Assignment (Set-1)

Subject code: MI0033

Software Engineering

Q. 1 : Discuss the Objectives & Principles Behind Software Testing.

Ans. **Software Testing Fundamentals:**

Testing presents an interesting anomaly for the software engineer. During earlier software engineering activities, the engineer attempts to build software from an abstract concept to a tangible product. Now comes testing. The engineer creates a series of test cases that are intended to "demolish" the software that has been built. Testing is the one step in the software process that could be viewed (psychologically, at least) as destructive rather than constructive.

(i) Testing Objectives :

In an excellent book on software testing, Glen Myers states a number of rules that can serve well as testing objectives:

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as return discovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error. These objectives imply a dramatic change in viewpoint. They move counter to the commonly held view that a successful test is one in which no errors are found. Our objective is to design tests that systematically uncover different classes of errors, and to do so with a minimum amount of time and effort. If testing is conducted successfully (according to the objectives stated previously), it will uncover errors in the software. As a secondary benefit, testing demonstrates that software functions appear to be working according to specification, that behavioral and performance requirements appear to have been met. In addition, data collected as testing is conducted provide a good indication of software reliability, and some indication of software quality as a whole. But testing cannot show the absence of errors and defects, it can show only that software errors and defects are present. It is important to keep this (rather gloomy) statement in mind as testing is being conducted.

(ii) Testing Principles :

Before applying methods to design effective test cases, a software engineer must understand the basic principles that guide software testing. Davis [DAV95] suggests a set of testing principles that have been adapted for use in this book :

- **All tests should be traceable to customer requirements.** As we have seen, the objective of software testing is to uncover errors. It follows that the most severe defects (from the customer's point of view) are those that cause the program to fail to meet its requirements.
- **Tests should be planned long before testing begins.** Test planning can begin as soon as the requirements model is complete. Detailed definition of test cases can begin as soon as the design

model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.

- **The Pareto principle applies to software testing.** Stated simply, the Pareto principle implies that 80 percent of all errors uncovered during testing will most likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.
- **Testing should begin "in the small" and progress toward testing "in the large".** The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.
- **Exhaustive testing is not possible.** The number of path permutations for even a moderately sized program is exceptionally large. For this season, it is impossible to execute every combination of paths during testing. It is possible, however, to adequately cover program logic and to ensure that all conditions in the component-level design have been exercised.
- **To be most effective, testing should be conducted by an independent third party.** By most effective, we mean testing that has the highest probability of finding errors (the primary objective of testing). For reasons that have been introduced earlier in this unit, the software engineer who created the system is not the best person to conduct all tests for the software.

(iii) Testability :

In ideal circumstances, a software engineer designs a computer program, a system, or a product with "testability" in mind. This enables the individuals charged with testing to design effective test cases more easily. But what is testability ? James Bach describes testability in the following manner. Software testability is simply how easily [a computer program] can be tested. Since testing is so profoundly difficult, it pays to know what can be done to streamline it. Sometimes programmers are willing to do things that will help the testing process and a checklist of possible design points, features, etc., can be useful in negotiating with them. There are certainly metrics that could be used to measure testability in most of its aspects. Sometimes, testability is used to mean how adequately a particular set of tests will cover the product. It's also used by the military to mean how easily a tool can be checked and repaired in the field. Those two meanings are not the same as software testability. The checklist that follows provides a set of characteristics that lead to testable software.

Q. 2. Discuss the CMM 5 Levels for Software Process.

Ans. **The Software Process :**

In recent years, there has been a significant emphasis on "process maturity". The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity. To determine an organization's current state of process maturity, the SEI uses an assessment that results in a five point grading scheme. The grading scheme determines compliance with a *capability maturity model* (CMM) [PAU93] that

defines key activities required at different levels of process maturity. The SEI approach provides a measure of the global effectiveness of a company's software engineering practices, and establishes five process maturity levels that are defined in the following manner :

Level 1 : Initial – The Software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2 : Repeatable – Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3 : Defined – The software process for both management and engineering activities is documented, standardized, and integrated into an organized-wide software process. All projects use a documented and approved version of the organizations process for developing and supporting software. This level includes all characteristic defined for level 2.

Level 4 : Managed – Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5 : Optimizing – Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4. The five levels defined by the SEI were derived as a consequence of evaluating responses to the SEI assessment questionnaire that is based on the CMM. The results of the questionnaire are distilled to a single numerical grade that provides an indication of an organization's process maturity.

The SEI has associated key process areas (KPAs) with each of the maturity levels. The KPAs describe those software engineering functions (e.g., software project planning, requirements management) that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics :

- **Goals –** the overall objectives that the KPA must achieve.
- **Commitments –** requirements (imposed on the organization) that must be met to achieve the goals, or provide proof of intent to comply with the goals.
- **Abilities –** those things must be in place (organizationally and technically) to enable the organization to meet the commitments.
- **Activities –** the specific tasks required to achieve the KPA function.
- **Methods for monitoring implementation –** the manner in which the activities are monitored as they are put into place.
- **Methods for verifying implementation –** the manner in which proper practice for the KPA can be verified.

Q.*3. Discuss the Water Fall Model for Software Development.*

Ans. **The Linear Sequential Model :**

Sometimes called the ***classic life cycle*** or the ***waterfall model***, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding testing, and support. The linear sequential model for software engineering. Modeled after a conventional engineering cycle, the linear sequential model encompasses the following activities :

System / information engineering and modeling – because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when software must interact with other elements such as hardware, people, and databases. System engineering and analysis encompass requirements gathering at the system level, with a small amount of top level design and analysis. Information engineering encompasses requirements gathering at the strategic business level and at the business area level.

Software requirements analysis : The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

Design – Software design is actually a multistep process that focuses on four distinct attributes of a program : data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.

Code generation – The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

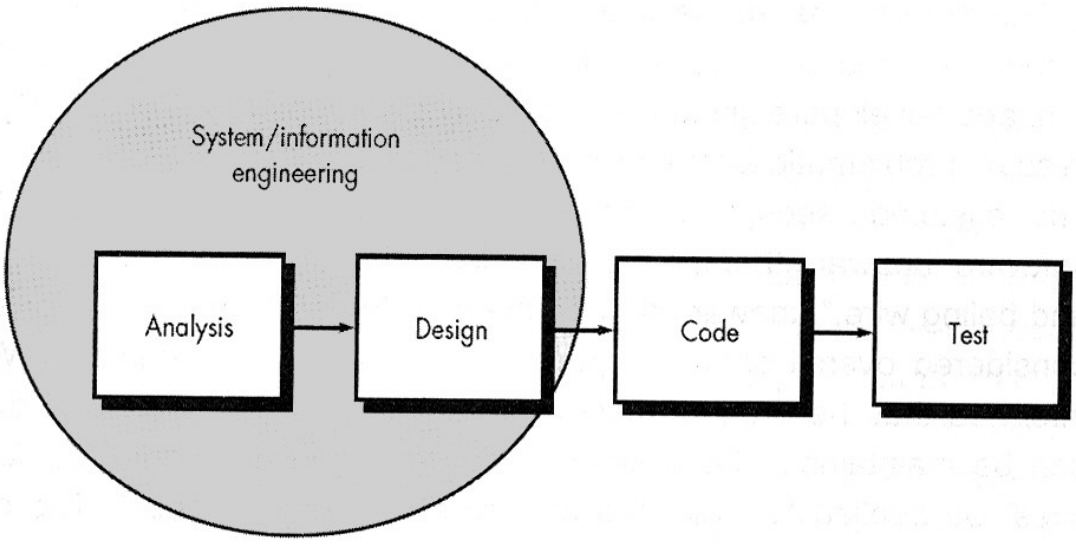
Test – Once the code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with the required results.

Support – Software will undoubtedly undergo change after it is delivered to the customer (a possible exception is embedded software). Change will occur because errors have been encountered, because the software must be adapted to accommodate changes in its external environment (e.g. a change required because of a new operating system or peripheral device), or because the customer requires functional or performance enhancements. Software support / maintenance reapplies each of the preceding phases to an existing program rather than a new one. The linear sequential model is the oldest and the most widely used paradigm for software engineering. However, criticism of the paradigm has caused even active supporters to questions its

efficacy [HAN95]. Among the problems that are sometimes encountered when the linear sequential model is applied are :

1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
2. It is often difficult for the customer to state all requirements explicitly. The linear sequential model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
3. The customer must have patience. A working version of the program(s) will not be available until late in the project time-span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

In an interesting analysis of actual projects Bradac [BRA94], found that the linear nature of the classic life cycle leads to “blocking states” in which some project team members must wait for other members of the team to complete dependent tasks. In fact, the time spent waiting can exceed the time spent on productive work ! The blocking state tends to be more prevalent at the beginning and end of a linear sequential process. Each of these problems is real. However, the classic life cycle paradigm has a definite and important place in software engineering work. It provides a template into which methods for analysis, design, coding, testing, and support can be placed. The classic life cycle remains a widely used procedural model for software engineering. While it does have weaknesses, it is significantly better than a haphazard approach to software development.



Q. 4 . Explain the different types of Software Measurement Techniques.

Ans. **Software Measurement Techniques :**

Measurements in the physical world can be categorized in two ways : direct measures (e.g. the length of a bolt) and indirect measures (e.g. the “quality” of bolts produced, measured by counting rejects). Software metrics can be categorized similarly. Direct measures of the software engineering process include cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size,

and defects reported over some set period of time. Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability, and many other "- abilities".

1. Size Oriented Metrics :

Size-oriented software metrics are derived by normalizing quality and / or productivity measures by considering the size of the software that has been produced. If a software organization maintains simple records, a table of size-oriented measures can be created. The table lists each software development project that has been completed over the past few years and corresponding measures for that project. 12,100 lines of code were developed with 24 person-months of effort at a cost $168,000. It should be noted that the effort and cost recorded in the table represent all software engineering activities (analysis, design, code, and test), not just coding. Further information for project alpha indicates that 365 pages of documentation were developed, 134 errors were recorded before the software was released, and 29 defects were encountered after release to the customer within the first year of operation. Three people worked on the development of software for project alpha.

2. Function Oriented Metrics :

Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures. Function-oriented metrics were first proposed by Albrecht [ALB79], who suggested a measure called the function point. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

3. Extended Function Point Metrics :

The function point measure was originally designed to be applied to business information systems applications. To accommodate these applications, the data dimension (the information domain values discussed previously) was emphasized to the exclusion of the functional and behavioral (control) dimensions. For this reason, the function point measure was inadequate for many engineering and embedded systems (which emphasize function and control). A number of extensions to the basic function point measure have been proposed to remedy this situation.

Q. 5. Explain the COCOMO Model & Software Estimation Technique.

Ans. **Software Estimation Technique :**

Software cost and effort estimation will never be an exact science. Too many variables – human, technical, environmental, political – can affect the ultimate cost of software and effort applied to develop it. However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable riks.

To achieve reliable cost and effort estimates, a number of options arise :

1. Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete !).

2. Base estimates on similar projects that have already been completed.

3. Use relatively simple decomposition techniques to generate project cost and effort estimates.

4. Use one or more empirical models for software cost and effort estimation. Unfortunately, the first option, however attractive, is not practical. Cost estimates must be provided "up front". However, we should recognize that the longer we wait, the more we know, and the more we know, the less likely we are to make serious errors in our estimates.

The second option can work reasonably well, if the current project is quite similar to past efforts and other project influences (e.g. the customer, business conditions, the SEE, deadlines) are equivalent. Unfortunately, past experience has not always been a good indicator of future results.

The COCOMO Model :

In his classic book on "software engineering economics", Barry Boehm [BOE81] introduced a hierarchy of software estimation models bearing the name COCOMO, for Constructive Cost Model. The original COCOMO model became one of the most widely used an discussed software cost estimation models in the industry. It has evolved into a more comprehensive estimation model, called COCOMO II [BOE96, BOE00]. Like its predecessor, COCOMO II is actually a hierarchy of estimation models that address the following areas :

Application composition model : Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.

Early design stage model : Used once requirements have been stabilized and basic software architecture has been established.

Post-architecture-stage model : Used during the construction of the software.

Q. 6 : Write a note on myths of Software.

Ans. Most knowledgeable professionals recognize myths for what they are – misleading attitudes that have caused serious problems for managers and technical people alike. However, old attitudes and habits are difficult to modify, and remnants of software myths are till believed.

Primarily, there are three types of software myths, all the three are stated below :

1. Management Myths – Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).

Myth – We already have a book that's full of standards and procedures for building software; won't that provide my people with everything they need to know ?

Reality – The book of standards may very well exist, but is it used ? Are software practitioners aware of its existence ? Does it reflect modern software engineering practice ? Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality ? In many cases, the answer to all of these questions is "no".

Myth – May people have state-of-the-art software development tools, after all, we buy them the newest computers.

Reality – It takes much more than the latest model mainframe, workstation, or PC to do high-quality software development. Computer-aided software engineering (CASE) tools are more important than hardware for achieving good quality and productivity, yet the majority of software developers still do not use them effectively.

Myth – If we get behind schedule, we can add more programmers and catch up (sometimes called the Mongolian horde concept).

Reality – Software development is not a mechanistic process like manufacturing. In the words of Brooks [BR075] : "adding people to a late software project makes it later". At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well-coordinated manner.

Myth – If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality – If an organization does not understand how to manage and control software projects internally, it will invariable struggle when it outsource software projects.

2. Customer Myths – A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing / sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and ultimately, dissatisfaction with the developer.

Myth – A general statement of objectives is sufficient to begin writing programs – we can fill in the details later.

Reality – A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.

Myth – Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality – It is true that software requirements change, but the impact of change varies with the time at which it is introduced. If serious attention is given to up-front definition, early requests for change can be accommodated easily. The customer can review requirements and recommend modifications with relatively little impact on cost. When changes are requested during software design, the cost impact grows rapidly. Resources have been committed and a design framework has been established. Change can cause upheaval that requires additional resources and major design modification, that is, additional cost. Changes in function, performance, interface, or other characteristics during implementation (code and test) have a severe impact on cost. Change, when requested after software is in production, can be over an order of magnitude more expensive than the same change requested earlier.

3. Practitioner's Myths – Myths that are still believed by software practitioners have been fostered by 50 years of programming culture. During the early days of software, programming was viewed as an art form. Old ways and attitudes die hard.

Myth – Once we write the program and get it to work, our job is done.

Reality – Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done". Industry data ([LIE80], [JON91], [PUT97]) indicates that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth – Until I get the program "running" I have no way of assessing its quality.

Reality – One of the most effective software quality assurance mechanisms can be applied from the inception of a project – the formal technical review. Software reviews are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

Myth – The only deliverable work product for a successful project is the working program.

Reality – A working program is only one part of a software configuration that includes many elements. Documentation provides a foundation for successful engineering and, more importantly, guidance for software support.

Myth – Software engineering will make up creates voluminous and unnecessary documentation and will invariably slow us down.

Reality – Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times. Many software professionals recognize the fallacy of the myths just described. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach. Recognition of software realities is the first step towards formulation of practical solutions for software engineering.

Software Myths :

Myth is defined as "widely held but false notation" by the oxford dictionary, so as in other fields software arena also has some myths to demystify. Pressman insists "Software myths- beliefs about software and the process used to build it- can be traced to earliest days of computing. Myths have a number of attributes that have made them insidious." So software myths prevail but though they do are not clearly visible they have the potential to harm all the parties involved in the software development process mainly the developer team.

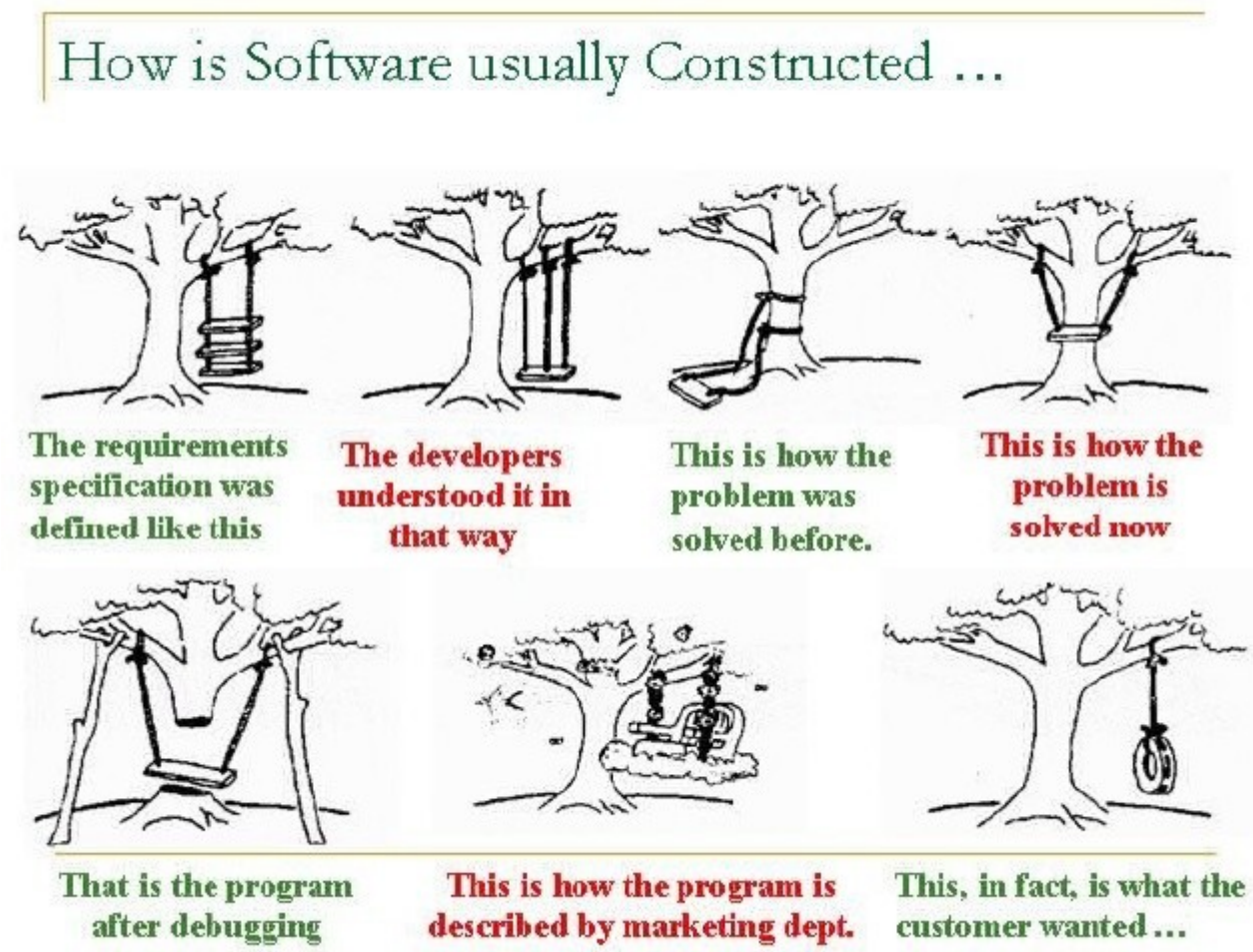
Tom DeMarco expresses "In the absence of meaningful standards, a new industry like software comes to depend instead on folklore." The given statement points out that the software industry caught pace just some decades back so it has not matured to a formidable level and there are no strict standards in

software development. There does not exist one best method of software development that ultimately equates to the ubiquitous software myths.

Primarily, there are three types of software myths, all the three are stated below:

1. Management Myth
2. Customer Myth
3. Practitioner/Developer Myth

Before defining the above three myths one by one lets scrutinize why these myths occur on the first place. The picture below tries to clarify the complexity of the problem of software development requirement analysis mainly between the developer team and the clients.



The above pictures elucidate that the techies understand the problem differently than what it really is and it results to a different solution as the problem itself is misunderstood. So the problem understanding i.e. requirement analysis must be done properly to avoid any problems in later stages as it will have devastating effects.

1. **Management Myths:** Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if those beliefs will lessen the pressure (even temporarily). Some common managerial myths stated by Roger Pressman include:

   I. We have standards and procedures for building software, so developers have everything they need to know.

   II. We have state-of-the-art software development tools; after all, we buy the latest computers.

   III. If we're behind schedule, we can add more programmers to catch up.

   IV. A good manger can manage any project.

   The managers completely ignore that fact that they are working on something intangible but very important to the clients which invites more trouble than solution. So a software project manger must have worked well with the software development process analyzing the minute deals associated with the field learning the nitty-gritty and the tips and trick of the trade. The realities are self understood as it is already stated how complex the software development process is.

2. **Customer Myths:** A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer. Commonly held myths by the clients are:

   I.A general statement of objectives is sufficient to begin writing programs - we can fill in the details later.
   II. Requirement changes are easy to accommodate because software is flexible.
   III. I know what my problem is; therefore I know how to solve it.

   This primarily is seen evidently because the clients do not have a first hand experience in software development and they think that it's an easy process.

3. **Practitioner/ Developer Myths:** Myths that are still believed by software practitioners have been fostered by over 50 years of programming culture. During the early days of software, programming was viewed as an art form. Old ways and attitudes die hard. A malpractice seen is developers are that they think they know everything and neglect the peculiarity of each problem.

I. If I miss something now, I can fix it later.

II. Once the program is written and running, my job is done.

III. Until a program is running, there's no way of assessing its quality.

IV. The only deliverable for a software project is a working program.

Every developer should try to get all requirement is relevant detail to effectively design and code the system.

Some misplaced assumptions that intensify the myths are listed below:

1. All requirements can be pre-specified
2. Users are experts at specification of their needs
3. Users and developers are both good at visualization
4. The project team is capable of unambiguous communication

On the whole, realities are always different from the myths. So the myths must be demystified and work should be based on systematic, scientific and logical bases than the irrational myths. The systemic view must be considered to determine the success of any software project its not only the matter of hard skills but soft skills of the developer team also matter to come up with a efficient system.

Assignment (Set-2)

Subject code: MI0033

Software Engineering

Q 1. Quality and reliability are related concepts but are fundamentally different in a number of ways. Discuss them.

