USN :										

CMR
INSTITUTE OF
TECHNOLOGY SCHEME & SOLUTION

Internal Assessment Test 1 – SEP 2016

Sub:	Data warehousing & Data mining					Code:	10IS74
Date:	07/ 09/2016	Duration:	90 mins	Max Marks:	50	Sem: VII	Branch: ISE

Note: Answer any five questions:

1. Explain the data warehouse architecture using Operational data stores[10]

logical area for a data warehouse.

While in the ODS, data can be scrubbed, resolved for redundancy and checked for compliance with the corresponding business rules. An ODS can be used for integrating disparate data from multiple sources so that business operations, analysis and reporting can be carried out while business operations are occurring. This is the place where most of the data used in current operation is housed before it's transferred to the data warehouse for longer term storage or archiving.

An ODS is designed for relatively simple queries on small amounts of data (such as finding the status of a customer order), rather than the complex queries on large amounts of data typical of the data warehouse. An ODS is similar to your short term memory in that it stores only very recent information; in comparison, the data warehouse is more like long term memory in that it stores relatively permanent information.

What Is a Data Warehouse?

A data warehouse is a database designed to enable business intelligence activities: it exists to help users understand and enhance their organization's performance. It is designed for query and analysis rather than for transaction processing, and usually contains historical data derived from transaction data, but can include data from other sources. Data warehouses separate analysis workload from transaction workload and enable an organization to consolidate data from several sources. This helps in:

- Maintaining historical records
- Analyzing the data to gain a better understanding of the business and to improve the business

In addition to a relational database, a data warehouse environment can include an extraction, transportation, transformation, and loading (ETL) solution, statistical analysis, reporting, data mining capabilities, client analysis tools, and other applications that manage the process of gathering data, transforming it into useful, actionable information, and delivering it to business users.

To achieve the goal of enhanced business intelligence, the data warehouse works with data collected from multiple sources. The source data may come from internally developed systems, purchased applications, third-party data syndicators and other sources. It may involve transactions, production, marketing, human resources and more. In today's world of big data, the data may be many billions of individual clicks on web sites or the massive data streams from sensors built into complex machinery.

Data warehouses are distinct from online transaction processing (OLTP) systems. With a data warehouse you separate analysis workload from transaction workload. Thus data warehouses are very much read-oriented systems. They have a far higher amount of data reading versus writing and updating. This enables far better analytical performance and avoids impacting your transaction systems. A data warehouse system can be optimized to consolidate data from many sources to achieve a key goal: it becomes your organization's "single source of truth". There is great value in having a consistent source of data that all users can look to; it prevents many disputes and enhances decision-making efficiency.

A data warehouse usually stores many months or years of data to support historical analysis. The data in a data warehouse is typically loaded through an extraction, transformation, and loading (ETL) process from multiple data sources. Modern data warehouses are moving toward an extract, load, transformation (ELT) architecture in which all or most data transformation is performed on the database that hosts the data warehouse. It is important to note that defining the ETL process is a very large part of the design effort of a data warehouse. Similarly, the speed and reliability of ETL operations are the foundation of the data warehouse once it is up and running.

Users of the data warehouse perform data analyses that are often time-related. Examples include consolidation of last year's sales figures, inventory analysis, and profit by product and by customer. But time-focused or not, users want to "slice and dice" their data however they see fit and a well-designed data warehouse will be flexible enough to meet those demands. Users will sometimes need highly aggregated data, and other times they will need to drill down to details. More sophisticated analyses include trend analyses and data mining, which use existing data to forecast trends or predict futures. The data warehouse acts as the underlying engine used by middleware business intelligence environments that serve reports, dashboards and other interfaces to end users.

Although the discussion above has focused on the term "data warehouse", there are two other important terms that need to be mentioned. These are the data mart and the operation data store (ODS).

Operational data stores exist to support daily operations. The ODS data is cleaned and validated, but it is not historically deep: it may be just the data for the current day. Rather than support the historically rich queries that a data warehouse can handle, the ODS gives data warehouses a place to get access to the most current data, which has not yet been loaded into the data warehouse. The ODS may also be used as a source to load the data warehouse. As data warehousing loading techniques have become more advanced, data warehouses may have less need for ODS as a source for loading data. Instead, constant trickle-feed systems can load the data warehouse in near real time.

A common way of introducing data warehousing is to refer to the characteristics of a data warehouse as set forth by William Inmon:

- Subject Oriented
- Integrated
- Nonvolatile
- Time Variant

Subject Oriented

Data warehouses are designed to help you analyze data. For example, to learn more about your company's sales data, you can build a data warehouse that concentrates on sales. Using this data warehouse, you can answer questions such as "Who was our best customer for this item last year?" or "Who is likely to be our best customer next year?" This ability to define a data warehouse by subject matter, sales in this case, makes the data warehouse subject oriented.

Integrated

Integration is closely related to subject orientation. Data warehouses must put data from disparate sources into a consistent format. They must resolve such problems as naming conflicts and inconsistencies among units of measure. When they achieve this, they are said to be integrated.

