and searching will be examined in depth.
Figure 1. High Level Google Architecture

4.1 Google Architecture:

Overview:

In this section, we will give a high level overview of how the whole system works as pictured in Figure 1. Further sections will discuss the applications and data structures not mentioned in this section. Most of Google is implemented in C or C++ for efficiency and can run in either Solaris or Linux.

In Google, the web crawling (downloading of web pages) is done by several distributed crawlers. There is a URLserver that sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the storeserver. The storeserver then compresses and stores the web pages into a repository. Every web page has an associated ID number called a docID which is assigned whenever a new URL is parsed out of a web page. The indexing function is performed by the indexer and the sorter. The indexer performs a number of functions. It reads the repository, uncompresses the documents, and parses them. Each document is converted into a set of word occurrences called hits.

The hits record the word, position in document, an approximation of font size, and capitalization. The indexer distributes these hits into a set of "barrels", creating a partially sorted forward index. The indexer performs another important function. It parses out all the links in every web page and stores important information about them in an anchors file. This file contains enough information to determine where each link points from and to, and the text of the link.

The URLresolver reads the anchors file and converts relative URLs into absolute URLs and in turn into docIDs. It puts the anchor text into the forward index, associated with the docID that the anchor points to. It also generates a database of links which are pairs of docIDs. The links database is used to compute PageRanks for all the documents.

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 8 The URLresolver reads the anchors file and converts relative URLs into absolute URLs and in turn into docIDs. It puts the anchor text into the forward index, associated with the docID that the anchor points to. It also generates a database of links which are pairs of docIDs. The links database is used to compute PageRanks for all the documents.

The sorter takes the barrels, which are sorted by docID (this is a simplification, see Section 4.2.5), and resorts them by wordID to generate the inverted index. This is done in place so that little temporary space is needed for this operation. The sorter also produces a list of wordIDs and offsets into the inverted index. A program called DumpLexicon takes this list together with the

lexicon produced by the indexer and generates a new lexicon to be used by the searcher. The searcher is run by a web server and uses the lexicon built by DumpLexicon together with the inverted index and the PageRanks to answer queries.

4.2 Major Data Structures:

Google's data structures are optimized so that a large document collection can be crawled, indexed, and searched with little cost. Although, CPUs and bulk input output rates have improved dramatically over the years, a disk seek still requires about 10 ms to complete. Google is designed to avoid disk seeks whenever possible, and this has had a considerable influence on the design of the data structures.

4.2.1 BigFiles:

BigFiles are virtual files spanning multiple file systems and are addressable by 64 bit integers. The allocation among multiple file systems is handled automatically. The BigFiles package also handles allocation and deallocation of file descriptors, since the operating systems do not provide enough for our needs. BigFiles also support rudimentary compression options.

4.2.2 Repository:

Figure 2. Repository Data Structure The repository contains the full HTML of every web page. Each page is compressed using zlib (see RFC1950). The choice of compression technique is a tradeoff between speed and compression ratio. We chose zlib's speed over a significant improvement in compression offered by bzip. The compression rate of bzip was approximately 4 to 1 on the repository as compared to zlib's 3 to 1 compression. In the repository, the documents are stored one after the other and are prefixed by docID, length, and URL as can be seen in Figure 2. The repository requires no other data structures to be used in order to access it. This helps with data consistency and makes development much easier; we can rebuild all the other data structures from only the repository and a file which lists crawler errors.

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 9

The compression rate of bzip was approximately 4 to 1 on the repository as compared to zlib's 3 to 1 compression. In the repository, the documents are stored one after the other and are prefixed by docID, length, and URL as can be seen in Figure 2. The repository requires no other data structures to be used in order to access it. This helps with data consistency and makes development

much easier; we can rebuild all the other data structures from only the repository and a file which lists crawler errors.

4.2.3 Document Index:

The document index keeps information about each document. It is a fixed width ISAM (Index sequential access mode) index, ordered by docID. The information stored in each entry includes the current document status, a pointer into the repository, a document checksum, and various statistics. If the document has been crawled, it also contains a pointer into a variable width file called docinfo which contains its URL and title. Otherwise the pointer points into the URLlist which contains just the URL. This design decision was driven by the desire to have a reasonably compact data structure, and the ability to fetch a record in one disk seek during a search. Additionally, there is a file which is used to convert URLs into docIDs. It is a list of URL checksums with their corresponding docIDs and is sorted by checksum. In order to find the docID of a particular URL, the URL's checksum is computed and a binary search is performed on the checksums file to find its docID. URLs may be converted into docIDs in batch by doing a merge with this file. This is the technique the URLresolver uses to turn URLs into docIDs. This batch mode of update is crucial because otherwise we must perform one seek for every link which assuming one disk would take more than a month for our 322 million link dataset.

4.2.4 Lexicon:

The lexicon has several different forms. One important change from earlier systems is that the lexicon can fit in memory for a reasonable price. In the current implementation we can keep the lexicon in memory on a machine with 256 MB of main memory. The current lexicon contains 14 million words (though some rare words were not added to the lexicon). It is implemented in two parts -- a list of the words (concatenated together but separated by nulls) and a hash table of pointers. For various functions, the list of words has some auxiliary information which is beyond the scope of this paper to explain fully.

4.2.5 Hit Lists:

A hit list corresponds to a list of occurrences of a particular word in a particular document including position, font, and capitalization information. Hit lists account for most of the space used in both the forward and the inverted indices. Because of this, it is important to represent them as efficiently as possible. We considered several alternatives for encoding position, font, and capitalization -- simple encoding (a triple of integers), a compact encoding (a hand optimized allocation of bits), and Huffman coding.

In the end we chose a hand optimized compact encoding since it required far less space than the simple encoding and far less bit manipulation than Huffman coding. The details of the hits are shown in Figure 3. 10 of this, it is important to represent them as efficiently as possible. We considered several alternatives for encoding position, font, and capitalization – simple encoding (a triple of integers), a compact encoding (a hand optimized allocation of bits), and Huffman coding. In the end we chose a hand optimized compact encoding since it required far less space than the simple encoding and far less bit manipulation than Huffman coding. The details of the hits are shown in Figure 3.

Our compact encoding uses two bytes for every hit. There are two types of hits: fancy hits and plain hits. Fancy hits include hits occurring in a URL, title, anchor text, or meta tag. Plain hits include everything else. A plain hit consists of a capitalization bit, font size, and 12 bits of word position in a document (all positions higher than 4095 are labeled 4096). Font size is represented relative to the rest of the document using three bits (only 7 values are actually used because 111 is the flag that signals a fancy hit). A fancy hit consists of a capitalization bit, the font size set to 7 to indicate it is a fancy hit, 4 bits to encode the type of fancy hit, and 8 bits of position.