Ans: Software quality is defined as conformance to explicitly stated functional and non-functional requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

This definition emphasizes upon three important points:

• Software requirements are the foundation from which quality is measured. Lack of conformance is lack of quality

• Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.

• A set of implicit requirements often goes unmentioned (ease of use, good maintainability etc.)

DeMarco defines product quality as a function of how much it changes the world for the better.

So, there are many different way to look at the quality.

Quality Assurance

Goal of quality assurance is to provide the management with the necessary data to be informed about product quality. It consists of auditing and reporting functions of management.

Cost of quality

does quality assurance add any value.

If we try to prevent problems, obviously we will have to incur cost. This cost includes:

• Quality planning

• Formal technical reviews

• Test equipment

• Training

The cost of appraisal includes activities to gain insight into the product condition. It compare these numbers to the cost of defect removal once the product has been shipped to the customer. Mostly I think it is profitable

Software Reliability:

“Probability of failure free operation of a computer program in a specified environment for a specified time”. For example, a program X can be estimated to have a reliability of 0.96 over 8 elapsed hours.

Software reliability can be measured, directed, and estimated using historical and development data. The key to this measurement is the meaning of term failure. Failure is defined as non-conformance to software requirements. It can be graded in many different ways as shown below:

• From annoying to catastrophic

• Time to fix from minutes to months

• Ripples from fixing

It is also pertinent to understand the difference between hardware and software reliability. Hardware reliability is predicted on failure due to wear rather than failure due to design. In the case of software, there is no wear and tear. The reliability of software is determined by Mean time between failure (MTBF). MTBF is calculated as:

MTBF = MTTF + MTTR

Where MTTF is the Mean Time to Failure and MTTR is the Mean time required to Repair.

Arguably MTBF is far better than defects/kloc as each error does not have the same failure rate and the user is concerned with failure and not with total error count.

A related issue is the notion of availability. It is defined as the probability that a program is operating according to requirements at a given point in time. It can be calculated as

Availability = (MTTF/MTBF) x 100

and clearly depends upon MTTR.

Q.2. Explain Version Control & Change Control.

Ans. **Version Control:**

Code evolves. As a project moves from first-cut prototype to deliverable, it goes through multiple cycles in which you explore new ground, debug, and then stabilize what you've accomplished. And this evolution doesn't stop when you first deliver for production. Most projects will need to be maintained and enhanced past the 1.0 stage, and will be released multiple times. Tracking all that detail is just the sort of thing computers are good at and humans are not.

Why Version Control?

Code evolution raises several practical problems that can be major sources of friction and drudgery — thus a serious drain on productivity. Every moment spent on these problems is a moment not spent on getting the design and function of your project right.

Perhaps the most important problem is reversion. If you make a change, and discover it's not viable, how can you revert to a code version that is known good? If reversion is difficult or unreliable, it's hard to risk making changes at all (you could trash the whole project, or make many hours of painful work for yourself).

Almost as important is change tracking. You know your code has changed; do you know why? It's easy to forget the reasons for changes and step on them later. If you have collaborators on a project, how do you know what they have changed while you weren't looking, and who was responsible for each change?

Amazingly often, it is useful to ask what you have changed since the last known-good version, even if you have no collaborators. This often uncovers unwanted changes, such as forgotten debugging code. I now do this routinely before checking in a set of changes.

-- Henry Spencer

Another issue is bug tracking. It's quite common to get new bug reports for a particular version after the code has mutated away from it considerably. Sometimes you can recognize immediately that the bug has already been stomped, but often you can't. Suppose it doesn't reproduce under the new version. How do you get back the state of the code for the old version in order to reproduce and understand it?

To address these problems, you need procedures for keeping a history of your project, and annotating it with comments that explain the history. If your project has more than one developer, you also need mechanisms for making sure developers don't overwrite each others' versions.

Version Control by Hand

The most primitive (but still very common) method is all hand-hacking. You snapshot the project periodically by manually copying everything in it to a backup. You include history comments in source files. You make verbal or email arrangements with other developers to keep their hands off certain files while you hack them.

As with most hand-hacking, this method does not scale well. It restricts the granularity of change tracking, and tends to lose metadata details such as the order of changes, who did them, and why. Reverting just a part of a large change can be tedious and time consuming, and often developers are forced to back up farther than they'd like after trying something that doesn't work.

Automated Version Control

To avoid these problems, you can use a version-control system (VCS), a suite of programs that automates away most of the drudgery involved in keeping an annotated history of your project and avoiding modification conflicts.

Most VCSs share the same basic logic. To use one, you start by registering a collection of source files — that is, telling your VCS to start archive files describing their change histories. Thereafter, when you want to edit one of these files, you have to check out the file — assert an exclusive lock on it. When you're done, you check in the file, adding your changes to the archive, releasing the lock, and entering a change comment explaining what you did.

Most of the rest of what a VCS does is convenience: labeling, and reporting features surrounding these basic operations, and tools which allow you to view differences between versions, or to group a given set of versions of files as a named release that can be examined or reverted to at any time without losing later changes.

Another problem is that some kinds of natural operations tend to confuse VCSs. Renaming files is a notorious trouble spot; it's not easy to automatically ensure that a file's version history will be carried along with it when it is renamed. Renaming problems are particularly difficult to resolve when the VCS supports branching.

Change Control:

Change control within Quality management systems (QMS) and Information Technology (IT) systems is a formal process used to ensure that changes to a product or system are introduced in a controlled and coordinated manner. It reduces the possibility that unnecessary changes will be introduced to a system without forethought, introducing faults into the system or undoing changes made by other users of software. The goals of a change control procedure usually include minimal disruption to services, reduction in back-out activities, and cost-effective utilization of resources involved in implementing change.

Change control is currently used in a wide variety of products and systems. For Information Technology (IT) systems it is a major aspect of the broader discipline of change management. Typical examples from the computer and network environments are patches to software products, installation of new operating systems, upgrades to network routing tables, or changes to the electrical power systems supporting such infrastructure.

Certain experts describe change control as a set of six steps[who?]:

Record / Classify

Assess

Plan

Build / Test

Implement

Close / Gain Acceptance

Q. 3. Discuss the SCM Process.

Ans: In software engineering, software configuration management (SCM) is the task of tracking and controlling changes in the software. Configuration management practices include revision control and the establishment of baselines.

SCM concerns itself with answering the question "Somebody did something, how can one reproduce it?" Often the problem involves not reproducing "it" identically, but with controlled, incremental changes. Answering the question thus becomes a matter of comparing different results and of analysing their differences. Traditional configuration management typically focused on controlled creation of relatively simple products. Now, implementers of SCM face the challenge of dealing with relatively minor increments under their own control, in the context of the complex system being developed.
The goals of SCM are generally:[citation needed]

Configuration identification - Identifying configurations, configuration items and baselines.

Configuration control - Implementing a controlled change process. This is usually achieved by setting up a change control board whose primary function is to approve or reject all change requests that are sent against any baseline.

Configuration status accounting - Recording and reporting all the necessary information on the status of the development process.

Configuration auditing - Ensuring that configurations contain all their intended parts and are sound with respect to their specifying documents, including requirements, architectural specifications and user manuals.

Build management - Managing the process and tools used for builds.

Process management - Ensuring adherence to the organization's development process.

Environment management - Managing the software and hardware that host the system.

Teamwork - Facilitate team interactions related to the process.

Defect tracking - Making sure every defect has traceability back to the source.

Effective Configuration Management can be defined as stabilising the evolution of software products and process at key points in the life cycle. The focus of CM includes:

Identification of Artefacts

Early identification and change control of artefacts and work products is integral to the project. The configuration manager needs to fully identify and control changes to all the elements that are required to recreate and maintain the software product.

Version Control

The primary goal of version control is to identify and manage project elements as they change over time. The Configuration Manager should establish a version control library to maintain all lifecycle entities. This library will ensure that changes (deltas) are controlled at their lowest atomic level eg documents, source files, scripts and kits etc.

Development Streaming (Branching)

To provide some level of stability and allow fluidity of parallel development (streaming) it is quite normal for project development to be split into branches (development groups).

The CM manager has to identify what branches will be required and ensure they are appropriately set up (eg security etc).

Baselining

Baselining provides the division with a concise picture of the project artifacts and relationships at a particular instance in time. It provides an official foundation on which subsequent work can be based, and to which only authorized changes can be made.

Through baselining (i.e. labelling, tagging) all constituent project components are aligned, uniquely identifiable and reproducible at both the atomic level (eg file) and at the higher kit levels.

Reasons for baselining include:

A baseline supports ease of roll back

A baseline improves CM managers ability to create change reports etc

A baseline supports creation of new parallel branches (e.g. dev branches)

A baseline supports troubleshooting and element comparison

A baseline provides a stable bill of material for the build system

Build Management

The fundamental objective of the build management process is to deliver a disciplined and automated build process.

Activities to consider:

Create automated build scripts (i.e. fetching from repository)

Enforce baselining before all formal builds (support bill of materials/traceability)

Set up stable build machines

Packaging

Typically the packaging process (see next section) will by synonymous [or tightly coupled] with the build process i.e. the build process will do packaging automatically after the build is complete.

Primary objectives of packaging are:

Manageable (i.e. often a single zipped up file or exe)

Reusable (i.e. Try to avoid need for rebuild)

Secure (i.e. Packages should be free from malicious or accidental modification)

Deployment

The configuration manager will typically be involved in the deployment process. Primary considerations include:

Ensuring deployment automated (reducing possibility for manual errors).

Promoting best practice concepts concept like promotion based releases (opposed to environmental rebuilds).

Ensuring releases are authorised and appropriate windows selected for deployment.

Providing streamlined rollback mechanism in case of problem.

Change Request Management

Change Request management can be described as management of change/enhancement requests.

Typically the Configuration Manger should set up a repository to manage these requests and support activities like status tracking, assignment etc.

Issue Tracking

Issue tracking is the formal tacking of problems/defects on your systems or environments.

Typically the Configuration Manger should set up a repository to track these problems as they occur, and track their status to eventual closure.

Q 4. Explain

i. Software doesn't Wear Out.

ii. Software is engineered & not manufactured.

Ans: **Software:**

It is a document that describe the operation and use of the program. Data structure that enables the program to manipulate the information.Instructions that when executed provides the desired features or function.

Software can be categorized in two types

Generic software:

Generic software is those which are developed for a broad category of customers. These are the users whose environment is well understood and common for all. This type of software sold in the open market where they face several competitors.

Customized software:

This type of software is meant or developed keeping in mind the needs of a particular customer e.g. hospital management system. These types of users have their own unique domain, environment and requirements.

Following are the characteristics of software engineering, we may say that:-

(I)**Software doesn't wear out:**

The hardware can wear out whereas software can't. In case of hardware we have a "bathtub" like curve, which is a curve that lies in between failure-rate and time. In this curve, in the starting time there is relatively high failure rate. But, after some period of time, defects get corrected and failure-rate drops to a steady-state for some time period. But, the failure-rate again rises due to the effects of rain, dust, temperature extreme and many other environment effects. The hardware begins to wear out.

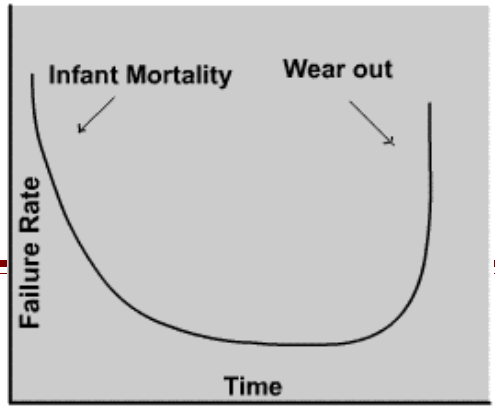


Figure above depicts failure rate as a function of time for hardware. The relationship often called the "bath tub curve"

indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and any other environmental maladies. Stated simply, the hardware begins to wear out.

Bath tub curve

Software is not suspected able to the environmental maladies that cause hardware to wear out. In, theory, therefore, the failure rate curve for the software should take the form of the "idealized curve". Undiscovered defects will cause high failure early in the life of a program. However these are corrected (ideally, without introducing other errors) and the curve flattens. However, the implication is clear--software doesn't wear out. But it does deteriorate!

(II) **Software is not manufactured in the classical sense, but it is developed or engineered:**

Software or hardware both get manufactured in the same manner and both of them uses the design model to implement the product. The only difference is in their implementation part. They both differ in their coding part. So, it is said that software is not manufactured but it is developed or engineered. The only difference lies in the cost of both the hardware and software.

Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software. Both activities are dependent on the people, but the relationship between people applied and work accomplished is entirely different. Both activities require the construction of a "product" but the approaches are different. Software costs are concentrated in engineering. This means that software projects can not be managed as if they were manufacturing projects.

Q. 5. Explain the Advantages of Prototype Model, & Spiral Model in Contrast to Water Fall model.

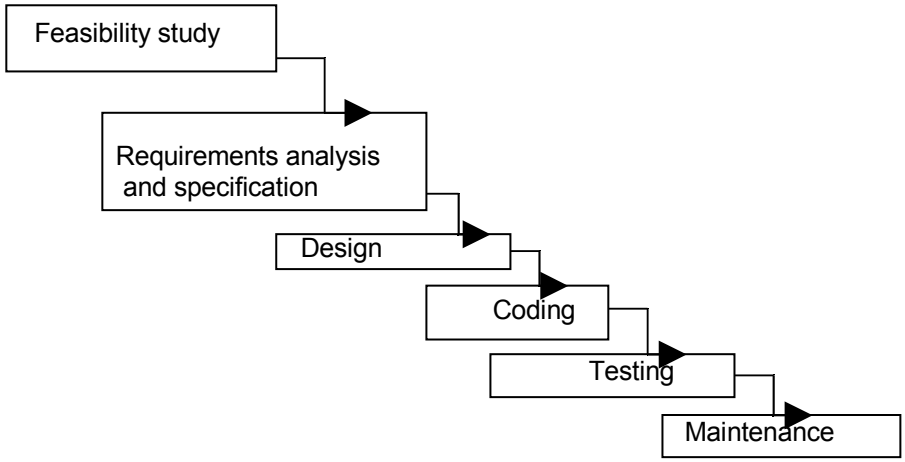
Ans: Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

- **Classical Waterfall Model**
- **Iterative Waterfall Model**
- **Prototyping Model**
- **Evolutionary Model**
- **Spiral Model**

Classical Waterfall Model

The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, we will see that it is not a practical model in the sense that it can not be used in actual software development projects. Thus, we can consider this model to be a *theoretical way of developing software*. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models, we must first learn the classical waterfall model.

Classical waterfall model divides the life cycle into the following phases as shown below:



Feasibility Study

The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product

- At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behaviour of the system.
- After they have an overall understanding of the problem, they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kinds of resources are required, what would be the cost of development and what would be the development time for each solution.
- Based on this analysis, they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

The following is an example of a feasibility study undertaken by an organization. It is intended to give one a feel of the activities and issues involved in the feasibility study phase of a typical software project.

Requirements Analysis and Specification

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

- Requirements gathering and analysis, and
- Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed with a view to clearly understand the customer requirements and weed out the incompleteness and inconsistencies in these requirements.

The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting software required by an organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user

typically has only a partial and incomplete view of the system. Therefore it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.

The customer requirements identified during the requirements gathering and analysis activity are organized into an SRS document. The important components of this document are functional requirements, the non-functional requirements, and the goals of implementation.

Design

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

Traditional design approach: Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the software design.

Object-oriented design approach: In this technique, various objects that occur in the problem domain and the solution domain are first identified, and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design.

Coding and Unit Testing

The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

Integration and System Testing

Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner.

The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to the requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- α – testing: It is the system testing performed by the development team.
- β – testing: It is the system testing performed by a friendly set of customers.
- Acceptance testing: It is the system testing performed by the customer himself after product delivery to determine whether to accept or reject the delivered product.

System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing-related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

Maintenance

Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratio. Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

Shortcomings of the Classical Waterfall Model

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

Prototyping Model

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

The Need for a Prototype

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs.

- how screens might look like
- how the user interface would behave
- how the system would produce outputs, etc.

This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model.

Spiral Model

The Spiral model of software development is shown in fig. 33.8. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study; the next loop with requirements specification; the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants)..

First quadrant (Objective Setting):

- During the first quadrant, we need to identify the objectives of the phase.
- Examine the risks associated with these objectives

Second quadrant (Risk Assessment and Reduction):

- A detailed analysis is carried out for each identified project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed

Third quadrant (Objective Setting):

- Develop and validate the next level of the product after resolving the identified risks.

Fourth quadrant (Objective Setting):

- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- With each iteration around the spiral, progressively a more complete version of the software gets built.

technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models. This is probably a factor deterring its use in ordinary projects.

Comparison of Different Life Cycle Models

The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model can not be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases.

This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However, this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks.

The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.

The evolutionary approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the system is acceptable to the customer.

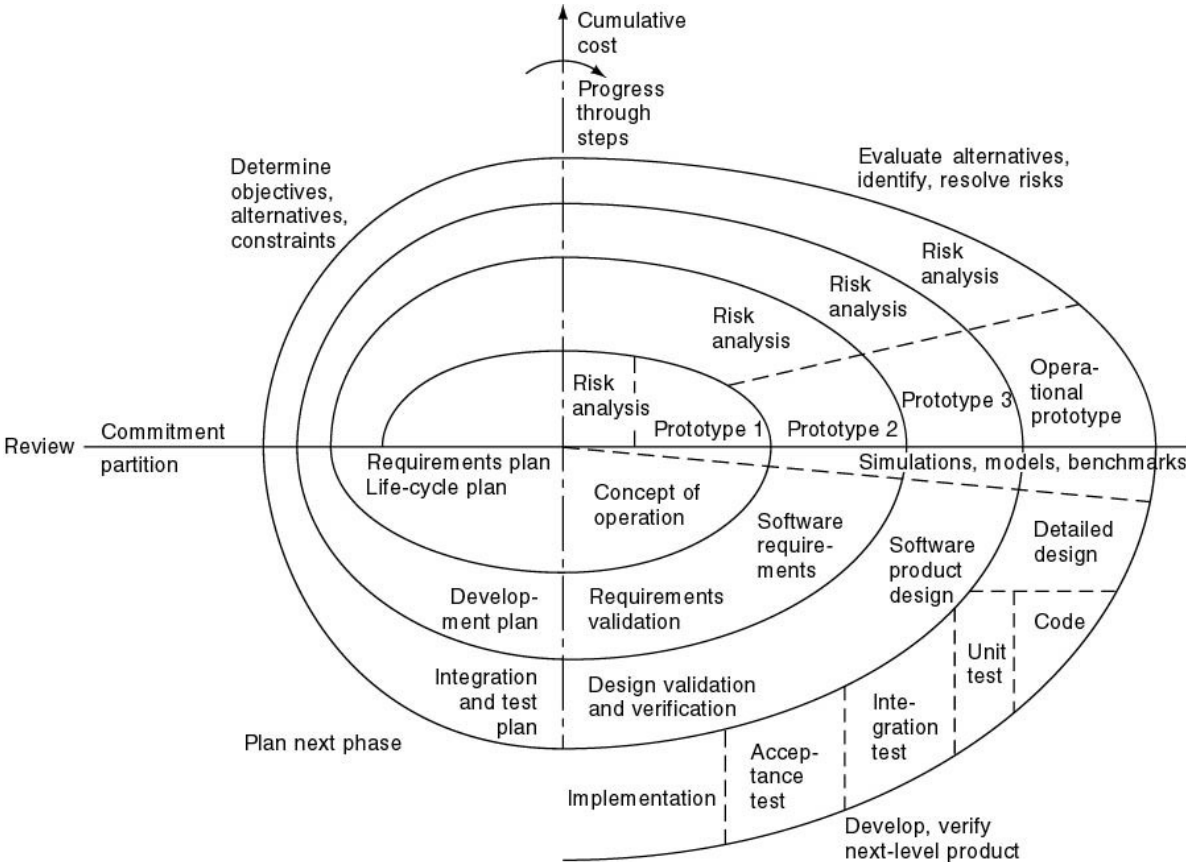
The spiral model is called a meta-model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models. This is probably a factor deterring its use in ordinary projects.

The different software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the long development process, customer confidence normally drops, as no working product is immediately visible. Developers answer customer queries using technical slang, and delays are announced. This gives rise to customer resentment. On the other hand, an evolutionary approach lets the customer experiment with a working product much earlier than the monolithic approaches. Another important advantage of the incremental model is that it reduces the customer's trauma of getting used to an entirely new system. The gradual introduction of the product via incremental phases provides time to the customer to adjust to the new product. Also, from the customer's financial viewpoint, incremental development does not require a large upfront capital outlay. The customer can order the incremental versions as and when he can afford them.

Q. 6. Write a Note on Spiral Model.

Ans: **SPIRAL Model:**

While the waterfall methodology offers an orderly structure for software development,demands for reduced time-to-market make its series steps inappropriate. The next evolutionary step from the waterfall is where the various steps are staged for multiple deliveries or handoffs. The ultimate evolution from the water fall is the spiral, taking advantage of the fact that development projects work best when they are both incremental and iterative, where the team is able to start small and benefit from enlightened trial and error along the way. The spiral methodology reflects the relationship of tasks with rapid prototyping, increased parallelism, and concurrency in design and build activities. The spiral method should still be planned methodically, with tasks and deliverables identified for each step in the spiral.



The Spiral Model is the neo approach in IT project system development and was originally devised by **Barry W. Boehm** through his article published in 1985 *"A Spiral Model of Software Development and Enhancement"*.

This model of development unites the features of the prototyping model with an iterative approach of system development; combining elements of design and prototyping-in-stages. This model is an effort to combine the advantages of top-down and bottom-up concepts highly preferential for large, exclusive, volatile, and complex projects.

The term "spiral" is used to describe the process that is followed in this model, as the development of the system takes place, the mechanisms go back several times over to earlier sequences, over and over again, circulating like a spiral.