Nonvolatile

Nonvolatile means that, once entered into the data warehouse, data should not change. This is logical because the purpose of a data warehouse is to enable you to analyze what has occurred.

Time Variant

A data warehouse's focus on change over time is what is meant by the term time variant. In order to discover trends and identify hidden patterns and relationships in business, analysts need large amounts of data. This is very much in contrast to online transaction processing (OLTP) systems, where performance requirements demand that historical data be moved to an archive.

Key Characteristics of a Data Warehouse

The key characteristics of a data warehouse are as follows:

- Data is structured for simplicity of access and high-speed query performance.
- End users are time-sensitive and desire speed-of-thought response times.
- Large amounts of historical data are used.
- Queries often retrieve large amounts of data, perhaps many thousands of rows.
- Both predefined and ad hoc queries are common.
- The data load involves multiple sources and transformations.

In general, fast query performance with high data throughput is the key to a successful data warehouse.

Contrasting OLTP and Data Warehousing Environments

There are important differences between an OLTP system and a data warehouse. One major difference between the types of system is that data warehouses are not exclusively in third normal form (3NF), a type of data normalization common in OLTP environments.

Data warehouses and OLTP systems have very different requirements. Here are some examples of differences between typical data warehouses and OLTP systems:

- Workload

  Data warehouses are designed to accommodate ad hoc queries and data analysis. You might not know the workload of your data warehouse in advance, so a data warehouse should be optimized to perform well for a wide variety of possible query and analytical operations.

  OLTP systems support only predefined operations. Your applications might be specifically tuned or designed to support only these operations.

- Data modifications

  A data warehouse is updated on a regular basis by the ETL process (run nightly or weekly) using bulk data modification techniques. The end users of a data warehouse do not directly update the data warehouse except when using analytical tools, such as data mining, to make predictions with associated probabilities, assign customers to market segments, and develop customer profiles.

  In OLTP systems, end users routinely issue individual data modification statements to the database. The OLTP database is always up to date, and reflects the current state of each business transaction.

- Schema design

Data warehouses often use partially denormalized schemas to optimize query and analytical performance.

OLTP systems often use fully normalized schemas to optimize update/insert/delete performance, and to guarantee data consistency.

- Typical operations

  A typical data warehouse query scans thousands or millions of rows. For example, "Find the total sales for all customers last month."

  A typical OLTP operation accesses only a handful of records. For example, "Retrieve the current order for this customer."

- Historical data

  Data warehouses usually store many months or years of data. This is to support historical analysis and reporting.

  OLTP systems usually store data from only a few weeks or months. The OLTP system stores only historical data as needed to successfully meet the requirements of the current transaction.

Common Data Warehouse Tasks

As an Oracle data warehousing administrator or designer, you can expect to be involved in the following tasks:

- Configuring an Oracle database for use as a data warehouse
- Designing data warehouses
- Performing upgrades of the database and data warehousing software to new releases
- Managing schema objects, such as tables, indexes, and materialized views
- Managing users and security
- Developing routines used for the extraction, transformation, and loading (ETL) processes
- Creating reports based on the data in the data warehouse
- Backing up the data warehouse and performing recovery when necessary
- Monitoring the data warehouse's performance and taking preventive or corrective action as required

In a small-to-midsize data warehouse environment, you might be the sole person performing these tasks. In large, enterprise environments, the job is often divided among several DBAs and designers, each with their own specialty, such as database security or database tuning.

These tasks are illustrated in the following:

- For more information regarding partitioning, see Oracle Database VLDB and Partitioning Guide.
- For more information regarding database security, see Oracle Database Security Guide.
- For more information regarding database performance, see Oracle Database Performance Tuning Guide and Oracle Database SQL Tuning Guide.
- For more information regarding backup and recovery, see Oracle Database Backup and Recovery User's Guide.
- For more information regarding ODI, see Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator.

Data Warehouse Architectures

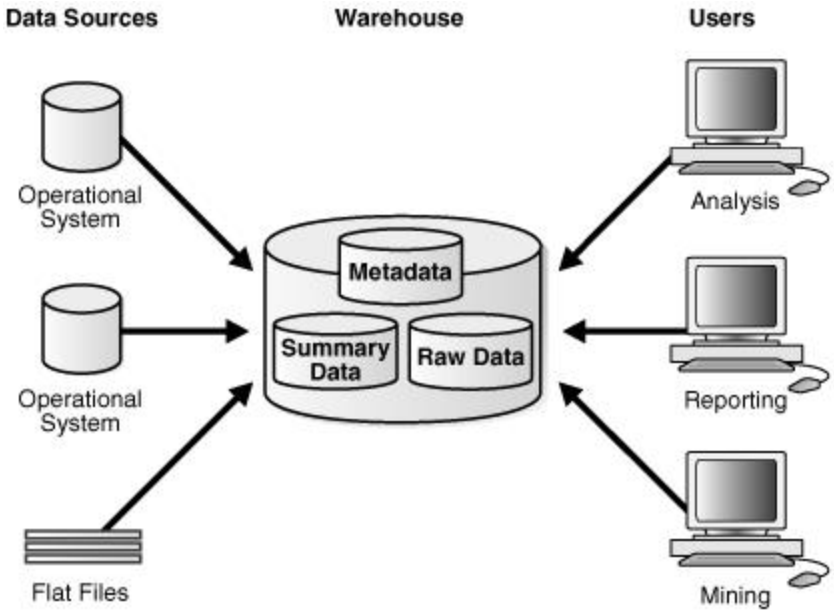
Data warehouses and their architectures vary depending upon the specifics of an organization's situation. Three common architectures are:

- Data Warehouse Architecture: Basic
- Data Warehouse Architecture: with a Staging Area
- Data Warehouse Architecture: with a Staging Area and Data Marts

Data Warehouse Architecture: Basic

Figure 1-1 shows a simple architecture for a data warehouse. End users directly access data derived from several source systems through the data warehouse.

Figure 1-1 Architecture of a Data Warehouse



Description of "Figure 1-1 Architecture of a Data Warehouse"

In Figure 1-1, the metadata and raw data of a traditional OLTP system is present, as is an additional type of data, summary data. Summaries are a mechanism to pre-compute common

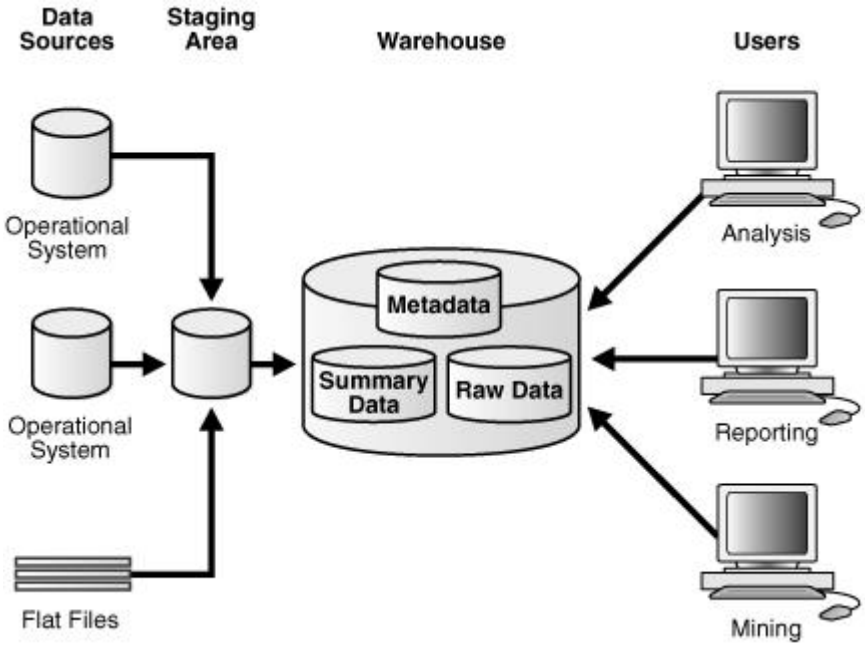
expensive, long-running operations for sub-second data retrieval. For example, a typical data warehouse query is to retrieve something such as August sales. A summary in an Oracle database is called a materialized view.

The consolidated storage of the raw data as the center of your data warehousing architecture is often referred to as an Enterprise Data Warehouse (EDW). An EDW provides a 360-degree view into the business of an organization by holding all relevant business information in the most detailed format.

Data Warehouse Architecture: with a Staging Area

You must clean and process your operational data before putting it into the warehouse, as shown in Figure 1-2. You can do this programmatically, although most data warehouses use a staging area instead. A staging area simplifies data cleansing and consolidation for operational data coming from multiple source systems, especially for enterprise data warehouses where all relevant information of an enterprise is consolidated. Figure 1-2 illustrates this typical architecture.