For anchor hits, the 8 bits of position are split into 4 bits for position in anchor and 4 bits for a hash of the docID the anchor occurs in. This gives us some limited phrase searching as long as there are not that many anchors for a particular word. We expect to update the way that anchor hits are stored to allow for greater resolution in the position and docIDhash fields. We use font size relative to the rest of the document because when searching, you do not want to rank otherwise identical documents differently just because one of the documents is in a larger font.

Figure 3. Forward and Reverse Indexes and the Lexicon The length of a hit list is stored before the hits themselves. To save space, the length of the hit list is combined with the wordID in the forward index and the docID in the inverted index. This limits it to 8 and 5 bits respectively (there are some tricks which allow 8 bits to be borrowed from the wordID). If the length is longer than would fit in that many bits, an escape code is used in those bits, and the next two bytes contain the actual length.

4.2.6 Forward Index:

The forward index is actually already partially sorted. It is stored in a number of barrels (we used 64). Each barrel holds a range of wordID's. If a document contains words that fall into a

particular barrel, the docID is recorded into the barrel, followed by a list of wordID's with hitlists which correspond to those words. This scheme requires slightly more storage because of duplicated docIDs but the difference is very small for a reasonable number of buckets and saves considerable time and coding complexity in the final indexing phase done by the sorter. Furthermore, instead of storing actual wordID's, we store each worded as a relative difference from the minimum wordID that falls into the barrel the <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 11 (we used 64).

Each barrel holds a range of wordID's. If a document contains words that fall into a particular barrel, the docID is recorded into the barrel, followed by a list of wordID's with hitlists which correspond to those words. This scheme requires slightly more storage because of duplicated docIDs but the difference is very small for a reasonable number of buckets and saves considerable time and coding complexity in the final indexing phase done by the sorter. Furthermore, instead of storing actual wordID's, we store each worded as a relative difference from the minimum wordID that falls into the barrel the wordID is in. This way, we can use just 24 bits for the wordID's in the unsorted barrels, leaving 8 bits for the hit list length.

4.2.7 Inverted Index:

The inverted index consists of the same barrels as the forward index, except that they have been processed by the sorter. For every valid wordID, the lexicon contains a pointer into the barrel that wordID falls into. It points to a doclist of docID's together with their corresponding hit lists. This doclist represents all the occurrences of that word in all documents. An important issue is in what order the docID's should appear in the doclist. One simple solution is to store them sorted by docID. This allows for quick merging of different doclists for multiple word queries.

Another option is to store them sorted by a ranking of the occurrence of the word in each document. This makes answering one word queries trivial and makes it likely that the answers to multiple word queries are near the start. However, merging is much more difficult. Also, this makes development much more difficult in that a change to the ranking function requires a rebuild of the index. We chose a compromise between these options, keeping two sets of inverted barrels -- one set for hit lists which include title or anchor hits and another set for all hit lists. This way, we check the first set of barrels first and if there are not enough matches within those barrels we check the larger ones.

4.3 Crawling the Web:

Running a web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application since it involves interacting with hundreds of thousands of web servers and various name servers which are all beyond the control of the system.

In order to scale to hundreds of millions of web pages, Google has a fast distributed crawling system. A single URLserver serves lists of URLs to a number of crawlers (we typically ran about 3). Both the URLserver and the crawlers are implemented in Python. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve web pages at a fast enough pace. At peak speeds, the system can crawl over 100 web pages per second using four crawlers. This amounts to roughly 600K per second of data. A major performance stress is DNS lookup. Each crawler maintains its own DNS cache so it does not need to do a DNS lookup before crawling each document. Each of the hundreds of connections can be in a number of different states: looking up DNS, connecting to host, sending request, and receiving response. These factors make the crawler a complex component of the system. It uses asynchronous IO to manage events, and a number of queues to move page fetches from state to state.

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 12 This amounts to roughly 600K per second of data. A major performance stress is DNS lookup. Each crawler maintains its own DNS cache so it does not need to do a DNS lookup before crawling each document. Each of the hundreds of connections can be in a number of different states: looking up DNS, connecting to host, sending request, and receiving response. These factors make the crawler a complex component of the system. It uses asynchronous IO to manage events, and a number of queues to move page fetches from state to state.

It turns out that running a crawler which connects to more than half a million servers, and generates tens of millions of log entries generates a fair amount of email and phone calls. Because of the vast number of people coming on line, there are always those who do not know what a crawler is, because this is the first one they have seen. Almost daily, we receive an email something like, "Wow, you looked at a lot of pages from my web site. How did you like it?" There are also some people who do not know about the robots exclusion protocol, and think their page should be protected from indexing by a statement like, "This page is copyrighted and should not be indexed", which needless to say is difficult for web crawlers to understand. Also, because of the huge amount of data involved, unexpected things will happen. For

example, our system tried to crawl an online game. This resulted in lots of garbage messages in the middle of their game! It turns out this was an easy problem to fix. But this problem had not come up until we had downloaded tens of millions of pages.

Because of the immense variation in web pages and servers, it is virtually impossible to test a crawler without running it on large part of the Internet. Invariably, there are hundreds of obscure problems which may only occur on one page out of the whole web and cause the crawler to crash, or worse, cause unpredictable or incorrect behavior. Systems which access large parts of the Internet need to be designed to be very robust and carefully tested. Since large complex systems such as crawlers will invariably cause problems, there needs to be significant resources devoted to reading the email and solving these problems as they come up.

4.4 Indexing the Web:

Parsing -- Any parser which is designed to run on the entire Web must handle a huge array of possible errors. These range from typos in HTML tags to kilobytes of zeros in the middle of a tag, non-ASCII characters, HTML tags nested hundreds deep, and a great variety of other errors that challenge anyone's imagination to come up with equally creative ones. For maximum speed, instead of using YACC to generate a CFG parser, we use flex to generate a lexical analyzer which we outfit with its own stack. Developing this parser which runs at a reasonable speed and is very robust involved a fair amount of work.

Indexing Documents into Barrels -- After each document is parsed, it is encoded into a number of barrels. Every word is converted into a wordID by using an in-memory hash table -- the lexicon. New additions to the lexicon hash table are logged to a file. Once the words are converted into wordID's, their occurrences in the current document are translated into hit lists and are written into the forward barrels. The main difficulty with parallelization of the indexing phase is that the lexicon needs to be shared.

Instead of sharing the lexicon, we took the approach of writing a log of all the extra words that were not in a base lexicon, which we fixed at 14 million words. That way multiple indexers can run in parallel and then the small log file of extra words can be processed by one final indexer.

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 13 lexicon hash table are logged to a file. Once the words are converted into wordID's, their occurrences in the current document are translated into hit lists and are written into the forward barrels.