The spiral model represents the evolutionary approach of IT project system development and carries the same activities over a number of cycles in order to elucidate system requirements and its solutions.

Similar to the waterfall model, the spiral model has sequential cycles/stages, with each stage having to be completed before moving on to next.

The prime difference between the waterfall model and the spiral model is that the project system development cycle moves towards eventual completion in both the models but in the spiral model the cycles go back several times over to earlier stages in a repetitive sequence.

Progress Cycles, IT Project Management Solutions

The progress cycle of this model is divided into four quadrants, and each quadrant with a different purpose;

Determining Objectives (I)----------Evaluating Alternatives (II)

****

Planning Next Phase (III)----------Planning Next Phase (IV)

First Quadrant: the top left quadrant determines and identifies the project objectives, alternatives, and constrains of the project. Similar to the system conception stage in the Waterfall Model, here objectives are determined with identifying possible obstacles and weighting alternative approaches.

Second Quadrant: the top right quadrant determines the different alternatives of the project risk analysis, and evaluates their task with each alternative eventually resolving them. Probable alternatives are inspected and associated risks are recognized. Resolutions of the project risks are evaluated, and prototyping is used wherever necessary.

Third Quadrant: the bottom right quadrant develops the system and this quadrant corresponds to the waterfall model with detailed requirements determined for the project.

Fourth Quadrant: the bottom left quadrant plans the next phase development process, providing opportunity to analyze the results and feedback.

In each phase, it begins with a system design and terminates with the client reviewing the progress through prototyping.

The major advantage of the spiral model over the waterfall model is the advance approach on setting project objectives, project risk management and project planning into the overall development cycle. Additionally, another significant advantage is, the user can be given some of the functionality before the entire system is completed.

The spiral model addresses complexity of predetermined system performance by providing an iterative approach to system development, repeating the same activities in order to clarify the problem and provide an accurate classification of the requirement within the bounds of multiple constraints.

Assignment (Set-1)

Subject code: MI0034

Database Management Systems

Q.1 : Differentiate between Traditional File System & Modern Database System ? Describe the properties of Database & the advantage of Database ?

Ans. ***Differentiate between Traditional File System & Modern Database System ?***

Traditional File System	Modern Database Management System
Traditional File System is the system that was followed before the advent of DBMS i.e. it is the older way.	This is the Modern way which has replaced the older concept of File System.
In Traditional file processing, data definition is part of the application program and works with only specific application.	• Data definition is part of the DBMS. • Application is independent and can be used with any application.
File systems are Design Driven; they require design/coding change when new kind of data occurs. **E.g. :** In a traditional employee the master file has Emp_name, Emp_id, Emp_addr, Emp_design, Emp_dept, Emp_sal, if we want to insert one more column 'Emp_Mob number' then it requires a complete restructuring of the file or redesign of the application code, even though basically all the data except that in one column is the same.	• One extra column (Attribute) can be added without any difficulty. • Minor coding changes in the Application program may be required.
Traditional File system keeps redundant (duplicate) information in many locations. This might result in the loss of Data Consistency. **For e.g. :** Employee names might exist in separate files like Payroll Master File and also in Employee Benefit Master File etc. Now if an employee changes his or her last name, the name might be changed in they pay roll master file but not be changed in Employee Benefit Master File etc. This might result in the loss of Data Consistency.	Redundancy is eliminated to the maximum extent in DBMS if properly defined.
In a File system data is scattered in various files, and each of these files may be in different formats, making it difficult to write new application programs to retrieve	This problem is completely solved here.

the appropriate data.	
Security features are to be coded in the Application Program itself.	Coding for security requirements is not required as most of them have been taken care by the DBMS.

- Hence, a data base management system is the software that manages a database, and is responsible for its storage, security, integrity, concurrency, recovery and access.
- The DBMS has a data dictionary, referred to as system catalog, which stores data about everything it holds, such as names, structure, locations and types. This data is also referred to as Meta data.

Describe the properties of Database & the advantage of Database ?

Properties of Database :

The following are the important properties of Database :

1. A database is a logical collection of data having some implicit meaning. If the data are not related then it is not called as proper database. E.g. Student studying in class II got 5th rank.

Stud_name	Class	Rank obtained
Vijetha	Class II	5th

2. A database consists of both data as well as the description of the database structure and constraints.

   E.g.

Field Name	Type	Description
Stud_name	Character	It is the student's name
Class	Alpha numeric	It is the class of the student

3. A database can have any size and of various complexity. If we consider the above example of employee database the name and address of the employee may consists of very few records each with simple structure.

   E.g.

Emp_name	Emp_id	Emp_addr	Emp_desig	Emp_Sal
Prasad	100	"Shubhodaya", Near Katariguppe Big Bazaar, BSK II stage, Bangalore	Project Leader	40000
Usha	101	#165, 4th main Chamrajpet,	Software	10000

		Bangalore	engineer	
Nupur	102	#12, Manipal Towers, Bangalore	Lecturer	30000
Peter	103	Syndicate house, Manipal	IT executive	15000

Like this there may be 'n' number of records.

4. The DBMS is considered as general-purpose software system that facilitates the process of defining, constructing and manipulating database for various applications.
5. A database provides insulation between programs, data and data abstraction. Data abstraction is a feature that provides the integration of the data source of interest and helps to leverage the physical data however the structure is.
6. The data in the database is used by variety of users for variety of purposes. For E.g. when you consider a hospital database management system the view of usage of patient database is different from the same used by the doctor. In this case the data are stored separately for the different users. In fact it is stored in a single database. This property is nothing but multiple views of the database.
7. Multiple user DBMS must allow the data to be shared by multiple users simultaneously. For this purpose the DBMS includes concurrency control software to ensure that the updation done to the database by variety of users at single time must get updated correctly. This properly explains the multiuser transaction processing.

Advantages of Database (DBMS) :

1. Redundancy is reduced.
2. Data located on a server can be shared by clients.
3. Integrity (accuracy) can be maintained.
4. Security features protect the Data from unauthorized access.
5. Modern DBMS support internet based application.
6. In DBMS the application program and structure of data are independent.
7. Consistency of Data is maintained.
8. DBMS supports multiple views. As DBMS has many users, and each one of them might use it for different purposes, and may require to view and manipulate only on a portion of the database, depending on requirement.

Q. 2 : What is the disadvantages of sequential file organization ? How do you overcome it? What are the advantages & disadvantages of Dynamic Hashing?

Ans. One disadvantage of sequential file organization is that we must use linear search or binary search to locate the desired record and that results in more i/o operations. In this there are a number of unnecessary comparisons. In hashing technique or direct file organization, the key value is converted into an address by performing some arithmetic manipulation on the key value, which provides very fast access to records.

Key Value ⟶ Hash function ⟶ Address

Let us consider a hash function h that maps the key value k to the value h(k). The VALUE h(k) is used as an address.

The basic terms associated with the hashing techniques are :

1) Hash table : It is simply an array that is having address of records.
2) Hash function : It is the transformation of a key into the corresponding location or address in the hash table (it can be defined as a function that takes key as input and transforms it into a hash table index).
3) Hash key : let 'R' be a record and it key hashes into a key value called hash key.

Internal Hashing :

For internal files, hash table is an array of records, having array in the range from 0 to M-1. Let as consider a hash function H(K) such that H(K)=key mod M which produces a remainder between 0 and M-1 depending on the value of key. This value is then used for the record address. The problem with most hashing function is that they do not guarantee that distinct value will hash to distinct address, a situation that occurs when two non-identical keys are hashed into the same location.

For example : let us assume that there are two non-identical keys k1=342 and k2=352 and we have some mechanism to covert key values to address. Then the simple hashing function is :

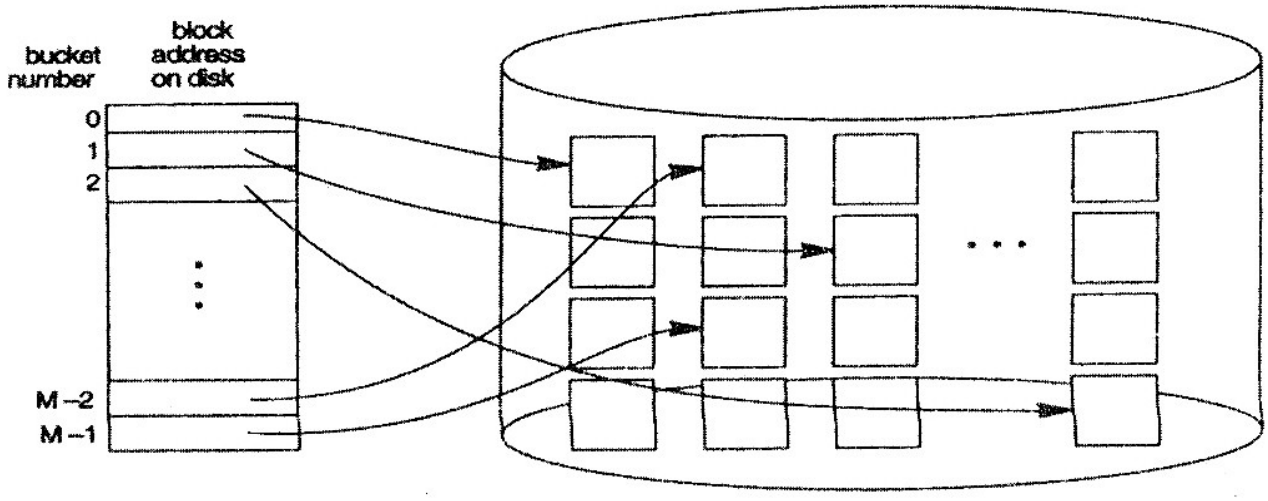
$$h(k) = k \bmod 10$$

Here h(k) produces a bucket address.

To insert a record with key value k, we must have its key first. E.g. : Consider h (K-1)=K1% 10 will get 2 as the hash value. The record with key value 342 is placed at the location 2, another record with 352 as its key value produces the same has address i.e. h(k1) = h(k2). When we try to place the record at the location where the record with key K1 is already stored, there occurs a collision. The process of finding another position is called collision resolution. There are numerous methods for collision resolution.

1) **Open addressing :** With open addressing we resolve the hash clash by inserting the record in the next available free or empty location in the table.
2) **Chaining :** Various overflow locations are kept, a pointer field is added to each record and the pointer is set to address of that overflow location.

External Hashing for Disk Files :

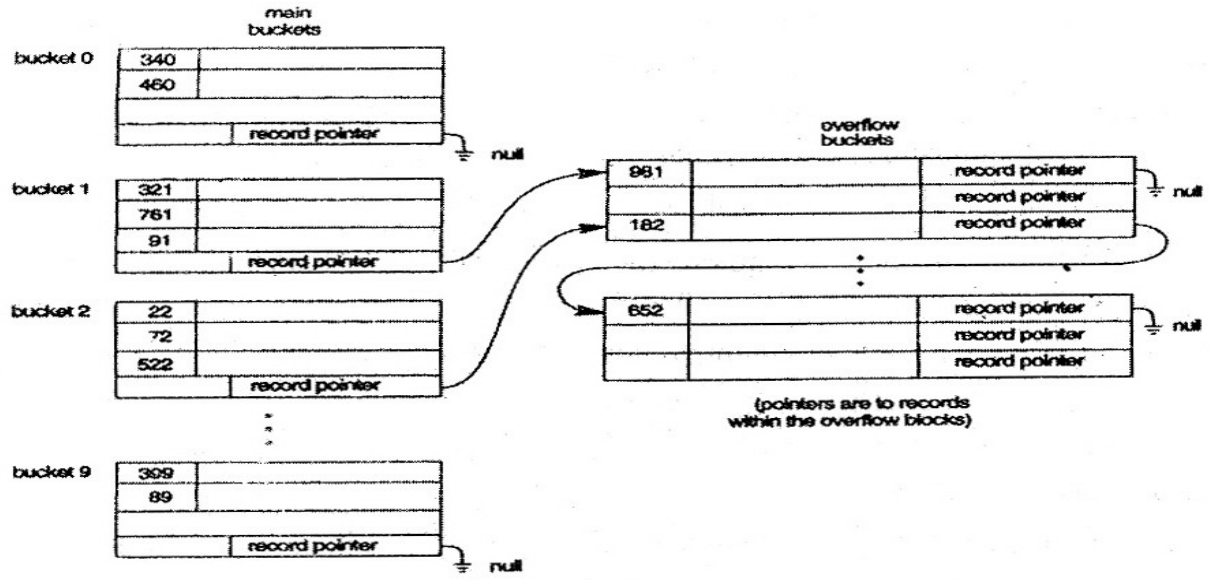


Matching bucket numbers to disk block addresses

Handling Overflow for Buckets by Chaining :

Hashing for disk files is called external hashing. Disk storage is divided into buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of continuous blocks.

The hashing function maps a key into a relative bucket number. A table maintained in the file header converts the bucket number into the corresponding disk block address.



Handling overflow for buckets by chaining

The collision problem is less severe with buckets, because many records will fit in a same bucket. When a bucket is filled to capacity and we try to insert a new record into the same bucket, a collision is caused. However, we can maintain a pointer in each bucket to address overflow records.

The hashing scheme described is called static hashing, because a fixed number of buckets 'M' is allocated. This can be serious drawback for dynamic files. Suppose M be a number of buckets, m be the maximum

number of records that can fit in one bucket, then at most m*M records will fit in the allocated space. If the records are fewer than m*M numbers, collisions will occur and retrieval will be slowed down.

What are the advantages & disadvantages of Dynamic Hashing ?

Advantages of Dynamic Hashing :

1. The main advantage is that splitting causes minor reorganization, since only the records in one bucket are redistributed to the two new buckets.
2. The space overhead of the directory table is negligible.
3. The main advantage of extendable hashing is that performance does not degrade as the file grows. The main space saving of hashing is that no buckets need to be reserved for future growth; rather buckets can be allocated dynamically.

Disadvantages of Dynamic Hashing :

1. The index tables grow rapidly and too large to fit in main memory. When part of the index table is stored on secondary storage, it requires extra access.
2. The directory must be searched before accessing the bucket, resulting in two-block access instead of one in static hashing.
3. A disadvantages of extendable hashing is that it involves an additional level of indirection.

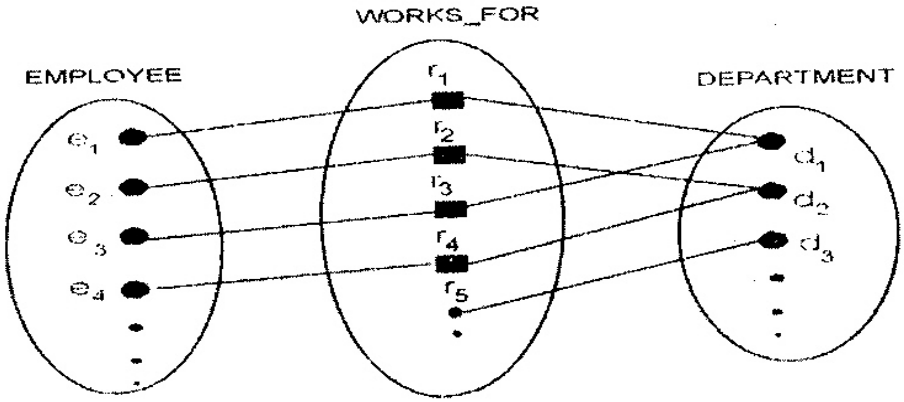
Q.3. What is relationship type ? Explain the difference among a relationship instance, relationship type & a relation set ?

Ans. In the real world, items have relationships to one another. E.g. : A book is published by a particular publisher. The association or relationship that exists between the entities relates data items to each other in a meaningful way. A relationship is an association between entities. A collection of relationships of the same type is called a relationship set.

A relationship type R is a set of associations between E, E2…….En entity types mathematically, R is a set of relationship instances ri.

E.g. : Consider a relationship type WORKS_FOR between two entity types – employee and department, which associates each employee with the department the employee works for. Each relationship instance in WORKS_FOR associates one employee entity and one department entity, where each relationship instance is ri which connects employee and department entities that participate in ri.

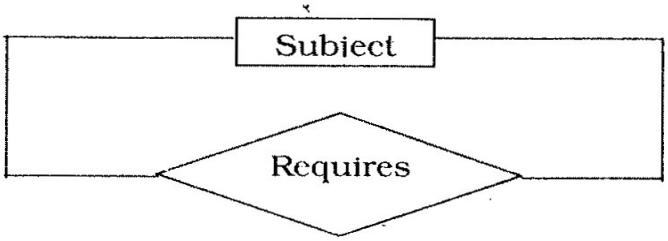
Employee el, e3 and e6 work for department d1, e2 and e4 work for d2 and e5 and e7 work for d3. Relationship type R is a set of all relationship instances.



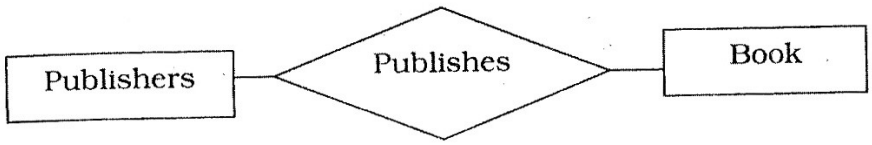
Some instances of the WORKS_FOR relationship

Degree of relationship type :

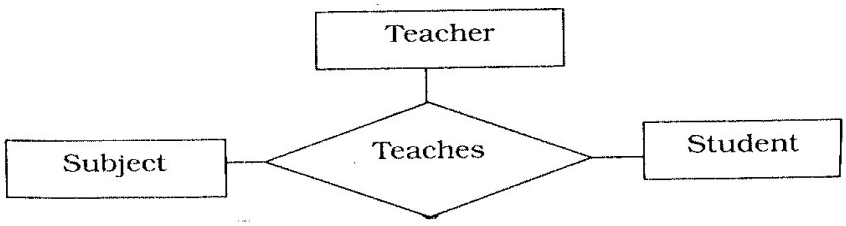
The number of entity sets that participate in a relationship set. A unary relationship exists when an association is maintained with a single entity.



A binary relationship exists when two entities are associated.



A tertiary relationship exists when there are three entities associated.



Degree of relationship type

Role Name and Recursive Relationship :

Each entry type to participate in a relationship type plays a particular role in the relationship. The role name signifies the role that a participating entity from the entity type plays in each relationship instance, e.g. : In the WORKS FOR relationship type, the employee plays the role of employee or worker and the department plays the role of department or employer. However in some cases the same entity type participates more than once in a relationship type in different roles. Such relationship types are called recursive.

E.g. : employee entity type participates twice in SUPERVISION once in the role of supervisor and once in the role of supervisee.

Q. 4 : What is SQL ? Discuss.

Ans. Structured Query Language (SQL) is a specialized language for updating, deleting, and requesting information from databases. SQL is an ANSI and ISO standard, and is the de facto standard database query language. A variety of established database products support SQL, including products from Oracle and Microsoft SQL Server. It is widely used in both industry and academia, often for enormous, complex databases.

In a distributed database system, a program often referred to as the database's "back end" runs constantly on a server, interpreting data files on the server as a standard relational database. Programs on client computers allow users to manipulate that data, using tables, columns, rows, and fields. To do this, client programs send SQL statements to the server. The server then processes these statements and returns replies to the client program.

Examples

To illustrate, consider a simple SQL command, SELECT. SELECT retrieves a set of data from the database according to some criteria, using the syntax :

SELECT list_of_column_names from list_of_relation_names where conditional_expression_that_identifies_specific_rows

- The list_of_relation_names may be one or more comma-separated table names or an expression operating on whole tables.
- The conditional_expression will contain assertions about the values of individual columns within individual rows within a table, and only those rows meeting the assertions will be selected. Conditional expressions within SQL are very similar to conditional expressions found in most programming languages.

For example, to retrieve from a table called Customers all columns (designated by the asterisk) with a value of Smith for the column Last_Name, a client program would prepare and send this SQL statement to the server back end :

SELECT * FROM Customers WHERE Last_Name='Smith';

The server back end may then reply with data such as this :

```
+-------------+----------------+--------------------+
| Cust_No  | Last_Name  | First_Name       |
+-------------+----------------+--------------------+
```

```
| 1001      | Smith          | John                 |
| 2039      | Smith          | David                |
| 2098      | Smith          | Matthew              |
+-----------+----------------+----------------------+
```

3 rows in set (0.05 sec)

Following is an SQL command that displays only two columns, column_name_1 and column_name_3, from the table myTable :

SELECT column_name_1, column_name_3 from myTable

Below is a SELECT statement displaying all the columns of the table myTable2 for each row whose column_name_3 value includes the string "brain" :

SELECT * from column_name_3 where column_name_3 like '%brain%'

Q. 5 : What is Normalization ? Discuss various types of Normal Forms ?

Ans. Normalization is the process of building database structures to store data, because any application ultimately depends on its data structures. If the data structures are poorly designed, the application will start from a poor foundation. This will require a lot more work to create a useful and efficient application. Normalization is the formal process for deciding which attributes should be grouped together in a relation. Normalization serves as a tool for validating and improving the logical design, so that the logical design avoids unnecessary duplication of data, i.e. it eliminates redundancy and promotes integrity. In the normalization process we analyze and decompose the complex relations into smaller, simpler and well-structured relations.

Discuss various types of Normal Forms ?

1. Normal forms based on Primary Keys :

A relation schema R is in first normal form if every attribute of R takes only single atomic values. We can also define it as intersection of each row and column containing one and only one value. To transform the un-normalized table (a table that contains one or more repeating groups) to first normal form, we identify and remove the repeating groups within the table.

E.g.

Dept.

D.Name	D.No.	D.location
R&D	5	(England, London, Delhi)
HRD	4	Bangalore

Figure A

Consider the figure that each dept can have number of locations. This is not in first normal form because D.location is not an atomic attribute. The dormain of D location contains multi-values.