Figure 1-2 Architecture of a Data Warehouse with a Staging Area



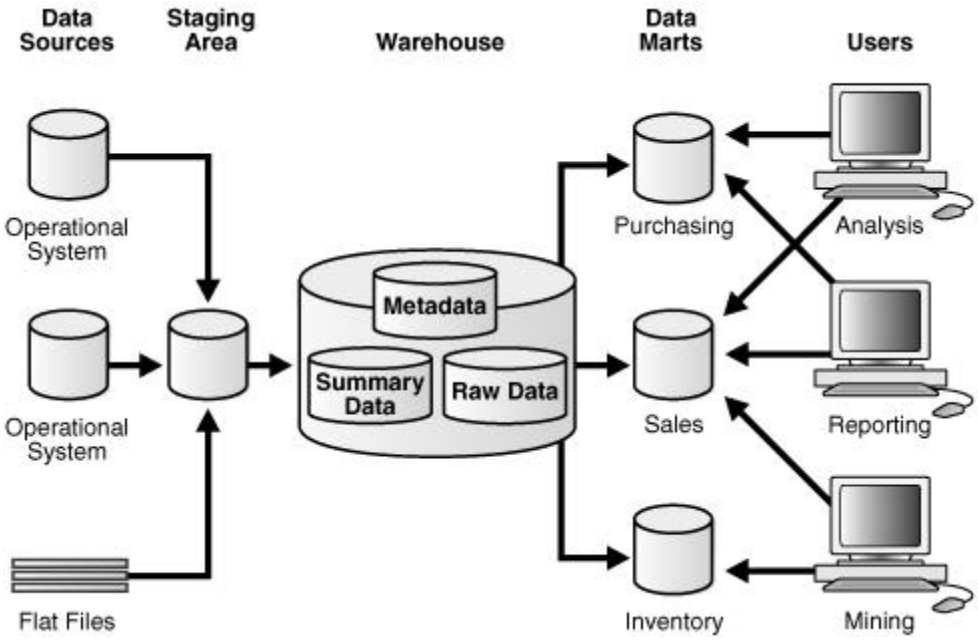
Description of "Figure 1-2 Architecture of a Data Warehouse with a Staging Area"

Data Warehouse Architecture: with a Staging Area and Data Marts

Although the architecture in Figure 1-2 is quite common, you may want to customize your warehouse's architecture for different groups within your organization. You can do this by adding data marts, which are systems designed for a particular line of business. Figure 1-3 illustrates an example where purchasing, sales, and inventories are separated. In this example, a

financial analyst might want to analyze historical data for purchases and sales or mine historical data to make predictions about customer behavior.

Figure 1-3 Architecture of a Data Warehouse with a Staging Area and Data Marts



2. Write short notes on data mart, Meta data, data mining, clean data and data error[10]

Data Cleaning

- Importance
  - "Data cleaning is one of the three biggest problems in data warehousing"—Ralph Kimball
  - "Data cleaning is the number one problem in data warehousing"—DCI survey
- Data cleaning tasks
  - Fill in missing values
  - Identify outliers and smooth out noisy data
  - Correct inconsistent data
  - Resolve redundancy caused by data integration

Missing Data

- Data is not always available
  - E.g., many tuples have no recorded value for several attributes, such as customer income in sales data

- Missing data may be due to
  - Equipment malfunction
  - Inconsistent with other recorded data and thus deleted
  - Data not entered due to misunderstanding
  - Certain data may not be considered important at the time of entry
  - Not register history or changes of the data
- Missing data may need to be inferred.

- How to Handle Missing Data?

  - Ignore the tuple: usually done when class label is missing (assuming the tasks in classification—not effective when the percentage of missing values per attribute varies considerably.
  - Fill in the missing value manually: tedious + infeasible?
  - Fill in it automatically with
    - A global constant: e.g., “unknown”, a new class?!
    - the attribute mean
    - the attribute mean for all samples belonging to the same class: smarter
    - the most probable value: inference-based such as Bayesian formula or decision tree

Noisy Data

- Noise: random error or variance in a measured variable
- Incorrect attribute values may due to
  - faulty data collection instruments
  - data entry problems
  - data transmission problems
  - technology limitation
  - inconsistency in naming convention
- Other data problems which requires data cleaning
  - duplicate records
  - incomplete data
  - inconsistent data

- How to Handle Noisy Data?

  - Binning method:
    - first sort data and partition into (equi-depth) bins

        - then one can smooth by bin means, smooth by bin median, smooth by bin boundaries, etc.
    - Clustering
        - detect and remove outliers
    - Combined computer and human inspection
        - detect suspicious values and check by human (e.g., deal with possible outliers)
    - Regression
        - smooth by fitting the data into regression functions
- A data mart

It serves the same role as a data warehouse, but it is intentionally limited in scope. It may serve one particular department or line of business. The advantage of a data mart versus a data warehouse is that it can be created much faster due to its limited coverage. However, data marts also create problems with inconsistency. It takes tight discipline to keep data and calculation definitions consistent across data marts. This problem has been widely recognized, so data marts exist in two styles. Independent data marts are those which are fed directly from source data. They can turn into islands of inconsistent information. Dependent data marts are fed from an existing data warehouse. Dependent data marts can avoid the problems of inconsistency, but they require that an enterprise-level data warehouse already exist.

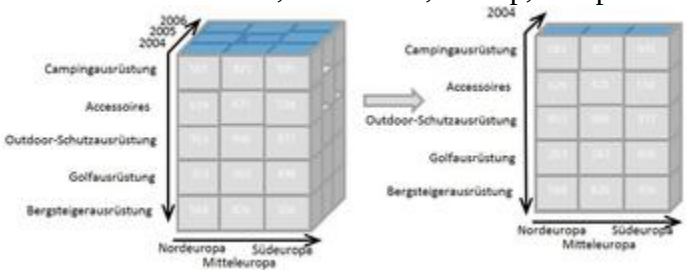
3. Explain with an example of all OLAP operations[10]

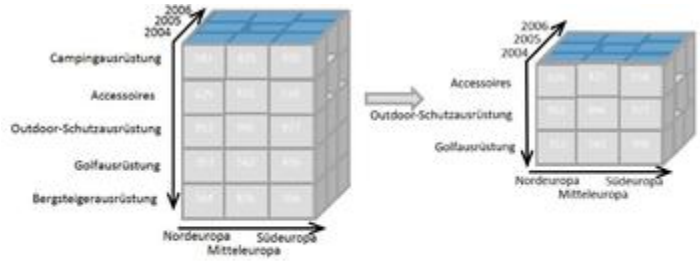
- OLAP data is typically stored in a star schema or snowflake schema in a relational data warehouse or in a special-purpose data management system. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables.