The main difficulty with parallelization of the indexing phase is that the lexicon needs to be shared. Instead of sharing the lexicon, we took the approach of writing a log of all the extra words that were not in a base lexicon, which we fixed at 14 million words.

That way multiple indexers can run in parallel and then the small log file of extra words can be processed by one final indexer. Sorting -- In order to generate the inverted index, the sorter takes each of the forward barrels and sorts it by wordID to produce an inverted barrel for title and anchor hits and a full text inverted barrel. This process happens one barrel at a time, thus requiring little temporary storage. Also, we parallelize the sorting phase to use as many machines as we have simply by running multiple sorters, which can process different buckets at the same time. Since the barrels don't fit into main memory, the sorter further subdivides them into baskets which do fit into memory based on wordID and docID. Then the sorter, loads each basket into memory, sorts it and writes its contents into the short inverted barrel and the full inverted barrel.

4.5 Searching:

The goal of searching is to provide quality search results efficiently. Many of the large commercial search engines seemed to have made great progress in terms of efficiency. Therefore, we have focused more on quality of search in our research, although we believe our solutions are scalable to commercial volumes with a bit more effort. The google query evaluation process is show in Figure 4.

1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4.

To put a limit on response time, once a certain number (currently 40,000) of matching documents are found, the searcher automatically goes to step 8 in Figure 4. This means that it is possible that sub-optimal results would be returned. We are currently investigating other ways to solve this problem.

In the past, we sorted the hits according to PageRank, which seemed to improve the situation.

4.5.1 The Ranking System:

Google maintains much more information about web documents than typical search engines.

Every hitlist includes position, font, and capitalization information. Additionally, we factor in hits from anchor text and the PageRank of the document. Combining all of this information into a rank is difficult. We designed our ranking function so that no particular factor can have too much influence. First, consider the simplest case -- a single word query. In order to rank a document with a single word query, Google looks at that document's hit list for that word. Google considers each hit to be one of several different types (title, anchor, URL, plain

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 14

1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4.

Sort the documents that have matched by rank and return the top k. Figure 4. Google Query Evaluation To put a limit on response time, once a certain number (currently 40,000) of matching documents are found, the searcher automatically goes to step 8 in Figure 4. This means that it is possible that sub-optimal results would be returned. We are currently investigating other ways to solve this problem.

In the past, we sorted the hits according to PageRank, which seemed to improve the situation.

4.5.1 The Ranking System:

Google maintains much more information about web documents than typical search engines. Every hitlist includes position, font, and capitalization information. Additionally, we factor in hits from anchor text and the PageRank of the document. Combining all of this information into a rank is difficult. We designed our ranking function so that no particular factor can have too much influence. First, consider the simplest case -- a single word query. In order to rank a document with a single word query, Google looks at that document's hit list for that word. Google considers each hit to be one of several different types (title, anchor, URL, plain text large font, plain text small font, ...), each of which has its own type-weight. The typeweights make up a vector indexed by type.

Google counts the number of hits of each type in the hit list. Then every count is converted into a count-weight. Count-weights increase linearly with counts at first but quickly taper off so that more than a certain count will not help. We take the dot product of the vector of count-weights with the vector of type-weights to compute an IR score for the document. Finally, the IR score is combined with PageRank to give a final rank to the document.

For a multi-word search, the situation is more complicated. Now multiple hit lists must be scanned through at once so that hits occurring close together in a document are weighted higher than hits occurring far apart. The hits from the multiple hit lists are matched up so that nearby hits are matched together. For every matched set of hits, a proximity is computed. The proximity is based on how far apart the hits are in the document (or anchor) but is classified into 10 different value "bins" ranging from a phrase match to "not even close". Counts are computed not only for every type of hit but for every type and proximity. Every type and proximity pair has a type-prox-weight.

The counts are converted into count-weights and we take the dot product of the count-weights and the typeprox- weights to compute an IR score. All of these numbers and matrices can all be displayed with the search results using a special debug mode. These displays have <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 15 multiple hit lists are matched up so that nearby hits are matched together. For every matched set of hits, a proximity is computed. The proximity is based on how far apart the hits are in the document (or anchor) but is classified into 10 different value "bins" ranging from a phrase match to "not even close". Counts are computed not only for every type of hit but for every type and proximity. Every type and proximity pair has a type-prox-weight. The counts are converted into count-weights and we take the dot product of the count-weights and the typeprox- weights to compute an IR score. All of these numbers and matrices can all be displayed with the search results using a special debug mode.

These displays have been very helpful in developing the ranking system.

4.5.2 Feedback:

The ranking function has many parameters like the type-weights and the typeprox- weights. Figuring out the right values for these parameters is something of a black art. In order to do this, we have a user feedback mechanism in the search engine. A trusted user may optionally evaluate all of the results that are returned. This feedback is saved. Then when we modify the ranking function, we can see the impact of this change on all previous searches which were ranked. Although far from perfect, this gives us some idea of how a change in the ranking function affects the search results.

5 Results and Performance

The most important measure of a search engine is the quality of its search results. While a complete user evaluation is beyond the scope of this paper, our own experience with Google has shown it to produce better results than the major commercial


Query: bill clinton

<http://www.whitehouse.gov/>

100.00%  (no date) (0K)


<http://www.whitehouse.gov/>

[Office of the President](#)

99.67%  (Dec 23 1996) (2K)

http://www.whitehouse.gov/WH/EOP/OP/html/OP_Home.html

[Welcome To The White House](#)

99.98%  (Nov 09 1997) (5K)

<http://www.whitehouse.gov/WH/Welcome.html>

[Send Electronic Mail to the President](#)

99.86%  (Jul 14 1997) (5K)

http://www.whitehouse.gov/WH/Mail/html/Mail_President.html

<mailto:president@whitehouse.gov>

99.98% 

<mailto:President@whitehouse.gov>

99.27% 

[The "Unofficial" Bill Clinton](#)

94.06%  (Nov 11 1997) (14K)

<http://zpub.com/un/un-bc.html>

[Bill Clinton Meets The Shrinks](#)

86.27%  (Jun 29 1997) (63K)