There is a technique to achieve the first normal form. Remove the attribute D.location that violates the first normal form and place into separate relation Dept_location

Ex. : Dept

Dept.no.	D.Name
5	R&D
6	HRD

Dept_location

Dept_location	Dept_No

Functional dependency : The concept of functional dependency was introduced by Prof. Codd in 1970 during the emergence of definitions for the three normal forms. A functional dependency is the constraint between the two sets of attributes in a relation from a database.

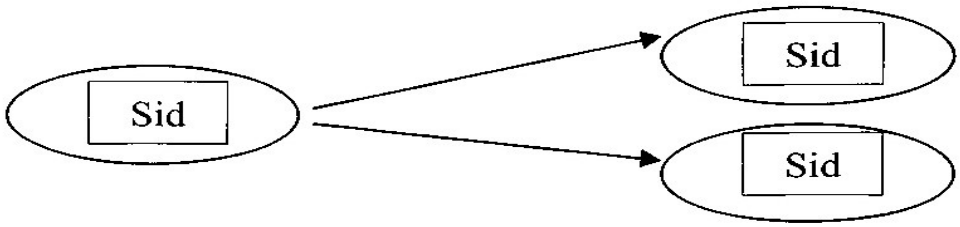
Given a relation R, a set of attributes X in R is said to functionally determine another attribute Y, in R, (X->Y) if and only if each value of X is associated with one value of Y. X is called the determinant set and Y is the dependant attribute.

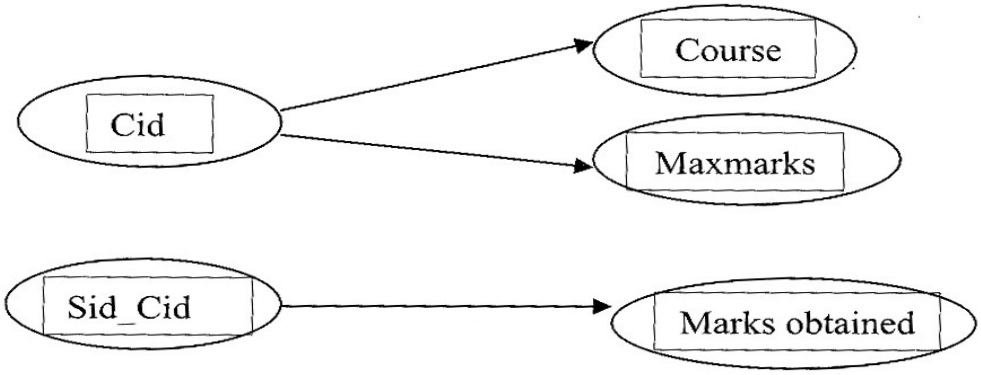
For e.g. : Consider the example of STUDENT_COURSE database.

STUDENT_COURSE

Sid	Sname	Address	Cid	Course	Max marks	Marks Obtained (%)
001	Nupur	Lucknow	MB010	Database Concepts	100	83
001	Nupur	Lucknow	MB011	C++	100	90
002	Priya	Chennai	MB010	Database Concepts	100	85
002	Priya	Chennai	MB011	C++	100	75
002	Priya	Chennai	MQ040	Computer Networks	75	65
003	Pal	Bengal	MB009	Unix	100	70
004	Prasad	Bangalore	MC011	System Software	100	85

In the STUDENT_COURSE database (Sid) student id does not uniquely identifies a tuple and therefore it cannot be a primary key. Similarly (Cid) course id cannot be primary key. But the combination of (Sid, Cid) uniquely identifies a row in STUDENT_COURSE. Therefore (Sid, Cid) is the primary key which uniquely retrieves Sname, address, course, marks, which are dependent on the primary key.

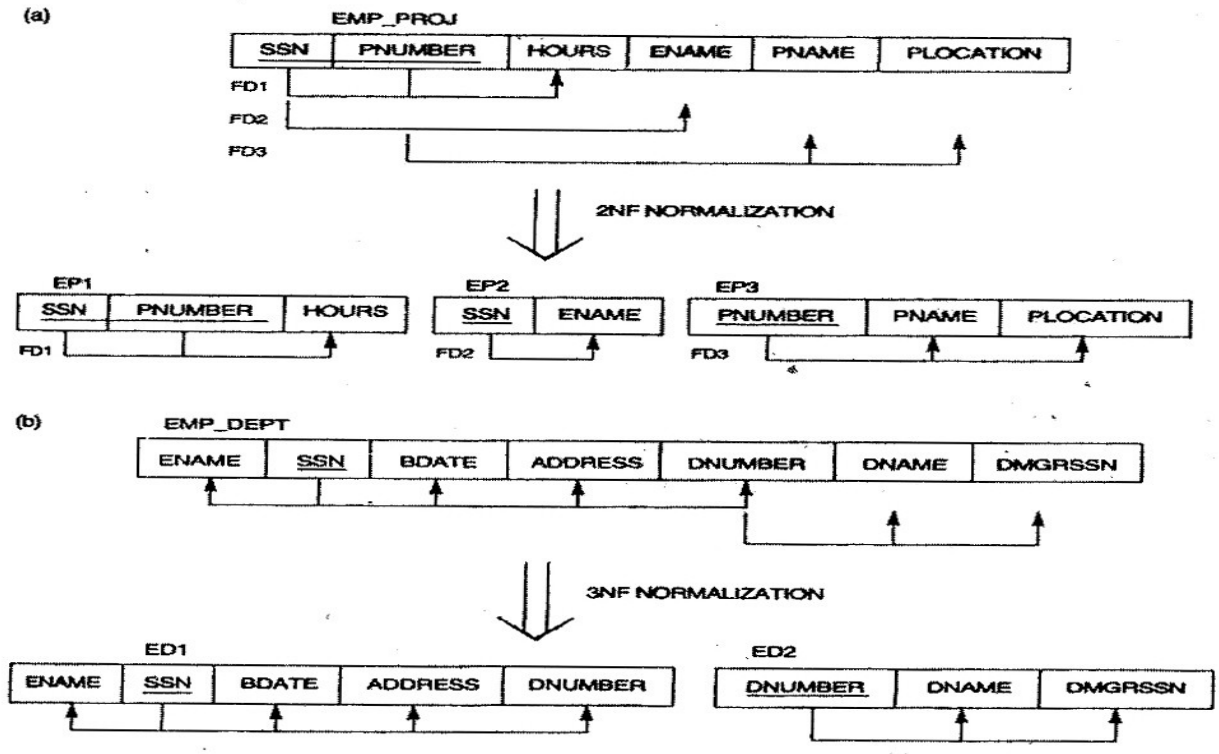




2. Second Normal Form (2NF) :

A second normal form is based on the concept of full functional dependency. A relation is in second normal form if every non-prime attribute A in R is fully functionally dependent on the Primary Key of R.

Emp_Project : Emp_Project **: 2NF and 3NF, (a) Normalizing EMP_PROJ into 2NF relations**



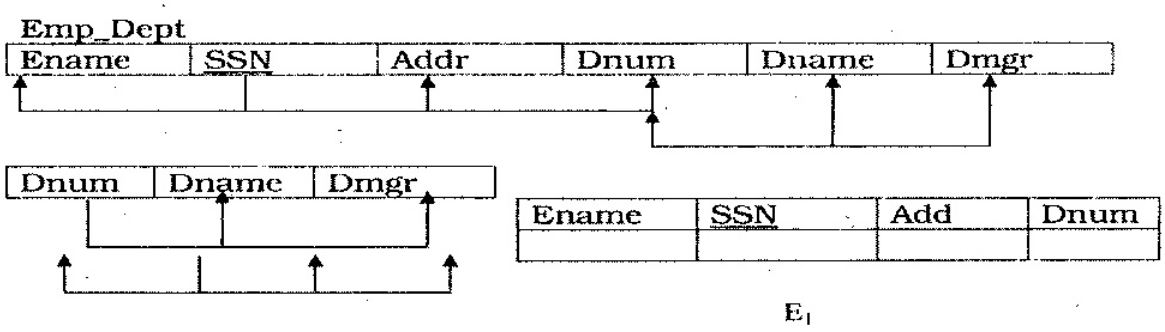
(b) Normalizing EMP_DEPT into 3NF relations

A Partial functional dependency is a functional dependency in which one or more non-key attributes are functionally dependent on part of the primary key. It creates a redundancy in that relation, which results in anomalies when the table is updated.

3. Third Normal Form (3NF) :

This is based on the concept of transitive dependency. We should design relational scheme in such a way that there should not be any transitive dependencies, because they lead to update anomalies. A functional dependence [FD] x->y in a relation schema 'R' is a transitive dependency. If there is a set of attributes 'Z' Le x-

>, z->y is transitive. The dependency SSN->Dmgr is transitive through Dnum in Emp_dept relation because SSN->Dnum and Dnum->Dmgr, Dnum is neither a key nor a subset [part] of the key.



E_1

According to codd's definition, a relational schema 'R' is in 3NF if it satisfies 2NF and no no_prime attribute is transitively dependent on the primary key. Emp_dept relation is not in 3NF, we can normalize the above table by decomposing into E1 and E2.

Note : Transitive is a mathematical relation that states that if a relation is true between the first value and the second value, and between the second value and the 3rd value, then it is true between the 1st and the 3rd value.

Example 2 :

Consider a relation schema 'Lots' which describes the parts of land for sale in various countries of a state. Suppose there are two candidate keys : properly_ID and {Country_name.lot#}; that is, lot numbers are unique only within each country, but property_ID numbers are unique across countries for entire state.

Based on the two candidate keys property_ID and {country name, Lot} we know that functional dependencies FD1 and FD2 hold. Suppose the following two additional functional dependencies hold in LOTS.

FD3 : Country_name -> tax_rate

FD4 : Area -> price

Here, FD3 says that the tax rate is fixed for a given country countryname -> taxrate, FD4 says that price of a Lot is determined by its area, area -> price. The Lots relation schema violates 2NF, because tax_rate is partially dependent upon candidate key {Country_namelot#}. Due to this, it decomposes lots relation into two relations – lots1 and lots 2.

Lots1 violates 3NF, because price is transitively dependent on candidate key of Lots1 via attribute area. Hence we could decompose LOTS1 into LOTS1A and LOTS1B.

1. It is fully functionally dependent on every key of 'R'
2. It is non_transitively dependent on every key of 'R'

Q. 6 : What do you mean by Shared Lock & Exclusive Lock ? Describe briefly two phase locking protocol ?

Ans. **Shared Lock :**

It is used for read only operations, i.e. used for operations that do not change or update the data.

E.g. SELECT statement:,

Shared locks allow concurrent transaction to read (SELECT) a data. No other transactions can modify the data while shared locks exist. Shared locks are released as soon as the data has been read.

Exclusive Locks :

Exclusive locks are used for data modification operations, such as UPDATE, DELETE and INSERT. It ensures that multiple updates cannot be made to the same resource simultaneously. No other transaction can read or modify data when locked by an exclusive lock.

Exclusive locks are held until transaction commits or rolls back since those are used for write operations.

There are three locking operations : read_lock(X), write_lock(X), and unlock(X). A lock associated with an item X, LOCK(X), now has three possible states : “read locked”, “write-locked”, or “unlocked”. A read-locked item is also called share-locked, because other transactions are allowed to read the item, whereas a write-locked item is called exclusive-locked, because a single transaction exclusive holds the lock on the item.

Each record on the lock table will have four fields : <data item name, LOCK, no_of_reads, locking_transaction(s)>, the value (state) of LOCK is either read-locked or write-locked.

Read_lock(X):

B, if LOCK(X)=’unlocked’

Then begin LOCK(X) “read-locked”

No_of_reads(x) 1

End

Else if LOCK(X)=”read-locked”

Then no_of_reads(X) no_of_reads(X)+1

else begin wait(until)LOCK(X)=”unlocked” and

the lock manager wakes up the transaction);

goto B

end;

write_lock(X):

B: if LOCK(X)=”unlocked”

Then LOCK(X) “wite-locked”;

else begin

wait(until LOCK(X)=”unlocked” and

the lock manager wkes up the transaction);

goto B

end;

unlock(X):

if LOCK(X)=”wite-locked”

Then begin LOCK(X) “un-locked”;

Wakeup one of the waiting transactions, if any

End

else if LOCK(X)=”read-locked”

then begin

no_of_reads(X) no_of_reads(X)-1

if no_of_reads(X)=0

then begin LOCK(X)="unlocked";

wakeup one of the waiting transactions, if any

end

end;

The Two Phase Locking Protocol

The two phase locking protocol is a process to access the shared resources as their own without creating deadlocks. This process consists of two phases.

1. Growing Phase : In this phase the transaction may acquire lock, but may not release any locks. Therefore this phase is also called as resource acquisition activity.
2. Shrinking Phase : In this phase the transaction may release locks, but may not acquire any new locks. This includes the modification of data and release locks. Here two activities are grouped together to form second phase.

In the beginning, transaction is in growing phase. Whenever lock is needed the transaction acquires it. As the lock is released, transaction enters the next phase and it can stop acquiring the new lock request.

Strict tow phase locking :

In the two phases locking protocol cascading rollback are not avoided. In order to avoid this slight modification are made to two phase locking and called strict two phase locking. In this phase all the locks are acquired by the transaction are kept on hold until the transaction commits.

Deadlock & starvation : In deadlock state there exists, a set of transactions in which every transaction in the set is waiting for another transaction in the set.

Suppose there exists a set of transactions waiting

{T1, T2, T3,……, Tn) such that T1 is waiting for a data item existing in T2, T2 for T3 etc… and Tn is waiting of T1. In this state none of the transaction will progress.

Assignment (Set-2)

Subject code: MI0034

Database Management Systems

Q. 1. Define Data Model & discuss the categories of Data Models? What is the difference between logical data Independence & Physical Data Independence?

Ans: **DATA MODEL:**

The product of the {database} design process which aims to identify and organize the required data logically and physically. A data model says what information is to be contained in a database, how the information will be used, and how the items in the database will be related to each other. For example, a data model might specify that a customer is represented by a customer name and credit card number and a product as a product code and price, and that there is a one-to-many relation between a customer and a product. It can be difficult to change a database layout once code has been written and data inserted. A well thought-out data model reduces the need for such changes. Data modelling enhances application maintainability and future systems may re-use parts of existing models, which should lower development costs. A data modelling language is a mathematical formalism with a notation for describing data structures and a set of operations used to manipulate and validate that data. One of the most widely used methods for developing data models is the {entity-relationship model}. The {relational model} is the most widely used type of data model. Another example is {NIAM}

Catagaries of DATA Model:-

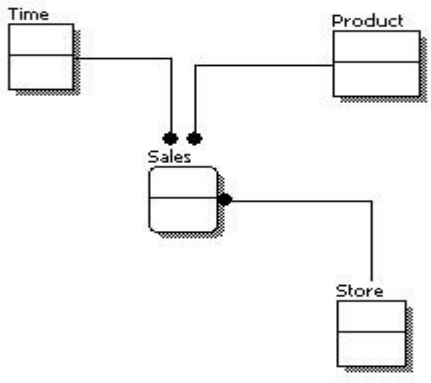
1. Conceptual (high-level, semantic) data models:

A **conceptual schema** or *conceptual data model* is a map of concepts and their relationships. This describes the semantics of an organization and represents a series of assertions about its nature. Specifically, it describes the things of significance to an organization (*entity classes*), about which it is inclined to collect information, and characteristics of (*attributes*) and associations between pairs of those things of significance (*relationships*).

Because a conceptual schema represents the semantics of an organization, and not a database design, it may exist on various levels of abstraction. The original ANSI four-schema architecture began with the set of *external schemas* that each represent one person's view of the world around him or her. These are consolidated into a single *conceptual schema* that is the superset of all of those external views. A data model can be as concrete as each person's perspective, but this tends to make it inflexible. If that person's world changes, the model must change. Conceptual data models take a more abstract perspective, identifying the fundamental things, of which the things an individual deals with are just examples.

A conceptual data model identifies the highest-level relationships between the different entities. Features of conceptual data model include:

- Includes the important entities and the relationships among them.
- No attribute is specified.
- No primary key is specified.



The figure is an example of a conceptual data model.

From the figure above, we can see that the only information shown via the conceptual data model is the entities that describe the data and the relationships between those entities. No other information is shown through the conceptual data model.

2. **Physical (low -level, internal) data models**

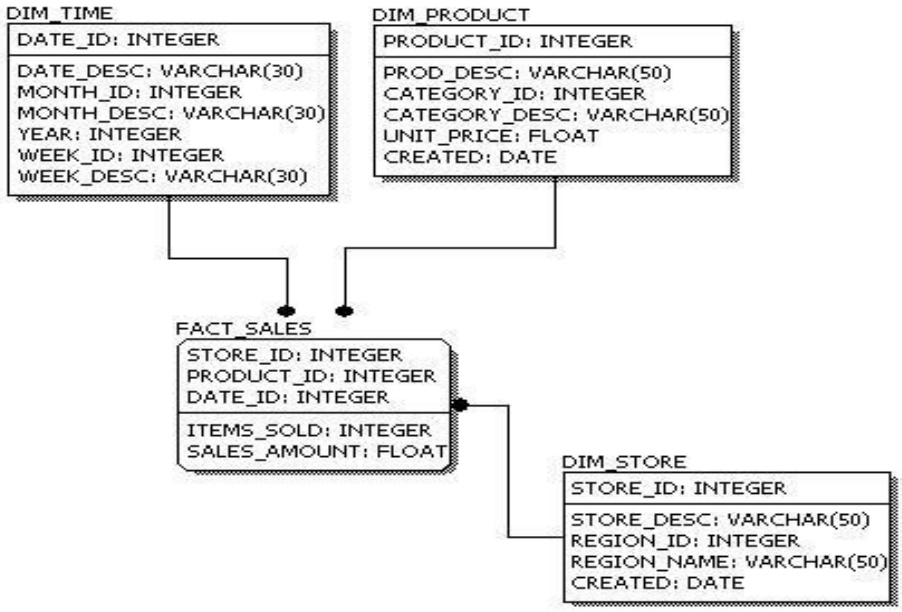
Features of physical data model include:

- Specification all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Denormalization may occur based on user requirements.
- Physical considerations may cause the physical data model to be quite different from the logical data model.

At this level, the data modeler will specify how the logical data model will be realized in the database schema.

The steps for physical data model design are as follows:

1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
4. Modify the physical data model based on physical constraints / requirements.



3. Logical Data Model

Features of logical data model include:

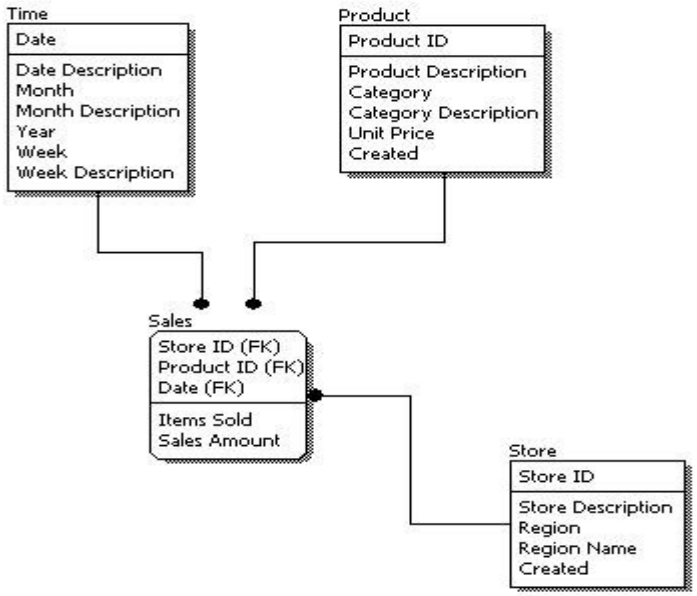
- Includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.

At this level, the data modeler attempts to describe the data in as much detail as possible, without regard to how they will be physically implemented in the database.

In data warehousing, it is common for the conceptual data model and the logical data model to be combined into a single step (deliverable).

The steps for designing the logical data model are as follows:

1. Identify all entities.
2. Specify primary keys for all entities.
3. Find the relationships between different entities.
4. Find all attributes for each entity.
5. Resolve many-to-many relationships.
6. Normalization.



Differences between a logical and physical data model

The difference between a logical and a physical data model are hard to grasp at first, but once you see the difference it seems obvious. A logical data model describes your model entities and how they relate to each other. A physical data model describes each entity in detail, including information about how you would implement the model using a particular (database) product.

In a logical model describing a person in a family tree, each person node would have attributes such as name(s), date of birth, place of birth, etc. The logical diagram would also show some kind of unique attribute or combination of attributes called a primary key that describes exactly one entry (a row in SQL) within this entity.

The physical model for the person would contain implementation details. These details are things like data types, indexes, constraints, etc.

The logical and physical model serves two different, but related purposes. A logical model is a way to draw your mental roadmap from a problem specification to an entity-based storage system. The user (problem owner) must understand and approve the

Q. 2. What is a B+Trees? Describe the structure of both internal and leaf nodes of a B+Tree?

Ans. **B+ Trees :**

In computer science, a **B-tree** is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic amortized time. The B-tree is a generalization of a binary search tree in that a node can have more than two children. (Comer, p. 123) Unlike self-balancing binary

search trees, the B-tree is optimized for systems that read and write large blocks of data. It is commonly used in databases and filesystems.

In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-defined range. When data is inserted or removed from a node, its number of child nodes changes. In order to maintain the pre-defined range, internal nodes may be joined or split. Because a range of child nodes is permitted, B-trees do not need re-balancing as frequently as other self-balancing search trees, but may waste some space, since nodes are not entirely full. The lower and upper bounds on the number of child nodes are typically fixed for a particular implementation. For example, in a 2-3 B-tree (often simply referred to as a **2-3 tree**), each internal node may have only 2 or 3 child nodes.