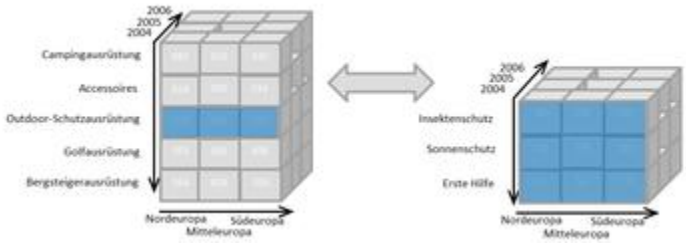
- Hierarchy
- The elements of a dimension can be organized as a hierarchy,[4] a set of parent-child relationships, typically where a parent member summarizes its children. Parent elements can further be aggregated as the children of another parent.[5]
- For example May 2005's parent is Second Quarter 2005 which is in turn the child of Year 2005. Similarly cities are the children of regions; products roll into product groups and individual expense items into types of expenditure.

- Operations
- Conceiving data as a cube with hierarchical dimensions leads to conceptually straightforward operations to facilitate analysis. Aligning the data content with a familiar visualization enhances analyst learning and productivity.[5] The user-initiated process of navigating by calling for page displays interactively, through the specification of slices

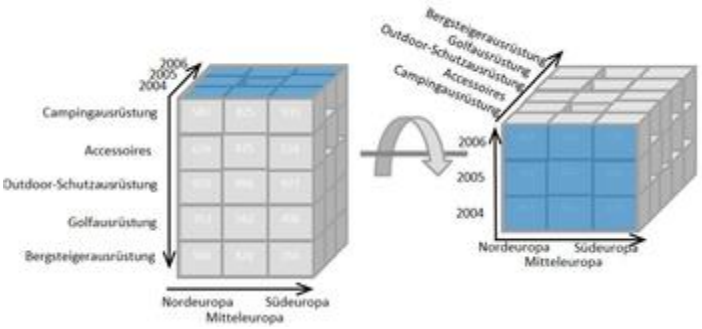
via rotations and drill down/up is sometimes called "slice and dice". Common operations include slice and dice, drill down, roll up, and pivot.

- 

- OLAP slicing
- Slice is the act of picking a rectangular subset of a cube by choosing a single value for one of its dimensions, creating a new cube with one fewer dimension.[5] The picture shows a slicing operation: The sales figures of all sales regions and all product categories of the company in the year 2004 are "sliced" out of the data cube.
- 

- OLAP dicing
- Dice: The dice operation produces a subcube by allowing the analyst to pick specific values of multiple dimensions.[6] The picture shows a dicing operation: The new cube shows the sales figures of a limited number of product categories, the time and region dimensions cover the same range as before.
- 

- OLAP Drill-up and drill-down

- Drill Down/Up allows the user to navigate among levels of data ranging from the most summarized (up) to the most detailed (down).[5] The picture shows a drill-down operation: The analyst moves from the summary category "Outdoor-Schutzausrüstung" to see the sales figures for the individual products.
- Roll-up: A roll-up involves summarizing the data along a dimension. The summarization rule might be computing totals along a hierarchy or applying a set of formulas such as "profit = sales - expenses".[5]
- 

- OLAP pivoting
- Pivot allows an analyst to rotate the cube in space to see its various faces. For example, cities could be arranged vertically and products horizontally while viewing data for a particular quarter. Pivoting could replace products with time periods to see data across time for a single product.[5][7]
- The picture shows a pivoting operation: The whole cube is rota

4. Explain a)OLAP – FASMI Characteristics, b)OLAP Characteristics [10]

The following are the differences between OLAP and OLTP systems.
1. Users: OLTP systems are designed for office workers while the OLAP systems are designedfordecisionmakers.Thereforewhile anOLTPsystem may beaccessed byhundreds or even thousands of users in a large enterprise, an OLAP system is likely to beaccessed only by a select group of managers and may be used only by dozens of users.
2. Functions: OLTP systems are mission-critical. They support day-to-day operations ofan enterprise and are mostly performance and availability driven. These systems carry outsimple repetitive operations. OLAP systems are management-critical to support decisionof an enterprise support functions using analytical investigations. They are morefunctionality driven. These are ad hoc and often much more complex operations.
3. Nature: Although SQL queries often return a set of records, OLTP systems aredesigned to process one record at a time, for example a record related to the customerwho might be on the phone or in the store. OLAP systems are not designed to deal withindividual customer records. Instead they involve queries that deal with many records at atime and provide summary or

aggregate data to a manager. OLAP applications involvedata stored in a data warehouse that has been extracted from many tables and perhapsfrom more than one enterprise database.
4. Design: OLTP database systems are designed to be application-oriented while OLAPsystems are designed to be subject-oriented. OLTP systems view the enterprise data as acollection of tables (perhaps based on an entity-relationship model). OLAP systems viewenterprise information as multidimensional).
5. Data: OLTP systems normally deal only with the current status of information. Forexample, information about an employee who left three years ago may not be available