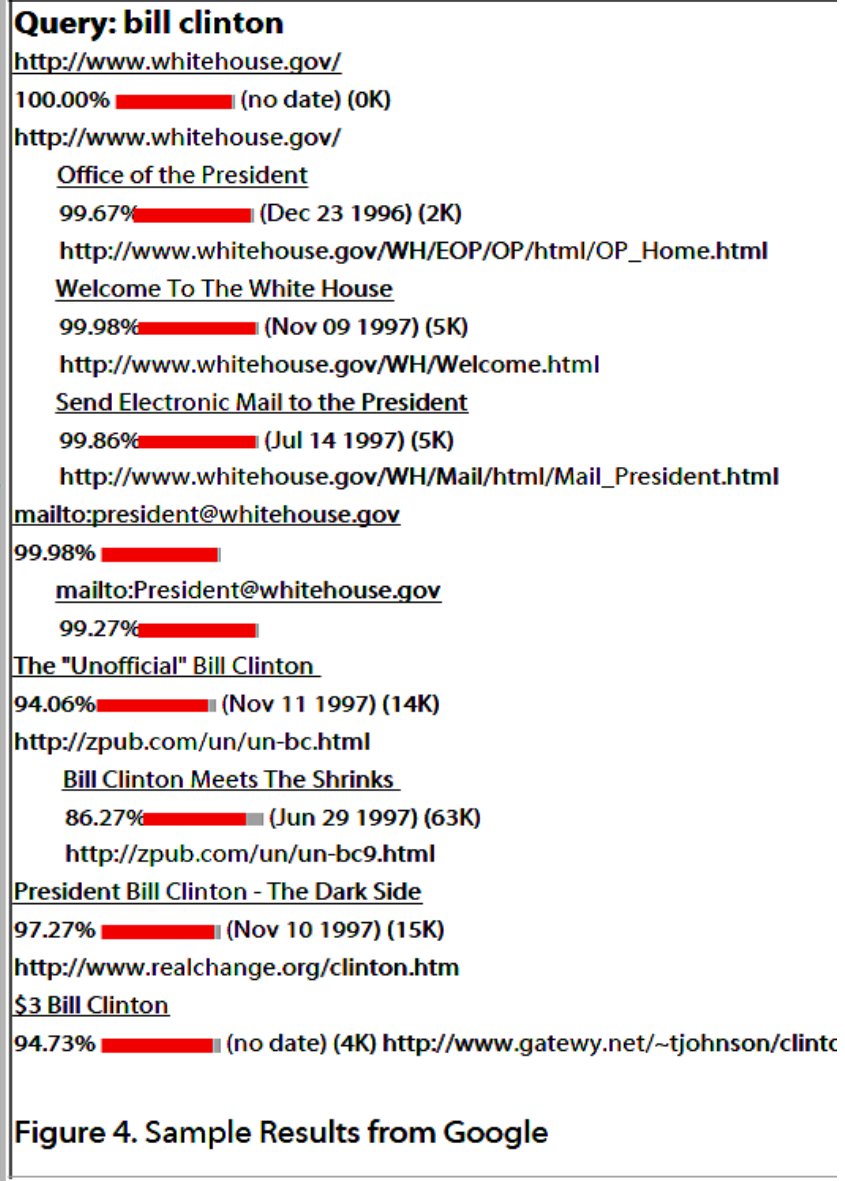
<http://zpub.com/un/un-bc9.html>

[President Bill Clinton - The Dark Side](#)

97.27%  (Nov 10 1997) (15K)

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm>

The most important measure of a search engine is the quality of its search results. While a complete user evaluation is beyond the scope of this paper, our own experience with Google has shown it to produce better results than the major commercial search engines for most searches. As an example which illustrates



the use of PageRank, anchor text, and proximity, Figure 4 shows Google's results for a search on "bill clinton". These results demonstrate some of Google's features. The results are clustered by server. This helps considerably when sifting through result sets. A number of results are from the whitehouse.gov domain which is what one may reasonably expect from such a search. Currently, most major commercial search engines do not return any results from whitehouse.gov, much less the right ones. Notice that there is no title for the first result. This is because it was not crawled. Instead, Google relied on anchor text to determine this was a good answer to the query.

Similarly, the fifth result is an email address which, of course, is not crawlable. It is also a result of anchor text. All of the results are reasonably high quality pages and, at last check, none

were broken links. This is largely because they all have high PageRank. The PageRanks are the percentages in red along with bar graphs. Finally, there are no results about a Bill other than Clinton or about a Clinton other than Bill. This is because we place heavy importance on the proximity of word occurrences. Of course a true test of the quality of a search engine would involve an extensive user study or results analysis which we do not have room for here. Instead, we invite the reader to try Google for themselves at <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm>

All of the results are reasonably high quality pages and, at last check, none were broken links. This is largely because they all have high PageRank. The PageRanks are the percentages in red along with bar graphs. Finally, there are no results about a Bill other than Clinton or about a Clinton other than Bill. This is because we place heavy importance on the proximity of word occurrences. Of course a true test of the quality of a search engine would involve an extensive user study or results analysis which we do not have room for here. Instead, we invite the reader to try Google for themselves at <http://google.stanford.edu>.

5.1 Storage Requirements:

Aside from search quality, Google is designed to scale cost effectively to the size of the Web as it grows. One aspect of this is to use storage efficiently. Table 1 has a breakdown of some statistics and storage requirements of Google. Due to compression the total size of the repository is about 53 GB, just over one third of the total data it stores. At current disk prices this makes the repository a relatively cheap source of useful data. More importantly, the total of all the data used by the search engine requires a comparable amount of storage, about 55 GB. Furthermore, most queries can be answered using just the short inverted index. With better encoding and compression of the Document Index, a high quality web search engine may fit onto a 7GB drive of a new PC.

Storage Statistics

Total Size of Fetched Pages 147.8 GB

Compressed Repository 53.5 GB

Short Inverted Index 4.1 GB

Full Inverted Index 37.2 GB

Lexicon 293 MB

Temporary Anchor Data

(not in total)

6.6 GB

Document Index Incl.

Variable Width Data

9.7 GB

Links Database 3.9 GB

Total Without Repository 55.2 GB

Total With Repository 108.7 GB

Web Page Statistics

Number of Web Pages Fetched 24 million

Number of Urls Seen 76.5 million

Number of Email Addresses 1.7 million

Number of 404's 1.6 million

Table 1. Statistics

5.2 System Performance:

It is important for a search engine to crawl and index efficiently. This way information can be kept up to date and major changes to the system can be tested relatively quickly. For Google, the major operations are Crawling, Indexing, and Sorting. It is difficult to measure how long crawling took overall because disks filled up, name servers crashed, or any number of other problems which stopped the system. In total it took roughly 9 days to download the 26 million pages (including errors).

However, once the system was running smoothly, it ran much faster, downloading the last 11 million pages in just 63 hours, averaging just over 4 million pages per day or 48.5 pages per second. We ran the indexer and the crawler simultaneously. The indexer ran just faster than the crawlers. This is largely because we spent just enough time optimizing the indexer so that it would not be a bottleneck. These optimizations included bulk updates to the document index and placement of critical data structures on the local disk. The <http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 18

Storage Statistics

Total Size of Fetched Pages 147.8 GB

Compressed Repository 53.5 GB

Short Inverted Index 4.1 GB

Full Inverted Index 37.2 GB

Lexicon 293 MB

Temporary Anchor Data

(not in total)

6.6 GB

Document Index Incl.