Each internal node of a B-tree will contain a number of keys. Usually, the number of keys is chosen to vary between d and $2d$. In practice, the keys take up the most space in a node. The factor of 2 will guarantee that nodes can be split or combined. If an internal node has $2d$ keys, then adding a key to that node can be accomplished by splitting the $2d$ key node into two d key nodes and adding the key to the parent node. Each split node has the required minimum number of keys. Similarly, if an internal node and its neighbor each have d keys, then a key may be deleted from the internal node by combining with its neighbor. Deleting the key would make the internal node have $d - 1$ keys; joining the neighbor would add d keys plus one more key brought down from the neighbor's parent. The result is an entirely full node of $2d$ keys.

The number of branches (or child nodes) from a node will be one more than the number of keys stored in the node. In a 2-3 B-tree, the internal nodes will store either one key (with two child nodes) or two keys (with three child nodes). A B-tree is sometimes described with the parameters $(d + 1)$ — $(2d + 1)$ or simply with the highest branching order, $(2d + 1)$.

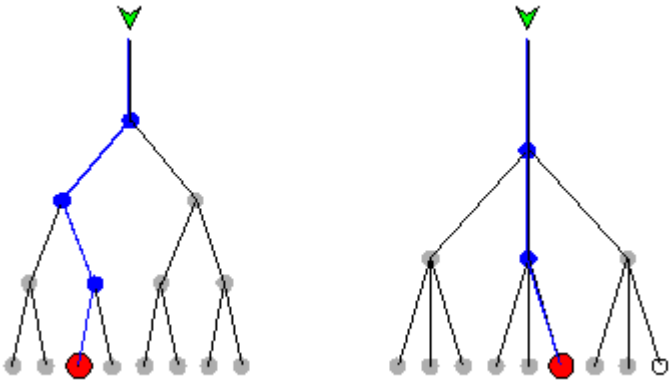
A B-tree is kept balanced by requiring that all leaf nodes are at the same depth. This depth will increase slowly as elements are added to the tree, but an increase in the overall depth is infrequent, and results in all leaf nodes being one more node further away from the root.

B-trees have substantial advantages over alternative implementations when node access times far exceed access times within nodes. This usually occurs when the nodes are in secondary storage such as disk drives. By maximizing the number of child nodes within each internal node, the height of the tree decreases and the number of expensive node accesses is reduced. In addition, rebalancing the tree occurs less often. The maximum number of child nodes depends on the information that must be stored for each child node and the size of a full disk block or an analogous size in secondary storage. While 2-3 B-trees are easier to explain, practical B-trees using secondary storage want a large number of child nodes to improve performance.

Variants

The term **B-tree** may refer to a specific design or it may refer to a general class of designs. In the narrow sense, a B-tree stores keys in its internal nodes but need not store those keys in the records at the leaves. The general class includes variations such as the B^+-tree and the B^*-tree.

- In the B^+-tree, copies of the keys are stored in the internal nodes; the keys and records are stored in leaves; in addition, a leaf node may include a pointer to the next leaf node to speed sequential access. (Comer, p. 129)
- The B^*-tree balances more neighboring internal nodes to keep the internal nodes more densely packed.(Comer, p. 129) For example, a non-root node of a B-tree must be only half full, but a non-root node of a B^*-tree must be two-thirds full.
- Counted B-trees store, with each pointer within the tree, the number of nodes in the subtree below that pointer.[1] This allows rapid searches for the Nth record in key order, or counting the number of records between any two records, and various other related operations.



3. Describe Projection operation, Set theoretic operation & join operation?

Projection Operator
Projection is also a Unary operator.
The Projection operator is pi:
Projection limits the attributes that will be returned from the original relation.
The general syntax is: attributes R
Where attributes is the list of attributes to be displayed and R is the relation.
The resulting relation will have the same number of tuples as the original relation (unless there are duplicate tuples produced).
The degree of the resulting relation may be equal to or less than that of the original relation.

Projection Examples
Assume the same EMP relation above is used.
Project only the names and departments of the employees:
name, dept (EMP)
Results:Name Dept
Smith CS
Jones Econ
Green Econ
Brown CS
Smith Fin

Combining Selection and Projection
The selection and projection operators can be combined to perform both operations.

Show the names of all employees working in the CS department:
name (Dept = 'CS' (EMP))
Results: Name
Smith
Brown

Show the name and rank of those Employees who are not in the CS department or Adjuncts:
name, rank ((Rank = 'Adjunct' Dept = 'CS') (EMP))
Result: Name Rank
Green Assistant
Smith Associate
Exercises
Evaluate the following expressions:
name, rank ((Rank = 'Adjunct' Dept = 'CS') (EMP))
fname, age (Age > 22 (R S))

For this expression, use R and S from the Set Theoretic Operations section above.

office > 300 (name, rank (EMP))

Aggregate Functions
We can also apply Aggregate functions to attributes and tuples:
SUM
MINIMUM
MAXIMUM
AVERAGE, MEAN, MEDIAN
COUNT
Aggregate functions are sometimes written using the Projection operator or the Script F character: as in the Elmasri/Navathe book.
Aggregate Function Examples
Assume the relation EMP has the following tuples:

Name	Office	Dept	Salary
Smith	400	CS	45000
Jones	220	Econ	35000
Green	160	Econ	50000
Brown	420	CS	65000
Smith	500	Fin	60000

Find the minimum Salary: MIN (salary) (EMP)
Results:MIN(salary)
35000

Find the average Salary: AVG (salary) (EMP)
Results:AVG(salary)
51000

Count the number of employees in the CS department: COUNT (name) (Dept = 'CS' (EMP))
Results:COUNT(name)
2

Find the total payroll for the Economics department: SUM (salary) (Dept = 'Econ' (EMP))
Results:SUM(salary)
85000

Set Theoretic Operations
Consider the following relations R and S

R First	Last	Age
Bill	Smith	22
Sally	Green	28
Mary	Keen	23
Tony	Jones	32

S First	Last	Age
Forrest	Gump	36
Sally	Green	28
DonJuan	DeMarco	27

Union: R S
Result: Relation with tuples from R and S with duplicates removed.

Difference: R - S
Result: Relation with tuples from R but not from S

Intersection: R S
Result: Relation with tuples that appear in both R and S.

R	S First	Last	Age
Bill	Smith	22	
Sally	Green	28	
Mary	Keen	23	
Tony	Jones	32	
Forrest	Gump	36	
DonJuan		DeMarco	27

R - S	First	Last	Age
Bill	Smith	22	
Mary	Keen	23	
Tony	Jones	32	

R	S First	Last	Age
Sally	Green	28	

Join Operation
Join operations bring together two relations and combine their attributes and tuples in a specific fashion.
The generic join operator (called the Theta Join is:
It takes as arguments the attributes from the two relations that are to be joined.
For example assume we have the EMP relation as above and a separate DEPART relation with (Dept, MainOffice, Phone) :
EMP EMP.Dept = DEPART.Dept DEPART

The join condition can be
When the join condition operator is = then we call this an Equijoin
Note that the attributes in common are repeated.
Join Examples
Assume we have the EMP relation from above and the following DEPART relation: Dept MainOffice Phone

CS	404	555-1212
Econ	200	555-1234
Fin	501	555-4321
Hist	100	555-9876

Find all information on every employee including their department info:
EMP emp.Dept = depart.Dept DEPART

Results:	Name	Office	EMP.Dept	Salary	DEPART.Dept	MainOffice	Phone
Smith	400	CS	45000	CS	404	555-1212	
Jones	220	Econ	35000	Econ	200	555-1234	
Green	160	Econ	50000	Econ	200	555-1234	
Brown	420	CS	65000	CS	404	555-1212	
Smith	500	Fin	60000	Fin	501	555-4321	

Find all information on every employee including their department info where the employee works in an office numbered less than the department main office:
EMP (emp.office < depart.mainoffice) (emp.dept = depart.dept) DEPART

Results:	Name	Office	EMP.Dept	Salary	DEPART.Dept	MainOffice	Phone
Smith	400	CS	45000	CS	404	555-1212	
Green	160	Econ	50000	Econ	200	555-1234	
Smith	500	Fin	60000	Fin	501	555-4321	

Natural Join
Notice in the generic (Theta) join operation, any attributes in common (such as dept above) are repeated.
The Natural Join operation removes these duplicate attributes.
The natural join operator is: *
We can also assume using * that the join condition will be = on the two attributes in common.

Example: EMP * DEPART

Results:Name	Office	Dept	Salary	MainOffice	Phone
Smith	400	CS	45000	404	555-1212
Jones	220	Econ	35000	200	555-1234
Green	160	Econ	50000	200	555-1234
Brown	420	CS	65000	404	555-1212
Smith	500	Fin	60000	501	555-4321

Outer Join

In the Join operations so far, only those tuples from both relations that satisfy the join condition are included in the output relation.

The Outer join includes other tuples as well according to a few rules.

Three types of outer joins:

Left Outer Join includes all tuples in the left hand relation and includes only those matching tuples from the right hand relation.

Right Outer Join includes all tuples in the right hand relation and includes ony those matching tuples from the left hand relation.

Full Outer Join includes all tuples in the left hand relation and from the right hand relation.

Examples:

Assume we have two relations: PEOPLE and MENU: PEOPLE:Name Age Food

Alice	21	Hamburger
Bill	24	Pizza
Carl	23	Beer
Dina	19	Shrimp

MENU:	Food	Day
	Pizza	Monday
	Hamburger	Tuesday
	Chicken	Wednesday
	Pasta	Thursday
	Tacos	Friday

PEOPLE people.food = menu.food MENU

Name	Age	people.Food	menu.Food	Day
Alice	21	Hamburger	Hamburger	Tuesday
Bill	24	Pizza	Pizza	Monday
Carl	23	Beer	NULL	NULL
Dina	19	Shrimp	NULL	NULL

PEOPLE people.food = menu.food MENU

Name	Age	people.Food	menu.Food	Day
Bill	24	Pizza	Pizza	Monday
Alice	21	Hamburger	Hamburger	Tuesday
NULL	NULL	NULL	Chicken	Wednesday
NULL	NULL	NULL	Pasta	Thursday
NULL	NULL	NULL	Tacos	Friday

PEOPLE people.food = menu.food MENU

Name	Age	people.Food	menu.Food	Day
Alice	21	Hamburger	Hamburger	Tuesday
Bill	24	Pizza	Pizza	Monday
Carl	23	Beer	NULL	NULL
Dina	19	Shrimp	NULL	NULL
NULL	NULL	NULL	Chicken	Wednesday
NULL	NULL	NULL	Pasta	Thursday
NULL	NULL	NULL	Tacos	Friday

Q. 4. Discuss Multi Table Queries?

Ans: **Multiple Table Queries :**

Most of the queries you create in Microsoft Access will more that likely need to include the data from more than one table and you will have to join the tables in the query. The capability to join tables is the power of the relational database.

As you know, in order to join database tables, they must have a field in common. The fields on which you join tables must be the same or compatible data types and they must contain the same kind of data, however they do not have to have the same field name (although they probably will). Occasionally, the two database tables that you want to bring the data from may not have a field in common and you will have to add another table to the query with the sole purpose of joining the tables.

Different types of query joins will return different sets of results. When creating new queries, it is prudent to test them on a set of records for which you know what the result should be. That's a good way to be sure that you have the correct join and are getting accurate results. Just because a query runs and doesn't give you an error doesn't mean that the resulting data set is what you intended to return.

Failure to join tables in a database query will result in a cross or Cartesian product (A Cartesian product is defined as all possible combinations of rows in all tables. Be sure you have joins before trying to return data, because a Cartesian product on tables with many records and/or on many tables could take several hours to complete.), in which every record in one table is joined with every record in the second table - probably not very meaningful data.

There are inner joins and outer joins - each with variations on the theme.
Inner Join

A join of two tables that returns records for which there is a matching value in the field on which the tables are joined.

The most common type of join is the inner join, or equi-join. It joins records in two tables when the values in the fields on which they are joined are equal. For example, if you had the following Customers and Orders tables and did an equi-join on (or, as is sometimes said, over) the CustomerID fields, you would see the set of records that have the same CustomerID in both tables. With the following data, that would be a total of 7 records. Customers listed in the Customers table who had not placed an order would not be included in the result. There has to be the same value in the CustomerID field in both tables.

Customers : Table

CustomerID	CustomerName
1	The Bookshop
2	Book Worm
3	Books Galore
4	Barries Bookstore
5	The Big Book
6	Books & More
7	The Front Cover
8	Inside Out
(AutoNumber)	

Record: 9 of

Orders : Table

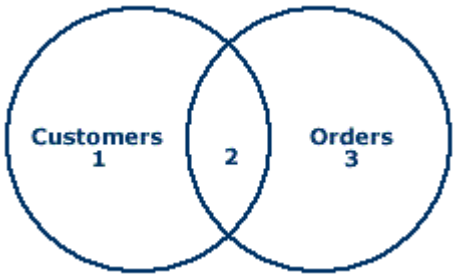
OrderNumber	CustomerID	OrderDate
76680	1	01/05/2004
78511	1	15/07/2004
79514	1	21/09/2004
71405	3	18/08/2004
79515	3	25/10/2004
70012	6	04/03/2004
78512	8	16/07/2004

Record: 8 of 8

An Inner Join of the Customer and Order Data

If, in the query result, you eliminated redundant columns - that is, displayed the CustomerID column only once in the result - this would be called a natural join.

An inner join returns the intersection of two tables. Following is a graphic of joining these tables. The Customers table contains data in areas 1 and 2. The Orders table contains data in areas 2 and 3. An inner join returns only the data in area 2.



Outer Joins

A join between two tables that returns all the records from one table and, from the second table, only those records in which there is a matching values in the field on which the tables are joined.

An outer join returns all the records from one table and only the records from the second table where the value in the field on which the tables are joined matches a value in the first table. Outer joins are referred to as left outer joins and right outer joins. The left and right concept comes from the fact that, in a traditional database diagram, the table on the one side of a 1:N relationship was drawn on the left.

Using our Customers and Orders tables again, if you performed a left outer join, the result would include a listing of all Customers and, for those that had placed orders, the data on those orders. You would get a total of 11 records from this data, which is a very different result from the 7 records provided by the inner join.

Customers : Table

CustomerID	CustomerName
1	The Bookshop
2	Book Worm
3	Books Galore
4	Barries Bookstore
5	The Big Book
6	Books & More
7	The Front Cover
8	Inside Out
(AutoNumber)	

Record: 9 of

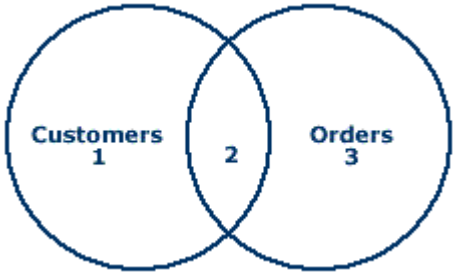
Orders : Table

OrderNumber	CustomerID	OrderDate
76680	1	01/05/2004
78511	1	15/07/2004
79514	1	21/09/2004
71405	3	18/08/2004
79515	3	25/10/2004
70012	6	04/03/2004
78512	8	16/07/2004

Record: 8 of 8

An Outer Join of the Customers and Orders table

In the diagram below, a left outer join on the Customers table will return the data in areas 1 and 2. By the way, this type of diagram is called a Venn diagram.



Not All Data Can Be Edited

Earlier, it was mentioned that the results of a query represent “live” data, meaning that a change to that data is actually a change to the data in the base table. However, you will find that you cannot edit all data that is

returned by a query. You can edit values in all fields from a query based on a single table or on two tables with a one-to-one relationship. But you can't edit all fields in a query based on tables with a one-to-many relationship nor from crosstab queries or
those with totals.

In general, you can edit:
all fields in a single table query
all fields in tables with a one-to-one relationship
all fields in the table on the many side of a one-to-many relationship
non-key fields in the table on the one side of a one-to-many relationship

You can't edit:
fields in the primary key in the table on the one side of a one-to-many relationship
fields returned by a crosstab query
values in queries in which aggregate operations are performed calculated fields

There are ways to work around some of these editing limitations but the precise technique will depend on the RDBMS you're using.

Q.5. Discuss Transaction Processing Concept? 10.2 Describe properties of Transactions?

Ans: In computer science, transaction processing is information processing that is divided into individual, indivisible operations, called transactions. Each transaction must succeed or fail as a complete unit; it cannot remain in an intermediate state.

Description

Transaction processing is designed to maintain a computer system (typically a database or some modern filesystems) in a known, consistent state, by ensuring that any operations carried out on the system that are interdependent are either all completed successfully or all canceled successfully.

For example, consider a typical banking transaction that involves moving $700 from a customer's savings account to a customer's checking account. This transaction is a single operation in the eyes of the bank, but it involves at least two separate operations in computer terms: debiting the savings account by $700, and crediting the checking account by $700. If the debit operation succeeds but the credit does not (or vice versa), the books of the bank will not balance at the end of the day. There must therefore be a way to ensure that either both operations succeed or both fail, so that there is never any inconsistency in the bank's database as a whole. Transaction processing is designed to provide this.

Transaction processing allows multiple individual operations to be linked together automatically as a single, indivisible transaction. The transaction-processing system ensures that either all operations in a transaction are completed without error, or none of them are. If some of the operations are completed but errors occur when the others are attempted, the transaction-processing system "rolls back" all of the operations of the transaction (including the successful ones), thereby erasing all traces of the transaction and restoring the system to the consistent, known state that it was in before processing of the transaction began. If all operations of a transaction are completed successfully, the transaction is committed by the system, and all changes to the database are made permanent; the transaction cannot be rolled back once this is done.

Transaction processing guards against hardware and software errors that might leave a transaction partially completed, with the system left in an unknown, inconsistent state. If the computer system crashes in the middle of a transaction, the transaction processing system guarantees that all operations in any uncommitted (i.e., not completely processed) transactions are cancelled.

Transactions are processed in a strict chronological order. If transaction n+1 intends to touch the same portion of the database as transaction n, transaction n+1 does not begin until transaction n is committed. Before any transaction is committed, all other transactions affecting the same part of the system must also be committed; there can be no "holes" in the sequence of preceding transactions.

Methodology

The basic principles of all transaction-processing systems are the same. However, the terminology may vary from one transaction-processing system to another, and the terms used below are not necessarily universal.

Rollback

Transaction-processing systems ensure database integrity by recording intermediate states of the database as it is modified, then using these records to restore the database to a known state if a transaction cannot be committed. For example, copies of information on the database prior to its modification by a transaction are set aside by the system before the transaction can make any modifications (this is sometimes called a before image). If any part of the transaction fails before it is committed, these copies are used to restore the database to the state it was in before the transaction began.

Rollforward

It is also possible to keep a separate journal of all modifications to a database (sometimes called after images); this is not required for rollback of failed transactions, but it is useful for updating the database in the event of a database failure, so some transaction-processing systems provide it. If the database fails entirely, it must be restored from the most recent back-up. The back-up will not reflect transactions committed since the back-up was made. However, once the database is restored, the journal of after images can be applied to the database (rollforward) to bring the database up to date. Any transactions in progress at the time of the failure can then be rolled back. The result is a database in a consistent, known state that includes the results of all transactions committed up to the moment of failure.

Deadlocks

In some cases, two transactions may, in the course of their processing, attempt to access the same portion of a database at the same time, in a way that prevents them from proceeding. For example, transaction A may access portion X of the database, and transaction B may access portion Y of the database. If, at that point, transaction A then tries to access portion Y of the database while transaction B tries to access portion X, a deadlock occurs, and neither transaction can move forward. Transaction-processing systems are designed to detect these deadlocks when they occur. Typically both transactions will be cancelled and rolled back, and then they will be started again in a different order, automatically, so that the deadlock doesn't occur again. Or sometimes, just one of the deadlocked transactions will be cancelled, rolled back, and automatically re-started after a short delay.

Deadlocks can also occur between three or more transactions. The more transactions involved, the more difficult they are to detect, to the point that transaction processing systems find there is a practical limit to the deadlocks they can detect.

Compensating transaction

In systems where commit and rollback mechanisms are not available or undesirable, a Compensating transaction is often used to undo failed transactions and restore the system to a previous state.

Transaction Properties

You can control the behavior of the transactions in your OpenAccess ORM applications by setting various transaction properties. The properties are always set for a specific transaction, and they are valid until the IObjectScope instance is disposed of. Transaction properties can be changed only if the transaction is not active.

The previous sections showed some transaction properties. In the following code example, the RetainValues property is set to true:C#

```
// prepare transaction properties

scope.TransactionProperties.RetainValues = true;
scope.Transaction.Begin();
```

...

```
ConsoleWriteLine( "RetainValues is "

        + scope.Transaction.Properties.RetainValues );

scope.Transaction.Commit();

// Properties are still valid

scope.Transaction.Begin();
...
```

VB.NET

```
' prepare transaction properties
scope.TransactionProperties.RetainValues = True
scope.Transaction.Begin()
'...
ConsoleWriteLine("RetainValues is " + scope.Transaction.Properties.RetainValues)
scope.Transaction.Commit()
' Properties are still valid
scope.Transaction.Begin()
```

Following is a list of the transaction properties, their allowed and default values, and a brief description.RetainValues

This property controls whether persistent class instances retain their values after commit of the transaction and if read access is allowed. By default it is set to true. However, regardless of this setting, objects are refreshed from the data store the next time the object is accessed within an active transaction.
RestoreValues

This property controls whether the values of objects are restored to their original values when a transaction (or one particular nesting level) is rolled back. By default it is set to false.
No Automatic Refreshes

As described earlier in this chapter, OpenAccess ORM uses optimistic concurrency control by default (refer to Concurrency Control Algorithms for more information about the various concurrency control mechanisms). This means that OpenAccess ORM does not validate read (but unmodified) objects at commit time, and therefore it is possible that if an object is read inside a transaction, it might be changed in the database, while the transaction is running.

So, in order to avoid long-living stale objects, OpenAccess ORM will refresh such objects if they are accessed in a subsequent transaction. This happens on the first access to such objects. Thus, only short-living stale objects are possible, at the cost of an SQL call for refreshing the object in a subsequent transaction.