FASMI Characteristics

In the FASMI characteristics of OLAP systems, the name derived from the first letters of the characteristics are:

Fast: Asnotedearlier,mostOLAPqueriesshould beansweredveryquickly,perhaps within seconds. The performance of an OLAP system has to be like that of asearch engine. If the response takes more than say 20 seconds, the user is likely to moveaway to something else assuming there is a problem with the query. Achieving suchperformance is difficult.The data structuresmust be efficient.The hardware must bepowerful enough for the amount of data and the number of users. Full pre-computation of aggregates helps but is often not practical due to the large number of aggregates fail.

Analytic:An OLAP system mustprovide rich analytic functionalityand it isexpectedthatmostOLAPqueriescan beansweredwithoutanyprogramming.Thesystem should be able to cope with any relevant queries for the application and the user.Oftentheanalysiswillbe usingthevendor'sowntoolsalthoughOLAPsoftwarecapabilities differ widely between products in the market.

Shared: An OLAP system is shared resource although it is unlikely to beshared by hundreds of users. An OLAP system is likely to be accessed only by a selectgroup of managers and may be used merely by dozens of users. Being a shared system,an OLAPsystem should be provide adequate security for confidentiality as well as integrity.

Multidimensional:This isthebasicrequirement.WhateverOLAPsoftware isbeing used, it must provide a multidimensional conceptual view of the data. It is becauseof themultidimensionalview of datathat weoftenreferto thedataas acube. Adimensionoftenhashierarchiesthatshowparent/ child relationshipsbetweenthemembers of a dimension. The multidimensional structure should allow such hierarchies.

Information: OLAP systems usually obtain information from a data warehouse.The system should be able to handle a large amount of input data. The capacity of anOLAP system to handle information and its integration with the data warehouse may becritical.

5. Explain Apriori algorithm with an example. [10]

Apriori Algorithm

A Java applet which combines DIC, Apriori and Probability Based Objected Interestingness Measures can be found here.

Apriori Algorithm: (by Agrawal et al at IBM Almaden Research Centre) can be used to generate all frequent itemset

Pass 1

1. Generate the candidate itemsets in C_1
2. Save the frequent itemsets in L_1

Pass k

1. Generate the candidate itemsets in C_k from the frequent itemsets in L_{k-1}
    1. Join L_{k-1} p with L_{k-1}q, as follows:
       insert into C_k
       select p.item$_1$, p.item$_2$, . . . , p.item$_{k-1}$, q.item$_{k-1}$
       from L_{k-1} p, L_{k-1}q
       where p.item$_1$ = q.item$_1$, . . . p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$
    2. Generate all (k-1)-subsets from the candidate itemsets in C_k
    3. Prune all candidate itemsets from C_k where some (k-1)-subset of the candidate itemset is not in the frequent itemset L_{k-1}
2. Scan the transaction database to determine the support for each candidate itemset in C_k
3. Save the frequent itemsets in L_k

Implementation: A working Apriori Itemset Generation program can be found on the Itemset Implementation page.

Example 1: Assume the user-specified minimum support is 50%

- Given: The transaction database shown below

TID	A	B	C	D	E	F
T_1	1	0	1	1	0	0
T_2	0	1	0	1	0	0
T_3	1	1	1	0	1	0
T_4	0	1	0	1	0	1

- The candidate itemsets in C_2 are shown below

Itemset X	supp(X)
{A,B}	25%
{A,C}	50%
{A,D}	25%
{B,C}	25%
{B,D}	50%
{C,D}	25%

- The frequent itemsets in L_2 are shown below

Itemset X supp(X)
{A,C} 50%
{B,D} 50%

6. Explain FP –Growth tree algorithm with an example.[10]

The FP-Growth Algorithm is an alternative way to find frequent itemsets without using candidate generations, thus improving performance. For so much it uses a divide-and-conquer strategy [17]. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the itemset association information.

In simple words, this algorithm works as follows: first it compresses the input database creating an FP-tree instance to represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity.

In large databases, it's not possible to hold the FP-tree in the main memory. A strategy to cope with this problem is to firstly partition the database into a set of smaller databases (called projected databases), and then construct an FP-tree from each of these smaller databases.

The next subsections describe the FP-tree structure and FP-Growth Algorithm, finally an example is presented to make it easier to understand these concepts.