Variable Width Data

9.7 GB

Links Database 3.9 GB

Total Without Repository 55.2 GB

Total With Repository 108.7 GB

Web Page Statistics

Number of Web Pages Fetched 24 million

Number of Urls Seen 76.5 million

Number of Email Addresses 1.7 million

Number of 404's 1.6 million

Table 1. Statistics

5.2.1 System Performance:

It is important for a search engine to crawl and index efficiently. This way information can be kept up to date and major changes to the system can be tested relatively quickly. For Google, the major operations are Crawling, Indexing, and Sorting. It is difficult to measure how long crawling took overall because disks filled up, name servers crashed, or any number of other problems which stopped the system. In total it took roughly 9 days to download the 26 million pages (including errors). However, once the system was running smoothly, it ran much faster, downloading the last 11 million pages in just 63 hours, averaging just over 4 million pages per day or 48.5 pages per second. We ran the indexer and the crawler simultaneously. The indexer ran just faster than the crawlers. This is largely because we spent just enough time optimizing the indexer so that it would not be a bottleneck. These optimizations included bulk updates to the document index and placement of critical data structures on the local disk. The indexer runs at roughly 54 pages per second. The sorters can be run completely in parallel; using four machines, the whole process of sorting takes about 24 hours.

5.3 Search Performance:

Improving the performance of search was not the major focus of our research up to this point. The current version of Google answers most queries in between 1 and 10 seconds. This time is mostly dominated by disk IO over NFS (since disks are spread over a number of machines). Furthermore, Google does not have any optimizations such as query caching, subindices on common terms,

and other common optimizations. We intend to speed up Google considerably through distribution and hardware, software, and algorithmic improvements. Our target is to be able to handle several hundred queries per second. Table 2 has some sample query times from the current version of Google. They are repeated to show the speedups resulting from cached IO.

<http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm> 19

Initial Query

Same Query

Repeated (IO
mostly cached)

Query

CPU

Time(s)

Total

Time(s)

CPU

Time(s)

Total

Time(s)

al gore 0.09 2.13 0.06 0.06

vice

president

1.77 3.84 1.66 1.80

hard

disks

0.25 4.86 0.20 0.24

search

engines

1.31 9.63 1.16 1.16

Table 2. Search Times

6 Conclusions:

Google is designed to be a scalable search engine. The primary goal is to provide high quality search results over a rapidly growing World Wide Web. Google employs a number of techniques to improve search quality including page rank, anchor text, and proximity information. Furthermore, Google is a complete architecture for gathering web pages, indexing them, and performing search queries over them.



OBJECT DATABASES SYSTEMS

Unit Structure:

- 6.1 Introduction
- 6.2 User Defined ADTs
- 6.3 Structured Types
- 6.4 Object, object identity and references
- 6.5 Inheritance
- 6.6 Database design for ORDBMS
- 6.7 New challenges in implementing ORDBMS
- 6.8 Comparison between OODBMS and ORDBMS

6.1 INTRODUCTION

Information today includes not only data but video, audio, graphs, and photos which are considered complex data types. Relational DBMS fails to support these complex data types because it supports small and fixed collection of data type.

Examples of domain with complex data type are CAD/CAM, multimedia repositories and document management.

These complex data type is supported by object-database system.

An **object database** is a database model in which information is represented in the form of objects as used in object-oriented programming.

Object-database system have 2 path:

Object-Oriented Database Systems:

It is an alternative for relational systems and complex object will play important role in OOD. The Object Database Management Group (ODMG) has developed a standard Object Data Model (ODM) and Object Query Language (OQL), which are the equivalent of the SQL.

Object-Relational Database System:

It is basically bridge between the relational and object oriented Paradigms.

6.2 USER DEFINED ADTs

There is a need to extend the base types provided in RDBMS and SQL as many real-world problems like *jpeg_image* type and *shape* are difficult to express using simple base types.

The combination of an atomic data type and its associated methods is called an abstract data type, or ADT. The concept of abstract data type hides the internal data structures and specifies all possible external operations that can be applied to an object, leading to the concept of encapsulation.

Traditional SQL has inbuilt ADT: integer, string. Object-relational system allows ADT as well as user defines ADT. The label *abstract* is used.

6.3 STRUCTURED TYPES

SQL allows users to define new data types, in addition to the built-in types (e.g., integers).

SQL introduced two type constructors that allow us to define new types with internal structure.

Types defined using type constructors are called structured types.

$\text{ROW}(n_1 \ t_1, \dots, n_n \ t_n)$: A type representing a row, or tuple, of n fields with fields n_1, \dots, n_n of types t_1, \dots, t_n respectively.

base ARRAY $[i]$: A type representing an array of (up to) i base-type items.

Eg:

CREATE TYPE theater_t AS ROW(*tno* integer, *n,arne* text, *address* text, *phone* text)

REF IS SYSTEM GENERATED;

The theater_t type the new ROW data type. In SQL, the ROW type has a special role because every table is a collection of Rows (every table is a set of rows or a multiset of rows). Values of other types can appear only as field values.

Eg:

CREATE TABLE Fillns

**(*filrnno* integer, *title* text, *stats* VARCHAR(25) ARRAY [10]),
director text, *budget* float);**

The *stats* field of table Films illustrates the new ARRAY type. It is an array of upto 10 elements, each of which is of type VARCHAR(25).

➤ **Collection Types**

Other types of constructors are:

- listof(base): A type representing a sequence of base-type items.
- setof (base): A type representing a *set* of base-type items. Sets cannot contain duplicate elements.
- bagof(base): A type representing a *bag* or *multiset* of base-type items.

6.4 OBJECT, OBJECT IDENTITY AND REFERENCES

➤ **Object and object Identity (OID):**

1. An database system provides a unique identity to each independent object stored in database.
2. This unique identity are basically system generated OID.
3. The value of this is not visible to user but is used internally by the system to identify each object uniquely and to create and manage inter object references.
4. The main property of OID is that it should be immutable means the OID value of a particular object should not change. By this, it preserves the identity of the real - world object.
5. Each OID be used only once, and even if object is removed from database, its OID should not be assigned to another object. This property implies that OID should not depend on attribute values of object
6. If the physical address of the object changes, an indirect pointer can be placed at the former address, which gives new physical location of object.

6.5 INHERITANCE

In object-database systems, inheritance can be used in two ways:

- For reusing and refining types and
- For creating hierarchies of collections of similar but not identical objects.

CREATE TYPE theater_t AS ROW(*tno* integer, *name* text, *address* text, *phone* text)

REF IS SYSTEM GENERATED;

Created theaters with type theater_t.

Theater-café is just like Theater just requires additional fields.

Theater-café is just like Theaters but contains additional attributes.