It is possible to have more control over this refreshing behavior, by disabling the automatic refresh function, which can be done as shown below:

```
scope.TransactionProperties.RefreshReadObjectsInNewTransaction = false;
```

The advantage of using this is that objects can keep their data for a long time, without the need for executing an SQL Statement again.
However, if you enable "no automatic refreshes" then you are responsible for avoiding stale data, i.e. you will need to call Refresh() or Evict() at appropriate times.

Therefore, please use this with care, since read (but not modified) objects, will not be refreshed automatic`ally in new transactions of the same ObjectScope, i.e., if an object is fetched from the database in the first transaction and is subsequently never explicitly refreshed, evicted or modified in subsequent transactions, it will still have the values from the first transaction.

Concurrency

This property determines the concurrency settings of a transaction. The default is TransactionMode.OPTIMISTIC|TransactionMode.NO_LOST_UPDATES.
AutomaticBegin

This property allows the user to specify that every Commit()/Rollback() of a transaction will start a new transaction immediately. Therefore, the user does not need to call Begin(), and one can work with just Commit() and Rollback() calls. In other words, there is always a started transaction. This is regardless of a failure of a Commit() [the next transaction will yet be started].

This is especially useful for multithreaded applications, i.e., working with multiple threads in one object scope by setting the <option.Multithreaded> to "true", since this allows an automatic synchronized Commit() + Begin().
FailFast

This property determines whether a transaction commit or flush, should fail at the first failure. When this property is set to true (default value), the transaction will fail on the occurrence of the first OptimisticVerificationException. When this property is set to false the IObjectScope will collect all the failures, i.e., the commit or flush will continue and collect all the OptimisticVerificationExceptions that occur. It might be time consuming to collect all the failures, therefore this property is set to true by default.
This property should be set to false only when the information about failing objects is necessary, since it might be very time consuming to collect all the failures.

Q. 6. Describe the advantage of Distributed database? What is Client/server Model? Discuss briefly the security and Internet violation?

Ans: A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

Collections of data (e.g. in a database) can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. Replication and distribution of databases improve database performance at end-user worksites.

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does depend on the needs of the business and the sensitivity/confidentiality of the data to be stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

Advantages of distributed databases

Management of distributed data with different levels of transparency.
Increase reliability and availability.
Easier expansion.
Reflects organizational structure — database fragments are located in the departments they relate to.
Local autonomy — a department can control the data about them (as they are the ones familiar with it.)
Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations.
Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
Economics — it costs less to create a network of smaller computers with the power of a single large computer.
Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).

Reliable transactions - Due to replication of database.
Hardware, Operating System, Network, Fragmentation, DBMS, Replication and Location Independence.
Continuous operation.
Distributed Query processing.
Distributed Transaction management.

Single site failure does not affect performance of system. All transactions follow A.C.I.D. property: a-atomicity, the transaction takes place as whole or not at all; c-consistency, maps one consistent DB state to another; i-isolation, each transaction sees a consistent DB; d-durability, the results of a transaction must survive system failures. The Merge Replication Method used to consolidate the data between databases.

client–server model
The client–server model of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.[1] Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

The client–server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

Functions such as email exchange, web access and database access, are built on the client–server model. Users accessing banking services from their computer use a web browser client to send a request to a web server at a bank. That program may in turn forward the request to its own database client program that sends a request to a database server at another bank computer to retrieve the account information. The balance is returned to the bank database client, which in turn serves it back to the web browser client displaying the results to the user. The client–server model has become one of the central ideas of network computing. Many business applications being written today use the client–server model. So do the Internet's main application protocols, such as HTTP, SMTP, Telnet, and DNS.

The interaction between client and server is often described using sequence diagrams. Sequence diagrams are standardized in the Unified Modeling Language.

Specific types of clients include web browsers, email clients, and online chat clients.

Specific types of servers include web servers, ftp servers, application servers, database servers, name servers, mail servers, file servers, print servers, and terminal servers. Most web services are also types of servers.

Security

A condition that results from the establishment and maintenance of protective measures that ensures a state of inviolability from hostile acts or influences.

INTERNET VIOLATIONS

Internet crime is among the newest and most constantly evolving areas of American law. Although the Internet itself is more than three decades old, greater public usage began in the late 1980s with widespread ADOPTION only following in the 1990s. During that decade the Net was transformed from its modest military and academic roots into a global economic tool, used daily by over 100 million Americans and generating upwards of $100 billion in domestic revenue annually. But as many aspects of business, social, political, and cultural life moved online, so did crime, creating new challenges for lawmakers and law enforcement.

Crime on the Net takes both old and new forms. The medium has facilitated such traditional offenses as FRAUD and child PORNOGRAPHY. But it has also given rise to unique technological crimes, such as electronic intrusion in the form of hacking and computer viruses. High-speed Internet accounts helped fuel a proliferation of COPYRIGHT INFRINGEMENT in software, music, and movie PIRACY. National security is also threatened by the Internet's potential usefulness for TERRORISM. Taken together, these crimes have earned a new name: when FBI Director Louis J. Freeh addressed the U. S. Senate in 2000, he used the widely-accepted term "cybercrime."

Example: internet violation in school

Cause/Event	Consequences
Hacking into school servers; other computers	Suspension; administrative discretion and possible legal actions (action level III)
Using e-mail or Web sites to intimidate students (cyber-bullying)	Detention/Suspension; immediately sent to administrator; considered harassment under district Rights & Responsibilities Handbook
Downloading illegal music/media files from the Internet	Possible civil legal actions; data files wiped
Using inappropriate instant messaging/chats during class	Loss of computer for one or more class teacher/team discretion
Using or carrying computer in an unsafe manner	Loss of computer/ note to parents
Plagiarizing information by using the Internet	Failed assignment; loss of points and possible legal actions
Accessing pornographic/hate speech groups (others?) websites	Administrative discretion and possible legal actions; loss of computer privileges
Playing games on laptops or PDA's during class	ContentBarrier installed at parent's expense.
Physically damaging a computer through misuse or neglect (throwing, dropping, snagging)	Loss of computer; administrative discretion/restitution
Posing as a another person; misrepresenting self on the web, using another's identity (ID theft)	Suspension
Manipulating or changing settings without authorization	Administrative restrictions

Assignment (Set-1)

Subject code: MI0035

Computer Networks

Q. 1 : Explain all design issues for several layers in Computer. What is connection – oriented and connectionless service ?

Ans. **Design issues for the layers :**

The various key design issues are present in several layers in computer networks. The important design issues are :

1. **Addressing –** Mechanism for identifying senders and receivers, on the network need some form of addressing. There are multiple processes running on one machine. Some means is needed for a process on one machine to specify with whom it wants to communicate.
2. **Error Control –** There may be erroneous transmission due to several problems during communication. These are due to problem in communication circuits, physical medium, due to thermal noise and interference. Many error detecting and error correcting codes are known, but both ends of the connection must agree on which one being used. In addition, the receiver must have some mechanism of telling the sender which messages have been received correctly and which has not.
3. **Flow Control –** If there is a fast sender at one end sending data to a slow receiver, then there must be flow control mechanism to control the loss of data by slow receivers. There are several mechanisms used for flow control such as increasing buffer size at receivers, slow down the fast sender, and so on. Some process will not be in position to accept arbitrarily long messages. Then, there must be some mechanism to disassembling, transmitting and then reassembling messages.
4. **Multiplexing / de-multiplexing** – If the data has to be transmitted on transmission media separately, it is inconvenient or expensive to setup separate connection for each pair of communicating processes. So, multiplexing is needed in the physical layer at sender end and de-multiplexing is need at the receiver end.
5. **Routing** – When data has to be transmitted from source to destination, there may be multiple paths between them. An optimized (shortest) route muse be chosen. This decision is made on the basis of several routing algorithms, which chooses optimized route to the destination.

Connection Oriented and Connectionless Services :

Layers can offer two types of services namely connection oriented service and connectionless service.

Connection Oriented Service – The service user first establishes a connection, uses the connection and then releases the connection. Once the connection is established between source and destination, the path is fixed. The data transmission takes place through this path established. The order of the message sent will be same at the receiver end. Services are reliable and there is no loss of data. Most of the time, reliable service provides acknowledgement is an overhead and adds delay.

Connectionless Services – In this type of services, no connection is established between source and destination. Here there is no fixed path. Therefore, the messages must carry full destination address and each one of these messages are sent independent of each other. Messages sent will not be delivered at the destination in the same order. Thus, grouping and ordering is required at the receiver end, and the services are

not reliable. There is no acknowledgement confirmation from the receiver. Unreliable connectionless service is often called datagram service, which does not return an acknowledgement to the sender. In some cases, establishing a connection to send one short messages is needed. But reliability is required, and then acknowledgement datagram service can be used for these applications.

Another service is the request-reply service. In this type of service, the sender transmits a single datagram containing a request from the client side. Then at the other end, server reply will contain the answer. Request-reply is commonly used to implement communication in the client-server model.

Q.2 : Discuss OSI Reference model.

Ans. **The OSI Reference Model :**

The OSI model is based on a proposal developed by the International Standards Organization as a first step towards international standardization of the protocols used in the various layers. The model is called the ISO – (International Standard Organization – Open Systems Interconnection) Reference Model because it deals with connecting open systems – that is, systems that follow the standard are open for communication with other systems, irrespective of a manufacturer.

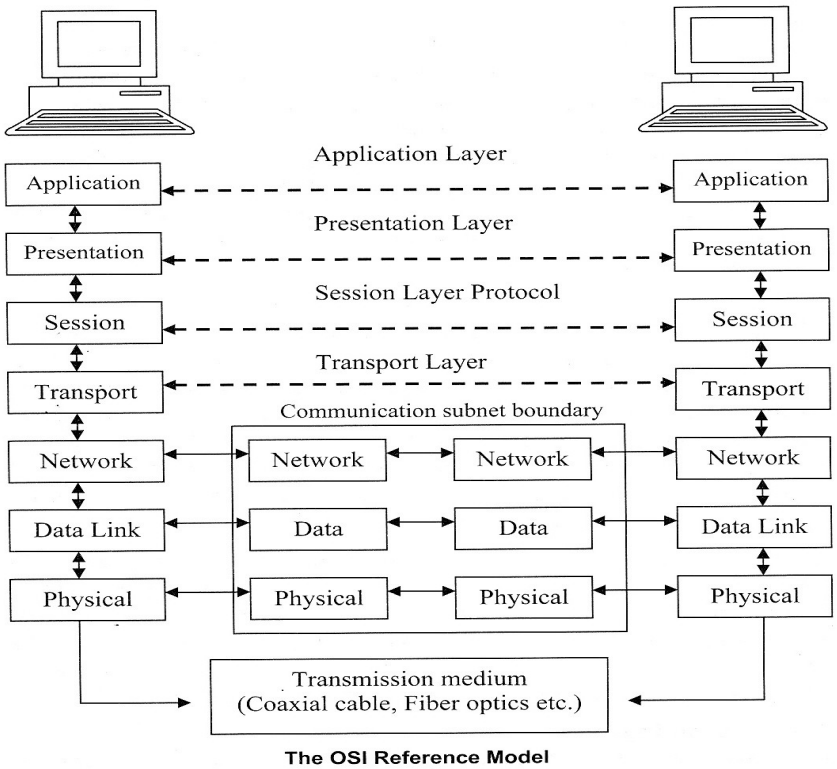
Its main objectives were to :

- Allow manufacturers of different systems to interconnect equipment through a standard interfaces.
- Allow software and hardware to integrate well and be portable on different systems.

The OSI model has seven layers. The principles that were applied to arrive at the seven layers are as follows :

1. Each layer should perform a well-defined function.
2. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.
3. The layer boundaries should be chosen to minimize the information flow across the interfaces.

The set of rules for communication between entities in a layer is called protocol for that layer.

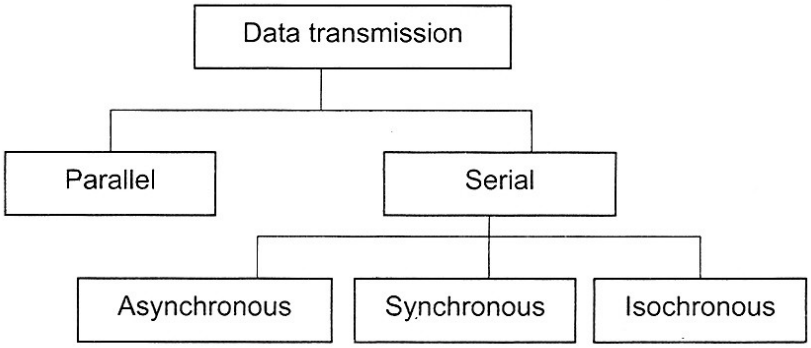


The OSI Reference Model

Q. 3 : Describe different types of Data Transmission Modes.

Ans. **Data Transmission Modes :**

The transmission of binary data across a link can be accomplished in either parallel or serial mode. In parallel mode, multiple bits are sent with each clock tick. In serial mode, 1 bit is sent with each clock tick. While there is one way to send parallel data, there are three subclasses of serial transmission : asynchronous, synchronous, and isochronous.



Data transmission and modes

Serial and Parallel

Serial Transmission :

In serial transmission one bit follows another, so we need only one communication channel rather than n to transmit data between two communicating devices.

The advantages of serial over parallel transmission is that with only one communication channel, serial transmission reduces cost of transmission over parallel by roughly a factor of n.

Since communication within devices is parallel, conversion devices are required at the interface between the sender and the line (parallel-to-serial) and between the line and the receiver (serial-to-parallel). Serial transmission occurs in one of three ways : asynchronous, synchronous, and isochronous.

Parallel Transmission :

Binary data, consisting of 1s and 0s, may be organized into groups of n bits each. Computers produce and consume data in groups of bits much as we conceive of and use spoken language in the form of words rather than letters. By grouping, we can send data n bits at a time instead of 1. This is called parallel transmission.

The mechanism for parallel transmission is a simple one : Use n wires to send n bits at one time. That way each bit has its own wire, and all n bits of one group can be transmitted with each clock tick from one device to another.

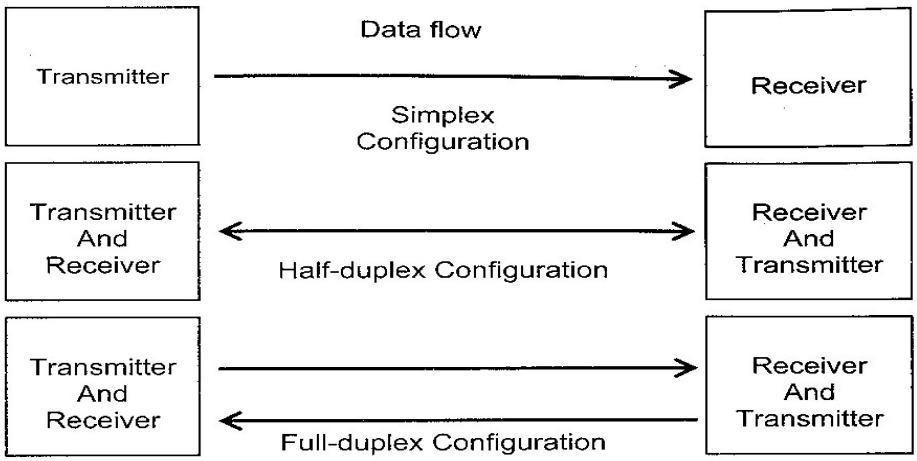
The advantage of parallel transmission is speed. All else being equal, parallel transmission can increase the transfer speed by a factor on n over serial transmission.

But there is a significant disadvantage : cost. Parallel transmission requires n communication lines just to transmit the data stream. Because this is expensive, parallel transmission is usually limited to short distances.

Simplex, Half-duplex and Full-duplex :

There are three modes of data transmission that correspond to the three types of circuits available. These are :

a) Simplex
b) Half-duplex
c) Full-duplex



Different Modes of Data Transmission

Simplex :

Simplex communications imply a simple method of communicating, which they are. In simplex communication mode, there is a one-way communication transmission. Television transmission is a good example of simplex communications. The main transmitter sends out a signal (broadcast), but it does not expect a reply as the receiving units cannot issue a reply back to the transmitter. A data collection terminal on a factory floor or a line printer (receive only). Another example of simplex communication is a keyboard attached to a computer because the keyboard can only send data to the computer.

At first thought it might appear adequate for many types of application in which flow of information is unidirectional. However, in almost all data processing applications, communication in both directions is required. Even for a "one-way" flow of information from a terminal to computer, the system will be designed to allow the computer to signal the terminal that data has been received. Without this capability, the remote used might enter data and never know that it was not received by the other terminal. Hence, simplex circuits are seldom used because a return path is generally needed to send acknowledgement, control or error signals.

Half-duplex :

In half-duplex mode, both units communicate over the same medium, but only one unit can send at a time. While one is in send mode, the other unit is in receiving mode. It is like two polite people talking to each other – one talks, the other listens, but neither one talks at the same time. Thus, a half-duplex line can alternately send and receive data. It requires two wires. This is the most common type of transmission for voice communications because only one person is supposed to speak at a time. It is also used to connect a terminal with a computer. The terminal might transmit data and then the computer responds with an acknowledgement. The transmission of data to and from a hard disk is also done in half-duplex mode.

Full-duplex :

In a half-duplex system, the line must be "turned around" each time the direction is reversed. This involves a special switching circuit and requires a small amount of time (approximately 150 milliseconds). With high speed capabilities of the computer, this turn-around time is unacceptable in many instances. Also, some applications

require simultaneous transmission in both directions. In such cases, a full-duplex system is used that allows information to flow simultaneously in both directions on the transmission path. Use of a full-duplex line improves efficiency as the line turn-around time required in a half-duplex arrangement is eliminated. It requires four wires.

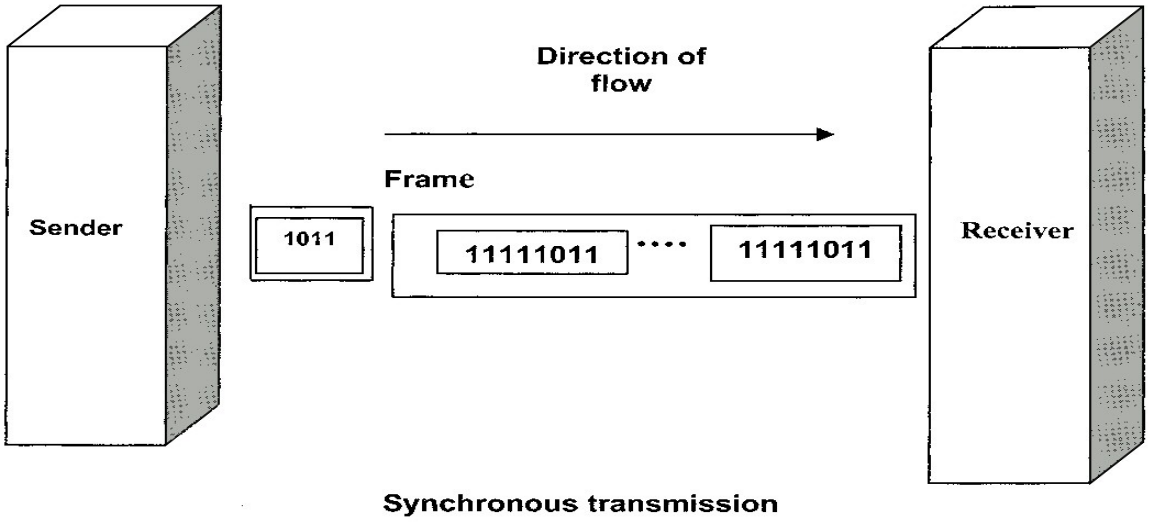
Synchronous and Asynchronous Transmission :

Synchronous Transmission :

In synchronous transmission, the bit stream is combined into longer "frames", which may contain multiple bytes. Each byte, however, is introduced onto the transmission link without a gap between it and the next one. It is left to the receiver to separate the bit stream into bytes for decoding purpose. In other words, data are transmitted as an unbroken sting of 1s and 0s, and the receiver separates that string into the bytes, or characters, it needs to reconstruct the information.

Without gaps and start and stop bits, there is no built-in mechanism to help the receiving device adjust its bits synchronization midstream. Timing becomes very important, therefore, because the accuracy of the received information is completely dependent on the ability of the receiving device to keep an accurate count of the bits as they come in.

The advantage of synchronous transmission is speed. With no extra bits or gaps to introduce at the sending end and remove at the receiving end, and, by extension, with fewer bits to move across the link, synchronous transmission is faster than asynchronous transmission of data from one computer to another. Byte synchronization is accomplished in the data link layer.



Synchronous transmission

Asynchronous Transmission :

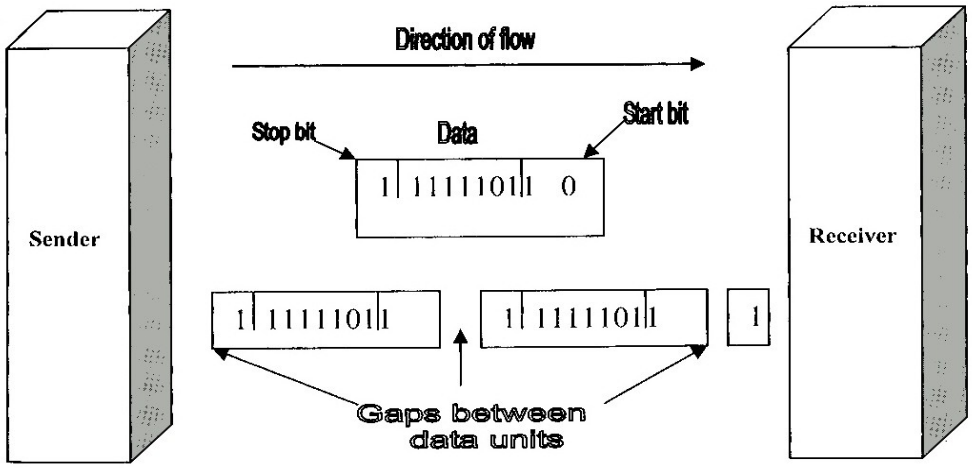
Asynchronous transmission is so named because the timing of a signal is unimportant. Instead, information is received and translated by agreed upon patterns. As long as those patterns are followed, the receiving device can retrieve the information without regard to the rhythm in which it is sent. Patterns are based on grouping the bit stream into bytes. Each group usually 8 bits, is sent along the link as a unit. The sending system handles each group independently, relaying it to the link whenever ready, without regard to t timer.