FP-Tree structure

The frequent-pattern tree (FP-tree) is a compact structure that stores quantitative information about frequent patterns in a database [4].

Han defines the FP-tree as the tree structure defined below [1]:

1. One root labeled as "null" with a set of item-prefix subtrees as children, and a frequent-item-header table (presented in the left side of Figure 1);
2. Each node in the item-prefix subtree consists of three fields:
    1. Item-name: registers which item is represented by the node;
    2. Count: the number of transactions represented by the portion of the path reaching the node;
    3. Node-link: links to the next node in the FP-tree carrying the same item-name, or null if there is none.

1. Each entry in the frequent-item-header table consists of two fields:

1. Item-name: as the same to the node;
2. Head of node-link: a pointer to the first node in the FP-tree carrying the item-name.

Additionally the frequent-item-header table can have the count support for an item. The Figure 1 below show an example of a FP-tree.

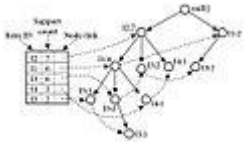
- 

Figure 1: An example of an FP-tree from [17].

The original algorithm to construct the FP-Tree defined by Han in [1] is presented below in Algorithm 1.

Algorithm 1: FP-tree construction

Input: A transaction database DB and a minimum support threshold ?.
Output: FP-tree, the frequent-pattern tree of DB.
Method: The FP-tree is constructed as follows.

1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as FList, the list of frequent items.
2. Create the root of an FP-tree, T, and label it as “null”. For each transaction Trans in DB do the following:

- Select the frequent items in Trans and sort them according to the order of FList. Let the sorted frequent-item list in Trans be [p | P], where p is the first element and P is the remaining list. Call insert tree([p | P], T).
- The function insert tree([p | P], T) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N ’s count by 1; else create a new node N , with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P, N) recursively.

By using this algorithm, the FP-tree is constructed in two scans of the database. The first scan collects and sort the set of frequent items, and the second constructs the FP-Tree.

FP-Growth Algorithm

After constructing the FP-Tree it's possible to mine it to find the complete set of frequent patterns. To accomplish this job, Han in [1] presents a group of lemmas and properties, and thereafter describes the FP-Growth Algorithm as presented below in Algorithm 2.

Algorithm 2: FP-Growth

Input: A database DB, represented by FP-tree constructed according to Algorithm 1, and a minimum support threshold ?.
Output: The complete set of frequent patterns.
Method: call FP-growth(FP-tree, null).
Procedure FP-growth(Tree, a) {
(01) if Tree contains a single prefix path then // Mining single prefix-path FP-tree {
(02) let P be the single prefix-path part of Tree;
(03) let Q be the multipath part with the top branching node replaced by a null root;
(04) for each combination (denoted as ß) of the nodes in the path P do
(05) generate pattern ß ∪ a with support = minimum support of nodes in ß;
(06) let freq pattern set(P) be the set of patterns so generated;
}
(07) else let Q be Tree;
(08) for each item ai in Q do { // Mining multipath FP-tree
(09) generate pattern ß = ai ∪ a with support = ai .support;
(10) construct ß's conditional pattern-base and then ß's conditional FP-tree Tree ß;
(11) if Tree ß ≠ Ø then
(12) call FP-growth(Tree ß , ß);
(13) let freq pattern set(Q) be the set of patterns so generated;
}
(14) return(freq pattern set(P) ∪ freq pattern set(Q) ∪ (freq pattern set(P) × freq pattern set(Q)))
}

When the FP-tree contains a single prefix-path, the complete set of frequent patterns can be generated in three parts: the single prefix-path P, the multipath Q, and their combinations (lines 01 to 03 and 14). The resulting patterns for a single prefix path are the enumerations of its subpaths that have the minimum support (lines 04 to 06). Thereafter, the multipath Q is defined (line 03 or 07) and the resulting patterns from it are processed (lines 08 to 13). Finally, in line 14 the combined results are returned as the frequent patterns found.

An example

This section presents a simple example to illustrate how the previous algorithm works. The original example can be viewed in [18].

Consider the transactions below and the minimum support as 3:

i(t)

1 ABDE

2 BCE

3 ABDE

4 ABCE

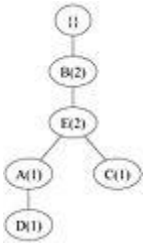
5 ABCDE

6 BCD

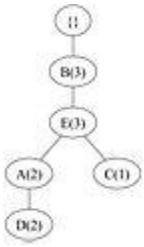
To build the FP-Tree, frequent items support are first calculated and sorted in decreasing order resulting in the following list: { B(6), E(5), A(4), C(4), D(4) }. Thereafter, the FP-Tree is iteratively constructed for each transaction, using the sorted list of items as shown in Figure 2.