With the help of Inheritance we can get 'specialization'

CREATE TYPE theatercafe_t UNDER theater_t (*menu* text);

Thratercafe_t* inherits the attributes and methods of *theater_t

- Creates a new type, *theatercafe_t*. have same attributes and methods as *theater_t* but it has one more attribute *menu* to type *text* of its own.
- Objects of subtype is also called as object of super type.
- Any operations on supertype (*Thrater_t*) is also applied to subtype (*Thratercafe_t*)
- Easily use quires and methods of supertype to subtype without any changes.
- Along with row type it can be use for atomic type also.

The substitution Principle: Given a supertype *A* and a subtype *B*, it is always possible to substitute an object of type *B* into a legal expression written for object of type *A*, without producing type errors.

➤ Binding Methods:

Suppose we have *image_t* type and subtype *jpeg_image_t*, but *display()* method of supertype cannot work for JPEG images(which is compress), it works only for standard image.

We need to write special *display()* method (for JPEG image in subtype *jpeg_image_t*).

For that we need to register method with that subtype:

```
CREATE FUNCTION display(jpeg_image) RETURNS jpeg_image  
AS EXTERNAL NAME 'a/b/c/jpeg.class' LANGUAGE 'JAVA';
```

Here we are making new method *display()* in subtype *jpeg_image_t* with same name as of supertype *image_t*. This is known as **overloading** the method name.

- Binding a method to an object is process of deciding which method to invoke because When system invoke method *display()* on an object of type *jpeg_image_t* , it is specialized and other is standard one.

6.6 DATABASE DESIGN FOR ORDBMS

Drawbacks of RDBMS:

- We have to write application code in an external language to manipulate a video object in the database. Each probe has an associated sequence of a location readings is obscured, and the sequence information associated with a probe is dispersed across several tuples.
- We are forced to separate the video information from the sequence information for a probe.
- Due to lots of drawbacks in RDBMS in storing and retrieving video streams, better solution is given in ORDBMS.
- Store videos as ADT objects and write methods.
- We can store location sequence for a probe in single tuple, along with the video information which cannot be done in RDBMS. So this will scrap the join queries for both sequence and video information.

Eg:

```
Probes_AllInfo(pid: integer, locseq: location_seq, camera: string,  
video: mpeg_stream)
```

Mpeg_stream type – defined as an ADT, with a method *display()* that takes a start time and an end time and displays the portion of the video recorded during that interval.

```
SELECT display(P.video, 1:10 P.M. May 10 2005, 1:12 P.M. May  
10 2005)
```

```
FROM Probes_AllInfo P
```

```
WHERE P.pid = 10
CREATE TYPE location_seq listof(row (time: timestamp, lat: real,
long: real))
```

```
SELECT P.pid, MIN(P.locseq.time)
```

```
FROM Probes_AllInfo P
```

➤ **Object Identity:**

- When the size of object is larger then OID is important.
- Although reference types and structured types seem similar, they are actually quite different.
- 3 differences:

Reference types	Structured types
It is affected by deletion of objects	not affected by deletion of object.
Object of it changes Value if it is updated	Value is changed only if it is updated directly
Each update is reflected in many place	Updating all copies of an object
Less storage space	Large storage space

➤ **Object identity versus foreign keys:**

An oid can point to an object of theatre_t that is stored anywhere in the database, whereas a foreign key reference is constrained to point to an object in a particular referenced relation.

6.7 NEW CHALLENGES IN IMPLEMENTING ORDBMS

There are several implementation challenges due to enhanced functionality of ORDBMS. Solution are given for some of them.

Storage and Access Methods:

The system must efficiently store ADT objects and structured objects and provide efficient indexed access to both.

Storing large ADT and structured type objects:

- Large ADTs, like BLOBs, require special storage, typically in a different location on disk from the tuples that contain them.
- Disk-based pointers are maintained from the tuples to the objects they contain.

- Flexible disk layout mechanisms are required for structured objects
- A complication arises with array types. Arrays are broken into contiguous chunks, which are then stored in some order on disk.

➤ **Indexing New Types:**

- Users can be allowed for efficient access via indexes.
- It provides efficient indexes for ADT methods and operators on structured objects.
- Template index structure – The Generalized Search Tree(GiST) – allows most of the tree index structures invented so far to be implemented with only a few lines of user-defined ADT code.

QUERY PROCESSING:

ADTs and structured types call for new functionality in processing queries in ORDBMSs.

➤ **User-defined aggregation functions:**

- To register an aggregation function, a user must implement three methods, which we call initialize, iterate and terminate.

➤ **Method Security:**

- ADTs give users the power to add code to the DBMS; this power can be abused.
- A buggy or malicious ADT method can bring down the database server or even corrupt the database.
- One mechanism to prevent problems is to have the user methods be interpreted rather than compiled.
- Typical interpreted languages for this purpose include Java and the procedural portions of SQL:1999.
- An alternate mechanism is to allow user methods to be compiled from a general-purpose programming language, such as C++.

➤ **Method catching:**

- User-defined ADT methods are very expensive.
- During query processing, it may make sense to cache the results of methods, in case they are invoked multiple times with the same argument.

- Within the scope of a single query, one can avoid calling a method twice on duplicate values in a column by either sorting the table on that column or using a hash-based scheme.
- An alternative is to maintain a cache of method inputs and matching outputs as a table in the database.

➤ **Pointer Swizzling:**

When an object O is brought into memory, they check each oid contained in O and replace oids of in-memory objects by in-memory pointers to those objects. This technique is called pointer swizzling, makes references to in-memory objects very fast.

QUERY OPTIMIZATION:

New indexes and query processing techniques widen the choices available to a query optimizer. To handle the new query processing functionality, an optimizer must know about the new functionality and use it appropriately.

➤ **Registering indexes with the optimizer:**

- As new index structures are added to a system-either via external interfaces or built-in template structures like GiSTs-the optimizer must be informed of their existence and their costs of access.
- For a given index structure, the optimizer must know (1) what WHERE-clause conditions are matched by that index, and (2) what the cost of fetching a tuple is for that index.
- Given this information, the optimizer can use any index structure in constructing a query plan.

➤ **Reduction factor and cost estimation for ADT methods:**

- For user defined conditions such as `is_Herbert()`, the optimizer also needs to be able to estimate reduction factors.
- A user who registers a method can also register an auxiliary function to estimate the method's reduction factor.
- To run the method on objects of various sizes and attempt to estimate the method's cost automatically.

➤ **Expensive selection optimization:**

- ORDBMS optimizers must consider carefully how to order selection conditions.

- The best ordering among selections is a function of their costs and reduction factors.

OODBMS:

OODBMS s support collection types make it possible to provide a query language over collections. Indeed, a standard has been developed by the Object Database Management Group and is called Object Query Language.