Without synchronization, the receiver cannot use timing to predict when the next group will arrive. To alert the receiver to the arrival of an new group, therefore, an extra bit is added to the beginning of each byte. This bit, usually a 0, is called the start bit. To let the receiver know that the byte is finished, 1 or more additional bits are appended to the end of the byte. These bits, usually 1s, are called stop bits.

By this method, each byte is increased in size to at least 10 bits, of which 8 bits is information and 2 bits or more are signals to the receiver. In addition, the transmission of each byte may then be followed by a gap of varying duration. This gap can be represented either by an idle channel or by a stream of additional stop bits.

The start and stop bits and the gap alert the receiver to the beginning and end of the each byte and also it to synchronize with the data stream. This mechanism is called asynchronous because, at the byte level, the sender and receiver do not have to be synchronized. But within each byte, the receiver must still by synchronized with the incoming bit stream.

That is, some synchronization is required, but only for the duration of a single byte. The receiving device resynchronizes at the onset of each new byte. When the receiver detects a start bit, it sets a timer and begins counting bits as they come in. After n bits, the receiver looks for a stop bit. As soon as it detects the stop bit, it waits until it detects the next start bit.



Asynchronous transmission

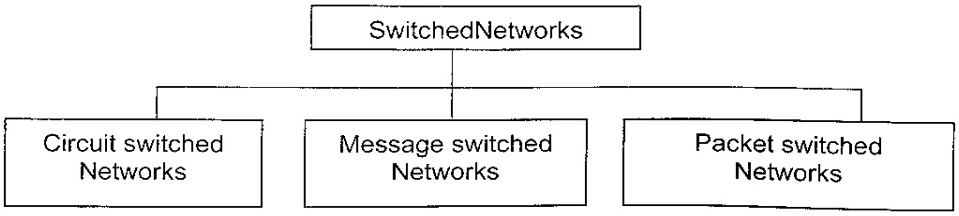
Isochronous Transmission :

In real-time audio and video, in which uneven delays between frames are not acceptable, synchronous transmission fails. For example, TV images are broadcast at the rate of 30 images per second; they must be viewed at the same rate. If each image is send by using one or more frames, there should be no delays between frames. For this type of application, synchronization between characters is not enough; the entire stream of bits must be synchronized. The isochronous transmission guarantees that the data arrive at a fixed rate.

Q. 4 : Define Switching. What is the difference between Circuit Switching and Packet Switching ?

Ans. **Switching :**

A network is a set of connected devices. Whenever we have multiple devices, we have the problem of how to connect them to make one-to-one communication possible. One of the better solutions is switching. A switch is network consists of a series of interlinked nodes, called switches. Switches are devices capable of crating temporary connections between two or more devices linked to the switch. In a switched network, some of these nodes are connected to the end systems (computers or telephones). Others are used only for routing. Switched networks are divided.



Different types of switching techniques

Difference between Circuit Switching and Packet Switching

Item	Circuit Switching	Packet Switching
What is send	Voice	Message (divided)
Call setup	Required	Not required
Dedicated Physical Path	Yes	No
Each packet follows the same route	Yes	No
Packets arrive in order	Yes	No
Is a switch crash is fatal	Yes	No
Bandwidth available	Yes	No
Time of possible congestion	At setup time	On every packet
Store-and-forward	No	Yes

Q.5 : Classify Guided Medium (wired). Compare Fiber Optics and Copper Wire.

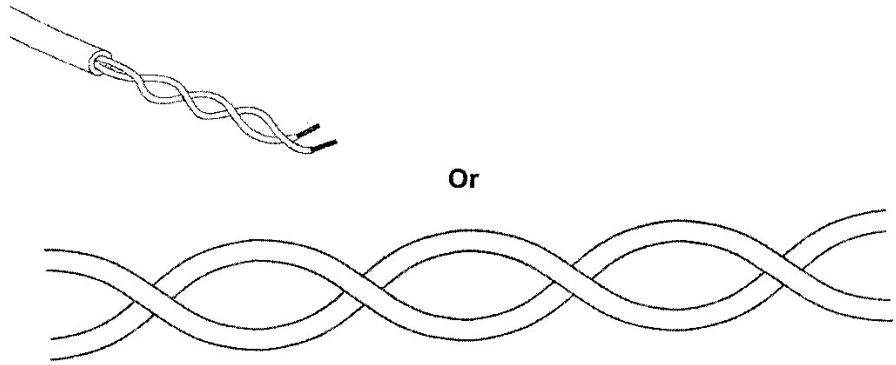
Ans. **Guided Transmission Medium (wired) :**

Guided media, which are those that provide a conduit form one device to another, include twisted-pair cable, coaxial cable, and fiber-optic cable. A single traveling along any of these media is directed and contained by

the physical limits of the medium. Twisted-pair and coaxial cable use metallic (copper) conductors that accept and transport signals on the form of electric current. Optical fiber is a cable that accepts and transports signals in the form of light.

1. Twisted Pair :

A twisted pair consists of two insulted copper wires, typically about 1 mm thick. The wires are twisted together in a helical form, just like a DNA molecule.



A twisted pair wire

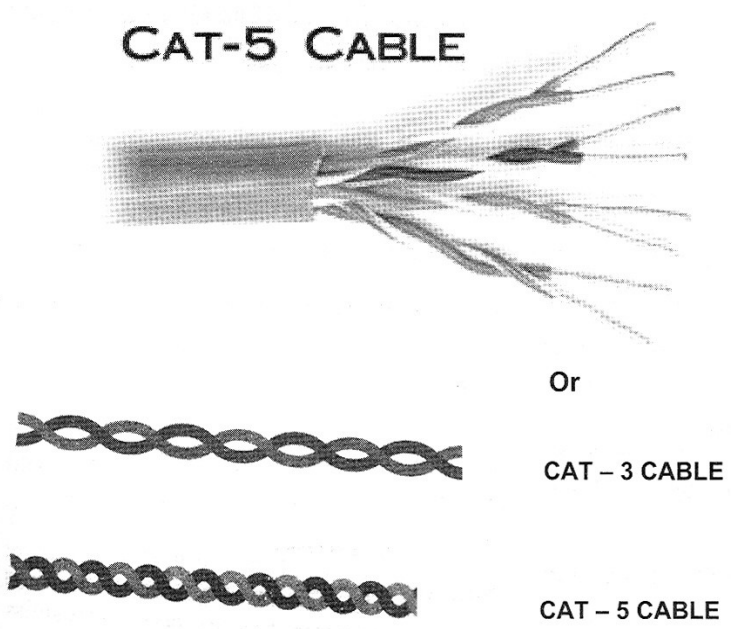
Twisting is done because two parallel wires constitute a fine antenna. When the wires are twisted, the waves from different twists cancel out, so the wire radiates less effectively.

The most common application of the twisted pair is the telephone system. All the telephones are connected to the telco office is by twisted pair. It runs several kilometers without amplification, but for long distance, repeaters are needed. If many wires are coming from one building or apartment, they are bundled together and encased in a protective sheath.

Twisted pairs can be used for transmitting either analog or digital signals. The bandwidth depends on the thickness of the wire and the distance traveled, but several megabits/sec can be achieved for a new kilometers. Due to their adequate performance and low cost, twisted pairs are widely used and are likely to remain so for years to come.

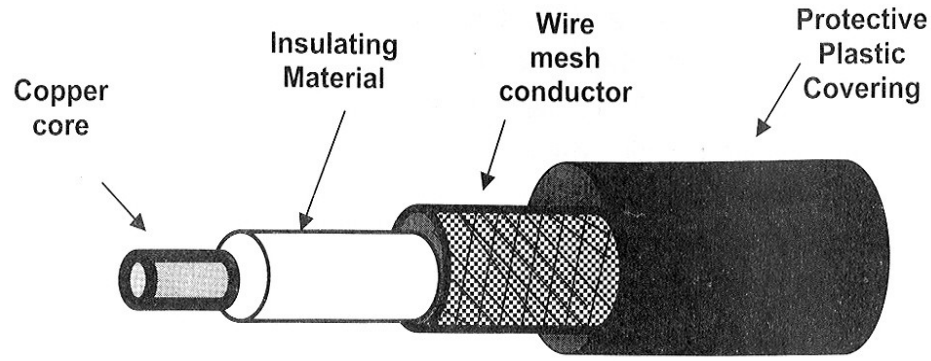
Twisted pair cabling comes in several varieties, two of which are important for computer networks :

1. Category 3 twisted pairs consist of two insulated wires gently twisted together. Four such pairs are typically grouped in a plastic sheath to protect the wires and keep them together. Which are capable of handling signals with bandwidth of 16 MHz. This scheme allowed up to four regular telephones or two multi-line telephones in each office to connect to the telephone company equipment in the wiring closet.
2. Category 5 twisted pairs similar to category 3 pairs, but with more twists per centimeter, which result in less crosstalk and a better-quality signals over longer distances, making them more suitable for high-speed computer communication. Which are capable of handling signals with bandwidth of 100 MHz ?



2. Coaxial Cable :

The coaxial cable consists of a stiff copper wire as the core, surrounded by an insulating material. The insulator is encased by a cylindrical conductor, often as a closely-woven braided mesh. The outer conductor is covered in a protective plastic sheath.



Coaxial Cable

The construction and shielding of the coaxial cable give it a good combination of high bandwidth and excellent noise immunity. The bandwidth possible depends on the cable quality, length, and single-to-noise ratio of the data signal, but coaxial cables have close bandwidth of 1 GHz.

Coaxial cable widely used within the telephone system for long-distance lines but now replaced by the fiber optics. Coax is still widely used for cable television and metropolitan area networks.

3. Optical Fiber :

A fiber-optic cable is made of glass or plastic and transmits signals in the form of light. To understand optical fiber, we need to components of optical fiber.

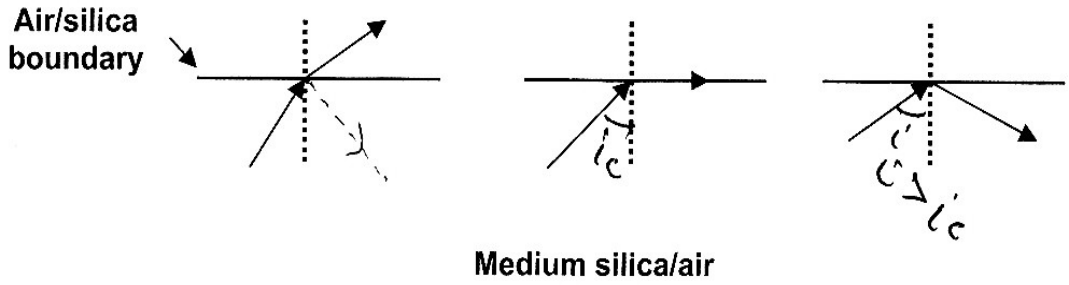
An optical transmission system has three key components :

1. The light source
2. The transmission medium
3. The detector

A pulse of light indicates a 1 bit and the absence of light indicates a 0 bit. The transmission medium is an ultra-thin fiber of glass. The detector generates an electrical pulse when light falls on it. By attaching a light source to one end of an optical fiber and a detector to other, we have unidirectional data transmission system that accepts an electrical signal, converts and transmits it by light pulses, and then reconverts the output to an electrical signal at the receiving end.

When light passes from one medium to another, for example, from fused silica to air, the ray is refracted (bend) at the silica/air boundary.

Here we see a light ray incident on boundary at an angle s1 emerging at an angle a1. The amount of refraction depends on the properties of the two media. For angles of incidence above certain critical value, the light is refracted back into the silica; none of its escapes into the air. Thus, a light ray incident at or above the critical angle is trapped inside the fiber, and can propagate for many kilometers with virtually no loss.



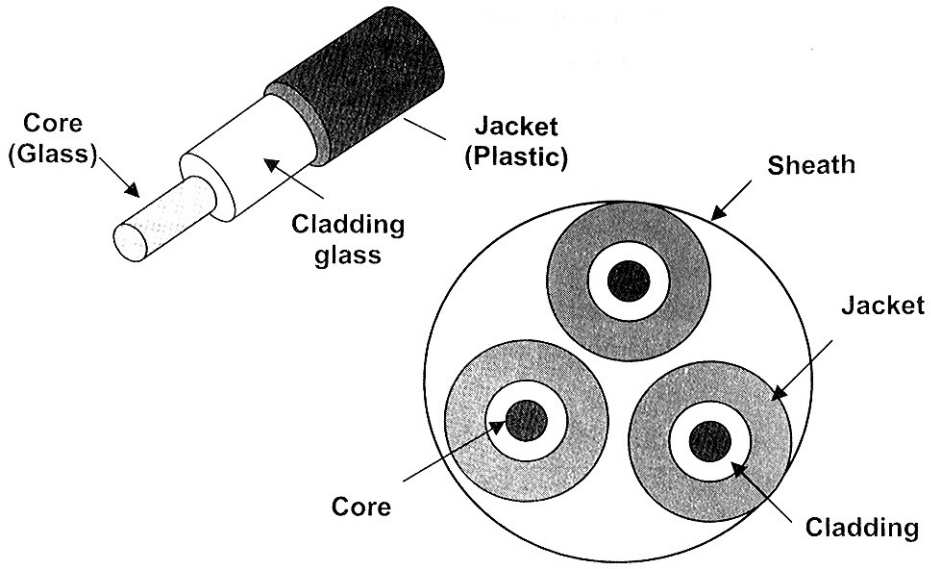
Medium silica/air

As the light ray trapped by total internal reflection in a medium. Like this many different rays will be bouncing around at different angles. Each ray is said to have a different modes. So a fiber having this property is called a multimode fiber.

If the fiber's diameter is reduced to a few wavelengths of light, the fiber acts like a wave guide, and the light can propagate only in a straight line, without bouncing, yielding a single-mode fiber.

Fiber optic cables are similar to coax. At the center is the glass core through which the light propagates. In multimode fibers, the core is typically 50 microns in diameter, about the thickness of human hair. In single-mode fibers, the core is 8 to 10 microns.

The core is surrounded by a glass cladding with a lower index of refraction than the core, to keep all the light in the core. Next comes a thin plastic jacket to protect the classing. Fibers are typically grouped in bundles, protected by an outer sheath.



Side view of single fiber and End view of a sheath with three fibers

Comparison of Fiber Optics and Copper Wire :

Fiber has many advantages over copper wire as a transmission media. These are :

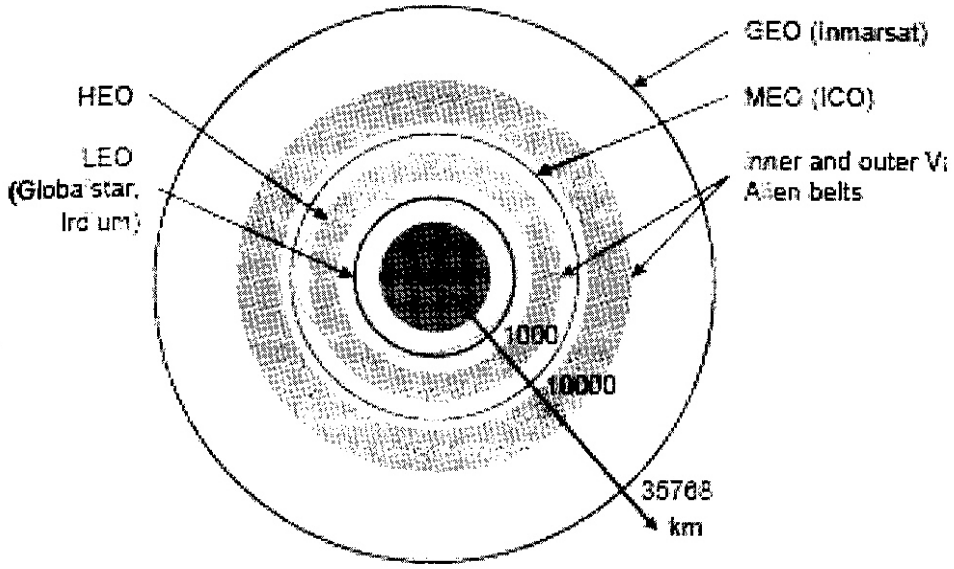
- It can handle much higher band widths than copper. Due to the low attenuation, repeaters are needed only about every 30 km. on long lines, versus about every 5 km. for copper.
- Fiber is not being affected by the power surges, electromagnetic interference, or power failures. Not it is affected by corrosive chemicals in the air, making it deal for harsh factory environment.
- Fiber is lighter the copper. One thousand twisted pairs copper cables of 1 km. long weight 8000 kg. But two fibers have more capacity and weigh only 100 kg., which greatly reduces the need for expensive mechanical support systems that must be maintained.
- Fibers do not leak light and are quite difficult to tap. This gives them excellent security against potential wire-tappers.
- If new routes designed, the fiber is the first choice because of lower installation cost.

Q. 6 : What are different types of Satellites ?

Ans. **Classification of Satellites :**

Four different types of satellite orbits can be identified depending on the shape and diameter of the orbit :

- GEO (Geostationary Orbit)
- LEO (Low Earth Orbit)
- MEO (Medium Earth Orbit) or ICO (Intermediate Circular Orbit)
- HEO (Highly Elliptical Orbit) elliptical orbits



Van-Allen-Belts; ionized particles 2000 – 6000 km. and 15000 – 30000 km. above earth surface.

1. **GEO (Geostationary Orbit) :**

   Altitude :

   Ca. 36000 km. above earth surface.

   Coverage :

   Ideally suited for continuous, regional coverage using a single satellite. Can also be used equally effectively for global coverage using a minimum of three satellites.

   Visibility :

   Mobile to satellite visibility decreases with increased latitude of the user. Poor Visibility in built-up, urban regions.

2. **LEO (Low Earth Orbit) :**

   Altitude :

   Ca. 500 – 1500 km.

   Coverage :

   Multi-satellite constellations of upwards of 30-50 satellites are required for global, continuous coverage. Single satellites can be used in store and forward mode for localized coverage but only appear for short periods of time.

   Visibility :

   The use of satellite diversity, by which more than one satellite is visible at any given time, can be used to optimize the link. This can be achieved by either selecting the optimum link or combining the reception of two or more links. The higher the guaranteed minimum elevation angle to the user, the more satellites is needed in the constellation.

3. **MEO (Medium Earth Orbit) :**

   Altitude :

   Ca. 6000 – 20000 km.

   Coverage :

   Multi-satellite constellations of between 10 and 20 satellites are required for global coverage.

   Visibility :

   Good to excellent global visibility, augmented by the use of satellite diversity techniques.

4. **HEO (Highly Elliptical Orbit) :**

   Altitude :

   Apogee : 40 000 – 50 000 km., Perigee : 1000-20 000 km.

   Coverage :

   Three or four satellites are needed to provide continuous coverage to a region.

   Visibility :

   Particularly designed to provide high guaranteed elevation angle to satellite for Northern and Southern temperate latitudes.

Assignment (Set-2)

Subject code: MI0035

Computer Networks

Q.1 Write down the features of Fast Ethernet and Gigabit Ethernet.

Ans: **Fast Ethernet**

In computer networking, Fast Ethernet is a collective term for a number of Ethernet standards that carry traffic at the nominal rate of 100 Mbit/s, against the original Ethernet speed of 10 Mbit/s. Of the fast Ethernet standards 100BASE-TX is by far the most common and is supported by the vast majority of Ethernet hardware currently produced. Fast Ethernet was introduced in 1995[1] and remained the fastest version of Ethernet for three years before being superseded by gigabit Ethernet.[2]
A fast Ethernet adapter can be logically divided into a Media Access Controller (MAC) which deals with the higher level issues of medium availability and a Physical Layer Interface (PHY). The MAC may be linked to the PHY by a 4 bit 25 MHz synchronous parallel interface known as a Media Independent Interface (MII) or a 2 bit 50 MHz variant Reduced Media Independent Interface (RMII). Repeaters (hubs) are also allowed and connect to multiple PHYs for their different interfaces.

The MII may (rarely) be an external connection but is usually a connection between ICs in a network adapter or even within a single IC. The specs are written based on the assumption that the interface between MAC and PHY will be a MII but they do not require it.

The MII fixes the theoretical maximum data bit rate for all versions of fast Ethernet to 100 Mbit/s. The data signaling rate actually observed on real networks is less than the theoretical maximum, due to the necessary header and trailer (addressing and error-detection bits) on every frame, the occasional "lost frame" due to noise, and time waiting after each sent frame for other devices on the network to finish transmitting

100BASE-TX is the predominant form of Fast Ethernet, and runs over two wire-pairs inside a category 5 or above cable (a typical category 5 cable contains 4 pairs and can therefore support two 100BASE-TX links). Like 10BASE-T, the proper pairs are the orange and green pairs (canonical second and third pairs) in TIA/EIA-568-B's termination standards, T568A or T568B. These pairs use pins 1, 2, 3 and 6.

In T568A and T568B, wires are in the order 1, 2, 3, 6, 4, 5, 7, 8 on the modular jack at each end. The color-order would be green/white, green, orange/white, blue, blue/white, orange, brown/white, brown for T568A, and orange/white, orange, green/white, blue, blue/white, green, brown/white, brown for T568B.

Each network segment can have a maximum distance of 100 metres (328 ft). In its typical configuration, 100BASE-TX uses one pair of twisted wires in each direction, providing 100 Mbit/s of throughput in each direction (full-duplex). See IEEE 802.3 for more details.