- 

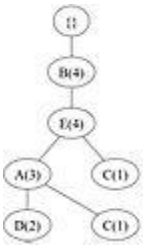
(a) Transaction 1: BEAD

- 

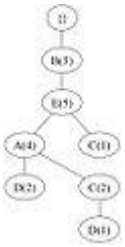
(b) Transaction 2: BEC

- 

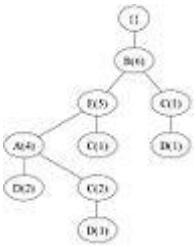
(c) Transaction 3: BEAD

- 

(d) Transaction 4: BEAC

- 

(e) Transaction 5: BEACD

- 

(f) Transaction 6: BCD

Figure 2: Constructing the FP-Tree iteratively.

As presented in Figure 3, the initial call to FP-Growth uses the FP-Tree obtained from the Algorithm 1, presented in Figure 2 (f), to process the projected trees in recursive calls to get the frequent patterns in the transactions presented before.

Using a depth-first strategy the projected trees are determined to items D, C, A, E and B, respectively. First the projected tree for D is recursively processed, projecting trees for DA, DE and DB. In a similar manner the remaining items are processed. At the end of process the frequent itemset is: { DAE, DAEB, DAB, DEB, CE, CEB, CB, AE, AEB, AB, EB }.

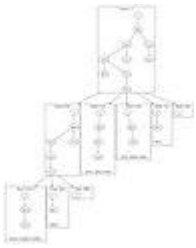
- 

Figure 3: Projected trees and frequent patterns founded by the recursively calls to FP-Growth Algorithm.

FP-Growth Algorithm Variations

As mentioned before, the popularity and efficiency of FP-Growth Algorithm contributes with many studies that propose variations to improve its performance [5] [6] [9] [12] [19] [12] [13] [14] [15] [16]. In this section some of them are briefly described.

DynFP-Growth Algorithm

The DynFP-Growth [13] [5], has focused in improving the FP-Tree algorithm construction based on two observed problems:

1. The resulting FP-tree is not unique for the same "logical" database;
2. The process needs two complete scans of the database.

To solve the first problem Gyorödi C., et al. [13] proposes the usage of a support descending order together with a lexicographic order, ensuring in this way the uniqueness of the resulting FP-tree for different "logically equivalent" databases. To solve the second problem they proposed devising a dynamic FP-tree reordering algorithm, and employing this algorithm whenever a "promotion" to a higher order of at least one item is detected.

An important feature in this approach is that it's not necessary to rebuild the FP-Tree when the actual database is updated. It's only needed to execute the algorithm again taking into consideration the new transactions and the stored FP-Tree.

Another adaptation proposed, because of the dynamic reordering process, is a modification in the original structures, by replacing the single linked list with a doubly linked list for linking the tree nodes to the header and adding a master-table to the same header. See [13] for more details.

FP-Bonsai Algorithm

The FP-Bonsai [6] improve the FP-Growth performance by reducing (pruning) the FP-Tree using the ExAnte data-reduction technique [14]. The pruned FP-Tree was called FP-Bonsai. See [6] for more details.

AFOPT Algorithm

Investigating the FP-Growth algorithm performance Liu proposed the AFOPT algorithm in [12]. This algorithm aims at improving the FP-Growth performance in four perspectives:

- Item Search Order: when the search space is divided, all items are sorted in some order. The number of the conditional databases constructed can differ very much using different items search orders;
- Conditional Database Representation: the traversal and construction cost of a conditional database heavily depends on its representation;
- Conditional Database Construction Strategy: constructing every conditional database physically can be expensive affecting the mining cost of each individual conditional database;
- Tree Traversal Strategy: the traversal cost of a tree is minimal using top-down traversal strategy.

See [12] for more details.

NONORDFP Algorithm

The Nonordfp algorithm [15] [5] was motivated by the running time and the space required for the FP-Growth algorithm. The theoretical difference is the main data structure (FP-Tree), which is more compact and which is not needed to rebuild it for each conditional step. A compact, memory efficient representation of an FP-tree by using Trie data structure, with memory layout that allows faster traversal, faster allocation, and optionally projection was introduced. See [15] for more details.

FP-Growth* Algorithm

This algorithm was proposed by Grahne et al [16] [19], and is based in his conclusion about the usage of CPU time to compute frequent item sets using FP-Growth. They observed that 80% of CPU time was used for traversing FP-Trees [9]. Therefore, they used an array-based data structure combined with the FP-Tree data structure to reduce the traversal time, and incorporates several optimization techniques. See [16] [19] for more details.

PPV, PrePost, and FIN Algorithm

These three algorithms were propsed by Deng et al [20] [21] [22], and are based on three novel data structures called Node-list [20], N-list [21], and Nodeset [22] respectively for facilitating the mining process of frequent itemsets. They are based on a FP-tree with each node encoding with pre-order traversal and post-order traversal. Compared with Node-lists, N-lists and Nodesets are more efficient.

- b) A frequent itemset is an itemset whose support is greater than some user-specified minimum support (denoted L_k, where k is the size of the itemset)

- A candidate itemset is a potentially frequent itemset (denoted C_k, where k is the size of the itemset)