- OQL is similar to SQL, with a SELECT-FROM-WHERE-style syntax and many of the proposed SQL:1999 extensions.
- OQL supports structured types, including sets,bags,arrays,and lists.
- OQL also supports reference types,path expressions,ADTs and inheritance,type extents, and SQL-style nested queries.
- Object Data Language(ODL) is similar to the DDL subset of SQL but supports the additional features found in OODBMSs, such as ADT definitions.

The ODMG Data Model and ODL:

The ODMG data model is the basis for an OODBMS. A database contains a collection of objects, which are similar to entities in the ER model. Every object has a unique oid, and a database contains collections of objects with similar properties; such a collection is called a class.

The properties of a class are specified using ODL and are of three kinds: attributes, relationships, and methods.

ODL definitions:

Interface Movie

(extent Movies key movieName)

{ attribute date start;

attribute date end;

attribute string moviename;

relationship Set <Theatre> shown At inverse
Theatre::nowShowing;

}

```

Interface Theatre
(extent Theatres key theatre0020Name)
{ attribute string theatreName; date start;
  attribute string address;
  attribute integer ticketPrice;
  relationship Set<Movie> nowshowing inverse Movie::ShownAt;
  float numshowing() raises(errorCountingMovies);
}

```

OQL:

```

SELECT mname: M.movieName, tname: T.theatreName
FROM Movies M, M.shownAt T
WHERE T.numshowing ( ) > 1

```

```

SELECT low, high, avgNum: AVG(SELECT P.T.numshowing()
FROM partition P)
FROM Theatres T
GROUP BY low: T.ticketPrice < 5, high: T.ticketPrice >= 5

```

OQL supports for queries that return collections other than set and multiset.

```

SELECT T.theatreName
FROM Theatres T
ORDER BY T.ticketPrice DESC) [0:4]

```

OQL also supports DISTINCT, HAVING, explicit nesting of subqueries, view definitions, and other SQL features.

6.8 COMPARISON BETWEEN OODBMS AND ORDBMS

➤ **Similarities between OODBMS AND ORDBMS:**

- Both support
 - user-defined ADTs,
 - structured types,
 - object identity and reference types,
 - And inheritance.

- Query language for manipulating collection types.
- Both provide functionality such as concurrency control and recovery.

Note: ORDBMS support an extended form of SQL.

OODBMS supports ODL/OQL.

Difference between OODBMS AND ORDBMS:

Criteria	RDBMS	ODBMS	ORDBMS
Defining standard	SQL2	ODMG-2.0	SQL3 (in process)
Support for object-oriented features	Does not support; It is difficult to map program object to the database	Supports extensively	Limited support; mostly to new data type
Usage	Easy to use	OK for programmers; some SQL access for end users	Easy to use except for some extensions
Support for complex relationships	Does not support abstract datatypes	Supports a wide variety of datatypes and data with complex inter-relationships	Supports Abstract datatypes and complex relationships
Performance	Very good performance	Relatively less performance	Expected to perform very well
Criteria	RDBMS	ODBMS	ORDBMS
Product maturity	Relatively old and so very mature	This concept is few years old and so relatively mature feature	Still in development stage so immature
The use of SQL	Extensive supports SQL	OQL is similar to SQL, but with additional features like Complex objects and object-oriented features	SQL3 is being developed with OO features incorporated in it
Advantages	Its dependence on SQL, relatively simple query optimization hence good performance	It can handle all types of complex applications, reusability of code, less coding	Ability to query complex applications and ability to handle large and complex applications

Disadvantage	Inability to handle complex applications	Low performance due to complex query optimization, inability to support large-scale systems	Low performance in web application
Support vendors from	It is considered to be highly successful so the market size is very large but many vendors are moving towards ORDBMS	Presently lacking vendor support due to vast size of RDBMS market	All major RDBMS vendors are after this so has very good future



7

DATABASE SECURITY

Database security is the system, processes, and procedures that protect a database from unintended activity. Unintended activity can be categorized as authenticated misuse, malicious attacks or inadvertent mistakes made by authorized individuals or processes. *Database security* is also a specialty within the broader discipline of computer security. It entails allowing or disallowing user actions on the database and the objects within it.

Traditionally databases have been protected from external connections by firewalls or routers on the network perimeter with the database environment existing on the internal network opposed to being located within a demilitarized zone. Additional network security devices that detect and alert on malicious database protocol traffic include network intrusion detection systems along with host-based intrusion detection systems.

Database security is more critical as networks have become more open.

Databases provide many layers and types of information security, typically specified in the data dictionary, including:

- Access control
- Auditing
- Authentication
- Encryption
- Integrity controls

Database security can begin with the process of creation and publishing of appropriate security standards for the database environment. The standards may include specific controls for the various relevant database platforms; a set of best practices that cross over the platforms; and linkages of the standards to higher level policies and governmental regulations.

The Three main concerns while designing a secure database application are:

1. **Secrecy:** Information should not be revealed to unauthorized users. For example: a student should not be allowed to view the details of others student.
2. **Integrity:** Only authorized users should be allowed to modify data. For example: An employee will be allowed to view his salary details but not allowed to modify it.
3. **Availability:** Authorized users should not be denied access. For example, an instructor who wishes to change a grade should be allowed to do so.

Access Control:

In an organisation, database has most important data and many users. Many of the users are able to have access to the limited part of database. A DBMS offers two main approaches to access control:

- a) Discretionary access control and
- b) Mandatory access Control.

Discretionary access control:

Discretionary access control (DAC) is a kind of access control defined by the Trusted Computer System Evaluation Criteria as "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are

discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject. Discretionary access control verifies whether the user who is attempting to perform an operation has been granted the required privileges to perform that operation on the database. Discretionary access control regulates all user access to named objects through privileges. A privilege is permission to access a named object in a prescribed manner; for example, permission to query a table. Privileges are granted to users at the discretion of other users--hence the term **Discretionary access control**.

You can perform the following types of discretionary access control:

- Create user roles to control which users can perform operations on which database objects.
- Control who is allowed to create databases.
- Prevent unauthorized users from registering user-defined routines.
- Control whether other users besides the DBSA are allowed to view executing SQL statements.

SQL supports discretionary access control with the help of SQL Commands GRANT AND REVOKE. The GRANT Command allows granting privileges to tables and views of one user to other user.

The Syntax of GRANT Command is as follows:

GRANT privileges ON object to users [WITH GRANT OPTION]

In our case, object is a table or view. Different types of privileges are available, they are:

- a) **SELECT:** This privilege helps us to know what kind of rights we have to access i.e. read all the columns present in the table.
- b) **INSERT (column name):** This privilege helps us to know the right to insert rows with values in the named column of the table. The privileges UPDATE (Column name) and UPDATE are similar.
- c) **DELETE:** This privilege helps us to know the right to delete rows from the table.
- d) **REFERENCES:** This privilege helps us to know the right to define the foreign keys that refer to the specified column of the object i.e. table.

If a user has some privileges with GRANT option he/she can pass those privileges to other user by using GRANT command. The user who actually creates the table has all rights to that

specific table, along with the right to grant privileges to other users. Similarly, the user who creates a view has those privileges on the view that he or she has on every one of the views or base tables. The user creating the view must have the SELECT privilege on each underlying table. The user creating the view will have SELECT privilege with GRANT OPTION only if he or she has SELECT privilege with GRANT OPTION on each and every table. If the view is updateable and the user holds INSERT, DELETE and UPDATE privileges on the underlying table, the user will automatically have the same privileges on the view. Only the owner of schema can execute the data definition statements like CREATE, ALTER and DROP on that schema as the right to execute these statements cannot be granted or revoked.

Mandatory Access Control:

In computer security, **mandatory access control (MAC)** refers to a type of access control by which the operating system constrains the ability of a *subject* or *initiator* to access or generally perform some sort of operation on an *object* or *target*. A well-known implementation of MAC is Multilevel Security (MLS), which, traditionally, has been available mainly on computer and software systems deployed at highly sensitive government organizations such as the intelligence community or the U.S. Department of Defence. In database, the object means tables or views. With MAC, administrator centrally controls the security policy. Users cannot try to breach the policy. For example: Users can not grant access to files that would otherwise be restricted. Implementation of organization wide security policy is allowed by MAC-enabled systems. In MAC, all users are enforced to follow the central policy defined by security administrators. The Trusted Computer System Evaluation Criteria defines MAC as “a means of restricting access to objects based on the sensitivity of the information contained in the objects and the formal authorization of subjects to access information of such sensitivity”. One of the application domains is private banking. In private banking, country laws and regulations often require to limit the amount of data that can be viewed by employee. For example, Swiss Banking laws do not allow a Swiss bank employee located in Toronto to access account information for customers based in Switzerland.

The popular model for MAC is described in terms of objects, subjects, security classes and clearances i.e objects are tables , views, rows etc, subjects are users , programs. Each database object is assigned a security class and each subject is assigned clearance for a security class.

COVERT CHANNELS:

Information can flow from a higher classification level to a lower classification level via indirect means, called Covert Channels, even if DBMS is forcing the mandatory access control. For example, if a transaction accessed data at more than one site in a distributed DBMS, then the actions at two sites must be coordinated properly.

Security for Internet Applications:

When a DBMS is accessed from a secured location, we can rely upon a simple password mechanism for authorizing users. But suppose one user want to purchase some product over the network. From Users point of view, he/she doesn't know whether the company is authorized one or not where is he/she placing the order. Is he/she giving credit card number to the valid organization or not? From Organization's point of view, the organization is not aware whether the customer approaching is a valid user or not etc. With this is we can understand that only simple password mechanism is not enough. Encryption technique is one of the modern techniques of authentication.

Encryption:

The basic idea behind encryption is to apply an encryption algorithm to the data, using a user-specified encryption key. The result of the algorithm is the encrypted data. There is also a decryption algorithm which takes the encrypted data and decryption key as input and returns the original data. Both the algorithms i.e. Encryption and Decryption algorithm are known to the public but the keys either both or anyone is secret. In practice, the public sector uses database encryption to protect citizen privacy and national security.

There are two types of encryption techniques, Symmetric encryption and Asymmetric encryption. In symmetric encryption, the encryption key is also used as the decryption key. It uses an encryption algorithm that consists of character substitutions and permutations. The main weakness of symmetric encryption is that all authorized users must be told the key, increasing the likelihood of its becoming known to an intruder. Another approach to encryption called public-key encryption has become increasingly popular in recent years. Each authorized user has a public encryption key, known to everyone, and a private decryption key, known only to oneself. Since the private decryption keys are known only to their owners, the weakness of symmetric encryption technique is avoided. A central issue for public-key encryption is

how encryption and decryption keys are chosen. Technically, public-key encryption algorithms rely on the existence of one-way functions, whose inverses are computationally very hard to determine.



Syllabus

M.C.A. (Sem-IV), Paper-IV

Advanced Database Techniques

1. Parallel and Distributed databases

- Architecture for Parallel databases
- Parallelizing Individual operations
- Parallel query Evaluation
 - Introduction to DDBMS
 - Architecture of DDBs
 - Storing data in DDBs
 - Distributed catalog management
 - Distributed query processing
 - Distributed concurrency control and recovery
 - Transaction Processing

2. Data warehousing

- Data Marts
- Getting data into the warehouse
- Extraction
- Transformation
- Cleansing
- Loading
- Summarization
- Meta data
- Data warehousing & ERP

- Data warehousing & KM
- Data warehousing & CRM

3. Planning & Project Management

- How is it different?
- Life-cycle approach
- The Development Phases
- Dimensional Analysis
- Dimensional Modeling
- Star Schema
- Snowflake Scheme

4. OLAP

- OLAP Architecture
- Relational OLAP
- Multidimensional OLAP
- Relational vs. Multidimensional OLAP
- Web based OLAP

Major features & functions

- Drill-Down and Roll-Up
- Slice-and-Dice or Rotation
- Implementation techniques for OLAP
- Join Indexes

5. Data Mining

- Introduction
- Data mining algorithms, Clustering, Classification, association rules.
- Knowledge discovery: KDD process
- Decision trees
- Neural Networks

Search Engines

- Characteristics
- Functionality
- Architecture
- Ranking of Web pages
- The search engine Industry
- The Enterprise Search

Case Study: The analysis of a large scale hyper textual search engine.

6. Object Databases Systems

- Introduction
- User-defined ADTs
- Structured types
- Object, object identity and references
- Inheritance
- Database design for ORDBMS
- New Challenges in implementing ORDBMS
 - Storage & access methods
 - Query processing & Optimization

OODBMS

Comparison between OODBMS and ORDBMS

7. Database Security

Term Work: Term work/Assignment: Each candidate will submit a journal in which at least 10 assignments based on the above syllabus and the internal test paper. Test will be graded for 10 marks and assignments will be graded for 15 marks.

References:

1. Raghu Ramakrishnan, Johannes Gerhke, "Database Management Systems" McGraw Hill.
2. Decision support & database system –Efreem G. Mallach.
3. Datawarehousing fundamental – Paulraj Ponniah Wiley.
4. Introduction to data mining with case studies – G.K. Gupta.
5. Elmasri and Navathe, "Fundamentals of Database Systems", Person Education.
6. Korth, Silberchatz, Sudarshan, "Database System Concepts" Mc Graw Hill.

7. Peter Rob and Coronel, "Database Systems, Design, Implementation and Management", Thomson Learning.
8. Data Warehousing (OLAP) S. Nagabhushana New Age.