Gigabit Ethernet

Gigabit Ethernet (GbE or 1 GigE) is a term describing various technologies for transmitting Ethernet frames at a rate of a gigabit per second, as defined by the IEEE 802.3-2008 standard. Half-duplex gigabit links connected through hubs are allowed by the specification but in the marketplace full-duplex with switches are normal.

Intel PRO/1000 GT PCI network interface cardContents

IEEE 802.3ab, ratified in 1999, defines gigabit Ethernet transmission over unshielded twisted pair (UTP) category 5, 5e, or 6 cabling and became known as 1000BASE-T. With the ratification of 802.3ab, gigabit Ethernet became a desktop technology as organizations could use their existing copper cabling infrastructure.

IEEE 802.3ah, ratified in 2004 added two more Gigabit fiber standards, 1000BASE-LX10 (which was already widely implemented as vendor specific extension) and 1000BASE-BX10. This was part of a larger group of protocols known as Ethernet in the First Mile.

Initially, gigabit Ethernet was deployed in high-capacity backbone network links (for instance, on a high-capacity campus network). In 2000, Apple's Power Mac G4 and PowerBook G4 were the first mass produced personal computers featuring the 1000BASE-T connection.[1] It quickly became a built-in feature in many other computers. As of 2009 Gigabit NICs (1000BASE-T) are included in almost all desktop and server computer systems.

Higher bandwidth 10 Gigabit Ethernet standards have since become available as the IEEE ratified a fiber-based standard in 2002, and a twisted pair standard in 2006. As of 2009 10Gb Ethernet is replacing 1Gb as the backbone network and has begun to migrate down to high-end server systems.[citation needed]

Varieties

There are five different physical layer standards for gigabit Ethernet using optical fiber (1000BASE-X), twisted pair cable (1000BASE-T), or balanced copper cable (1000BASE-CX).

The IEEE 802.3z standard includes 1000BASE-SX for transmission over multi-mode fiber, 1000BASE-LX for transmission over single-mode fiber, and the nearly obsolete 1000BASE-CX for transmission over balanced copper cabling. These standards use 8b/10b encoding, which inflates the line rate by 25%, from 1,000–1,250 Mbit/s to ensure a DC balanced signal. The symbols are then sent using NRZ.

IEEE 802.3ab, which defines the widely used 1000BASE-T interface type, uses a different encoding scheme in order to keep the symbol rate as low as possible, allowing transmission over twisted pair.

Ethernet in the First Mile later added 1000BASE-LX10 and -BX10.

Q.2 Differentiate the working between pure ALOHA and slotted ALOHA.

Ans: **ALOHA :**

ALOHA is a medium access protocol that was originally designed for ground based radio broadcasting however it is applicable to any system in which uncoordinated users are competing for the use of a shared channel. Pure ALOHA and slotted ALOHA are the two versions of ALOHA.

Pure ALOHA uses a very simple idea that is to let users transmit whenever they have data to send. Pure ALOHA is featured with the feedback property that enables it to listen to the channel and finds out whether the frame was destroyed. Feedback is immediate in LANs but there is a delay of 270 msec in the satellite transmission. It requires acknowledgment if listening to the channel is not possible due to some reason. It can provide a channel utilization of 18 percent that is not appealing but it gives the advantage of transmitting any time.

Slotted ALOHA divides time into discrete intervals and each interval corresponds to a frame of data. It requires users to agree on slot boundaries. It does not allow a system to transmit any time. Instead the system has to wait for the beginning if the next slot.

Q.3 Write down distance vector algorithm. Explain path vector protocol.

Ans: **Distance Vector Algorithms**

Routing is the task of finding a path from a sender to a desired destination. In the IP "Catenet model" this reduces primarily to a matter of finding gateways between networks. As long as a message remains on a single network or subnet, any routing problems are solved by technology that is specific to the network. For

example, the Ethernet and the ARPANET each define a way in which any sender can talk to any specified destination within that one network. IP routing comes in primarily when messages must go from a sender on one such network to a destination on a different one. In that case, the message must pass through gateways connecting the networks. If the networks are not adjacent, the message may pass through several intervening networks, and the gateways connecting them. Once the message gets to a gateway that is on the same network as the destination, that network's own technology is used to get to the destination.

Throughout this section, the term "network" is used generically to cover a single broadcast network (e.g., an Ethernet), a point to point line, or the ARPANET. The critical point is that a network is treated as a single entity by IP. Either no routing is necessary (as with a point to point line), or that routing is done in a manner that is transparent to IP, allowing IP to treat the entire network as a single fully-connected system (as with an Ethernet or the ARPANET). Note that the term "network" is used in a somewhat different way in discussions of IP addressing. A single IP network number may be assigned to a collection of networks, with "subnet" addressing being used to describe the individual networks. In effect, we are using the term "network" here to refer to subnets in cases where subnet addressing is in use.

A number of different approaches for finding routes between networks are possible. One useful way of categorizing these approaches is on the basis of the type of information the gateways need to exchange in order to be able to find routes. Distance vector algorithms are based on the exchange of only a small amount of information. Each entity (gateway or host) that participates in the routing protocol is assumed to keep information about all of the destinations within the system. Generally, information about all entities connected to one network is summarized by a single entry, which describes the route to all destinations on that network. This summarization is possible because as far as IP is concerned, routing within a network is invisible. Each entry in this routing database includes the next gateway to which datagrams destined for the entity should be sent. In addition, it includes a "metric" measuring the total distance to the entity. Distance is a somewhat generalized concept, which may cover the time delay in getting messages to the entity, the dollar cost of sending messages to it, etc. Distance vector algorithms get their name from the fact that it is possible to compute optimal routes when the only information exchanged is the list of these distances. Furthermore, information is only exchanged among entities that are adjacent, that is, entities that share a common network.

Although routing is most commonly based on information about networks, it is sometimes necessary to keep track of the routes to individual hosts. The RIP protocol makes no formal distinction between networks and hosts. It simply describes exchange of information about destinations, which may be either networks or hosts. (Note however, that it is possible for an implementor to choose not to support host routes. See section 3.2.) In fact, the mathematical developments are most conveniently thought of in terms of routes from one host or gateway to another. When discussing the algorithm in abstract terms, it is best to think of a routing entry for a network as an abbreviation for routing entries for all of the entities connected to that network. This sort of abbreviation makes sense only because we think of networks as having no internal structure that is visible at the IP level. Thus, we will generally assign the same distance to every entity in a given network.

We said above that each entity keeps a routing database with one entry for every possible destination in the system. An actual implementation is likely to need to keep the following information about each destination:

address: in IP implementations of these algorithms, this will be the IP address of the host or network.
gateway: the first gateway along the route to the destination.
interface: the physical network which must be used to reach the first gateway.
metric: a number, indicating the distance to the destination.
timer: the amount of time since the entry was last updated.

In addition, various flags and other internal information will probably be included. This database is initialized with a description of the entities that are directly connected to the system. It is updated according to information received in messages from neighboring gateways.

The most important information exchanged by the hosts and gateways is that carried in update messages. Each entity that participates in the routing scheme sends update messages that describe the routing database as it currently exists in that entity. It is possible to maintain optimal routes for the entire system by using only information obtained from neighboring entities. The algorithm used for that will be described in the next section.

As we mentioned above, the purpose of routing is to find a way to get datagrams to their ultimate destinations. Distance vector algorithms are based on a table giving the best route to every destination in the system. Of

course, in order to define which route is best, we have to have some way of measuring goodness. This is referred to as the "metric".

In simple networks, it is common to use a metric that simply counts how many gateways a message must go through. In more complex networks, a metric is chosen to represent the total amount of delay that the message suffers, the cost of sending it, or some other quantity which may be minimized. The main requirement is that it must be possible to represent the metric as a sum of "costs" for individual hops.

Formally, if it is possible to get from entity i to entity j directly (i.e., without passing through another gateway between), then a cost, d(i,j), is associated with the hop between i and j. In the normal case where all entities on a given network are considered to be the same, d(i,j) is the same for all destinations on a given network, and represents the cost of using that network. To get the metric of a complete route, one just adds up the costs of the individual hops that make up the route. For the purposes of this memo, we assume that the costs are positive integers.

Let D(i,j) represent the metric of the best route from entity i to entity j. It should be defined for every pair of entities. d(i,j) represents the costs of the individual steps. Formally, let d(i,j) represent the cost of going directly from entity i to entity j. It is infinite if i and j are not immediate neighbors. (Note that d(i,i) is infinite. That is, we don't consider there to be a direct connection from a node to itself.) Since costs are additive, it is easy to show that the best metric must be described by

$$D(i,i) = 0, \quad \text{all } i$$
$$D(i,j) = \min_k [d(i,k) + D(k,j)], \quad \text{otherwise}$$

and that the best routes start by going from i to those neighbors k for which d(i,k) + D(k,j) has the minimum value. (These things can be shown by induction on the number of steps in the routes.) Note that we can limit the second equation to k's that are immediate neighbors of i. For the others, d(i,k) is infinite, so the term involving them can never be the minimum.

It turns out that one can compute the metric by a simple algorithm based on this. Entity i gets its neighbors k to send it their estimates of their distances to the destination j. When i gets the estimates from k, it adds d(i,k) to each of the numbers. This is simply the cost of traversing the network between i and k. Now and then i compares the values from all of its neighbors and picks the smallest.

A proof is given in [2] that this algorithm will converge to the correct estimates of D(i,j) in finite time in the absence of topology changes. The authors make very few assumptions about the order in which the entities send each other their information, or when the min is recomputed. Basically, entities just can't stop sending updates or recomputing metrics, and the networks can't delay messages forever. (Crash of a routing entity is a topology change.) Also, their proof does not make any assumptions about the initial estimates of D(i,j), except that they must be non-negative. The fact that these fairly weak assumptions are good enough is important. Because we don't have to make assumptions about when updates are sent, it is safe to run the algorithm asynchronously. That is, each entity can send updates according to its own clock. Updates can be dropped by the network, as long as they don't all get dropped. Because we don't have to make assumptions about the starting condition, the algorithm can handle changes. When the system changes, the routing algorithm starts moving to a new equilibrium, using the old one as its starting point. It is important that the algorithm will converge in finite time no matter what the starting point. Otherwise certain kinds of changes might lead to non-convergent behavior.

The statement of the algorithm given above (and the proof) assumes that each entity keeps copies of the estimates that come from each of its neighbors, and now and then does a min over all of the neighbors. In fact real implementations don't necessarily do that. They simply remember the best metric seen so far, and the identity of the neighbor that sent it. They replace this information whenever they see a better (smaller) metric. This allows them to compute the minimum incrementally, without having to store data from all of the neighbors.

There is one other difference between the algorithm as described in texts and those used in real protocols such as RIP: the description above would have each entity include an entry for itself, showing a distance of zero. In fact this is not generally done. Recall that all entities on a network are normally summarized by a single entry for the network. Consider the situation of a host or gateway G that is connected to network A. C represents the cost of using network A (usually a metric of one). (Recall that we are assuming that the internal structure of a

network is not visible to IP, and thus the cost of going between any two entities on it is the same.) In principle, G should get a message from every other entity H on network A, showing a cost of 0 to get from that entity to itself. G would then compute C + 0 as the distance to H. Rather than having G look at all of these identical messages, it simply starts out by making an entry for network A in its table, and assigning it a metric of C. This entry for network A should be thought of as summarizing the entries for all other entities on network A. The only entity on A that can't be summarized by that common entry is G itself, since the cost of going from G to G is 0, not C. But since we never need those 0 entries, we can safely get along with just the single entry for network A. Note one other implication of this strategy: because we don't need to use the 0 entries for anything, hosts that do not function as gateways don't need to send any update messages. Clearly hosts that don't function as gateways (i.e., hosts that are connected to only one network) can have no useful information to contribute other than their own entry D(i,i) = 0. As they have only the one interface, it is easy to see that a route to any other network through them will simply go in that interface and then come right back out it. Thus the cost of such a route will be greater than the best cost by at least C. Since we don't need the 0 entries, non-gateways need not participate in the routing protocol at all.

Let us summarize what a host or gateway G does. For each destination in the system, G will keep a current estimate of the metric for that destination (i.e., the total cost of getting to it) and the identity of the neighboring gateway on whose data that metric is based. If the destination is on a network that is directly connected to G, then G simply uses an entry that shows the cost of using the network, and the fact that no gateway is needed to get to the destination. It is easy to show that once the computation has converged to the correct metrics, the neighbor that is recorded by this technique is in fact the first gateway on the path to the destination. (If there are several equally good paths, it is the first gateway on one of them.) This combination of destination, metric, and gateway is typically referred to as a route to the destination with that metric, using that gateway.

The method so far only has a way to lower the metric, as the existing metric is kept until a smaller one shows up. It is possible that the initial estimate might be too low. Thus, there must be a way to increase the metric. It turns out to be sufficient to use the following rule: suppose the current route to a destination has metric D and uses gateway G. If a new set of information arrived from some source other than G, only update the route if the new metric is better than D. But if a new set of information arrives from G itself, always update D to the new value. It is easy to show that with this rule, the incremental update process produces the same routes as a calculation that remembers the latest information from all the neighbors and does an explicit minimum. (Note that the discussion so far assumes that the network configuration is static. It does not allow for the possibility that a system might fail.)

To summarize, here is the basic distance vector algorithm as it has been developed so far. (Note that this is not a statement of the RIP protocol. There are several refinements still to be added.) The following procedure is carried out by every entity that participates in the routing protocol. This must include all of the gateways in the system. Hosts that are not gateways may participate as well.

Keep a table with an entry for every possible destination in the system. The entry contains the distance D to the destination, and the first gateway G on the route to that network. Conceptually, there should be an entry for the entity itself, with metric 0, but this is not actually included.
Periodically, send a routing update to every neighbor. The update is a set of messages that contain all of the information from the routing table. It contains an entry for each destination, with the distance shown to that destination.
When a routing update arrives from a neighbor G', add the cost associated with the network that is shared with G'. (This should be the network over which the update arrived.) Call the resulting distance D'. Compare the resulting distances with the current routing table entries. If the new distance D' for N is smaller than the existing value D, adopt the new route. That is, change the table entry for N to have metric D' and gateway G'. If G' is the gateway from which the existing route came, i.e., G' = G, then use the new metric even if it is larger than the old one.

A path vector protocol is a computer network routing protocol which maintains the path information that gets updated dynamically. Updates which have looped through the network and returned to the same node are easily detected and discarded. This algorithm is sometimes used in Bellman–Ford routing algorithms to avoid "Count to Infinity" problems.

It is different from the distance vector routing and link state routing. Each entry in the routing table contains the destination network, the next router and the path to reach the destination.

Path Vector Messages in BGP: The autonomous system boundary routers (ASBR), which participate in path vector routing, advertise the reachability of networks. Each router that receives a path vector message must verify that the advertised path is according to its policy. If the messages comply with the policy, the ASBR modifies its routing table and the message before sending it to the next neighbor. In the modified message it sends its own AS number and replaces the next router entry with its own identification.

BGP is an example of a path vector protocol. In BGP the routing table maintains the autonomous systems that are traversed in order to reach the destination system. Exterior Gateway Protocol (EGP) does not use path vectors.

Path vector protocols are a class of distance vector protocol in contrast to link state protoco

Q.4 State the working principle of TCP segment header and UDP header.

Ans: **Transmission Control Protocol:**

Transmission Control Protocol, or TCP as it is commonly referred to, is a transport-layer protocol that runs on top of IP. TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

TCP was specifically designed to be a reliable end-to-end byte stream transmission protocol over an unreliable network. The IP layer does not provide any guarantees that datagrams will be delivered with any degree of reliability. Hence it is up to the upper-layer protocol to provide this reliability. The key functionality associated with TCP is basic data transfer.

Basic Data Transfer. From an application perspective, TCP transfers a contiguous stream of bytes through the network. The application does not have to bother with chopping the data into basic blocks or datagrams. TCP does this by grouping the bytes in TCP segments, which are passed to IP for transmission to the destination.

Reliability. TCP assigns a sequence number to each byte transmitted and expects a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data are retransmitted. Since the data are transmitted in blocks (TCP segments), only the sequence number of the first data byte in the segment is sent to the destination host.

Flow Control. The receiving TCP, when sending an ACK back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers. This is sent in the ACK in the form of the highest sequence number it can receive without problems.

Multiplexing. Multiplexing is achieved through the concept of ports. A port is a 16-bit number used by the host-to-host protocol to identify to which higher-level protocol or application process it must deliver incoming messages. Two types of ports exist: (1) Well-known: these ports belong to standard applications servers such as telnet, ftp, and http. The well-known ports are controlled and assigned by the Internet Assigned Numbers Authority (IANA). Well-known ports range from 1 to 1023. (2) Ephemeral: A client can negotiate the use of a port dynamically and such ports can be called ephemeral. These ports are maintained for the duration of the session and then released. Ephemeral ports range from 1024 to 65535. Multiple applications can use the ports as a means of multiplexing for communicating with other nodes.

Connections. The reliability and flow control mechanisms require that TCP initializes and maintains certain status information for each data stream. The combination of this status, including sockets, sequence numbers,

and window sizes, is called a logical connection. Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes.

TCP entities exchange data in the form of segments. A segment consists of a fixed 20-byte header and an optional part followed by zero or more data bytes.

The fields in the TCP header are desribed as follows:

Source Port and Destination Port: These fields identify the local endpoints of a connection. Each TCP entity decides how to allocate its own ports. A number of well-known ports are reserved for specific applications (e.g., FTP).

Sequence and Acknowledgment Number: Indicate the sequence number of the packet. The ACK number specifies the next byte expected, and not the last byte correctly received.

TCP Header Length: Indicates how many 32-bit words are contained in the TCP header. This is required because of the Options field, which is of variable length.

Reserved: For future use.

The six 1-bit flags are as follows:

URG— Set to 1 if the Urgent pointer is in use.

ACK— Set to 1 to indicate that the Acknowledgment number is valid.

PSH— Indicates PUSHed data. The receiver is requested to deliver the data to the application and not buffer it until a full buffer has been received.

RST— Used to reset a connection.

SYN— Used to establish connections.

FIN— Used to release a connection.

Window Size: This field tells how many bytes may be sent starting at the byte acknowledged. Flow control in TCP is handled using a variable-size sliding window.

Checksum: Provided for reliability. It checksums the header and the data (and the pseudoheader when applicable). While computing the checksum, the Checksum field itself is replaced with zeros.

Urgent Pointer: Used to indicate a byte offset from the current sequence number at which urgent data are to be found.

Options: This field was designed to provide a way to add extra facilities not covered by the regular header.

TCP has been the workhorse of the Internet, and a significant portion of Internet traffic today is carried via TCP. The reliability and congestion control aspects of TCP make it ideally suited for a large number of applications. TCP is formally defined in RFC 793. RFC 1122 provides some clarification and bug fixes, and a few extensions are defined in RFC 1323.

User Data Protocol

User Data Protocol (UDP) is a connectionless transport protocol. UDP is basically an application interface to IP. It adds no reliability, flow control, or error recovery to IP. It simply serves as a multiplexer/demultiplexer for sending and receiving datagrams, using ports to direct the datagrams. UDP is a light-weight protocol with very minimal overhead. The responsibility of recovering from errors, retransmission, etc., is up to the application. Applications that need to communicate need to identify a target is more specific than simply the IP address. UDP provides this function via the concept of ports. The format of the UDP datagram is shown in Figure 2-6.
Figure 2-6. UDP header.

The following is a description of the fields of the UDP header:

Source and Destination Port: The two ports serve the same function as in TCP; they identify the endpoints within the source and destination nodes.

UDP Length: This field includes the 8-byte UDP header and the data.

UDP Checksum: The checksum is computed over the UDP header, the IP header, and the data.

Although UDP does not implement flow control or reliable/ordered delivery, it does a little more work than simply to demultiplex messages to some application—it ensures the correctness of the message via the checksum. UDP uses the same cheksum algorithm as IP. UDP is described in RFC 768.

Q.5 What is IP addressing? Discuss different classes of IP Addressing.

Ans: **IP Addressing**:-

An Internet Protocol address (IP address) is a numerical label assigned to each device (e.g. computer, printer) participating in a computer network that uses the Internet Protocol for communication. An IP address serves two principal functions: host or network interface identification and location addressing. Its role has been characterized as follows: "A name indicates what we seek. An address indicates where it is. A route indicates how to get there."

The designers of computer network communication protocols defined an IP address as a 32-bit number and this system, known as Internet Protocol Version 4 (IPv4), is still in use today. However, due to the enormous growth of the Internet and the predicted depletion of available addresses, a new addressing system (IPv6), using 128 bits for the address, was developed in 1995, standardized in 1998, and is now being deployed world-wide.

Although IP addresses are stored as binary numbers, they are usually displayed in human-readable notations, such as 172.16.254.1 (for IPv4), and 2001:db8:0:1234:0:567:1:1 (for IPv6).

The Internet Assigned Numbers Authority (IANA) manages the IP address space allocations globally and cooperates with five regional Internet registries (RIRs) to allocate IP address blocks to local Internet registries (Internet service providers) and other entities.

IP addresses were originally organized into classes. The address class determined the potential size of the network.

The class of an address specified which of the bits were used to identify the network, the network ID, or which bits were used to identify the host ID, host computer. It also defined the total number of hosts subnets per network. There were five classes of IP addresses: classes A through E.

Classful addressing is no longer in common usage and has now been replaced with classless addressing. Any netmask can now be assigned to any IP address range.

The four octets that make up an IP address are conventionally represented by a, b, c, and d respectively. The following table shows how the octets are distributed in classes A, B, and C.

Class	IP Address	Network ID	Host ID
A	a.b.c.d	a	b.c.d
B	a.b.c.d	a.b	c.d
C	a.b.c.d	a.b.c	d

Class A: Class A addresses are specified to networks with large number of total hosts. Class A allows for 126 networks by using the first octet for the network ID. The first bit in this octet, is always set and fixed to zero. And next seven bits in the octet is all set to one, which then complete network ID. The 24 bits in the remaining octets represent the hosts ID, allowing 126 networks and approximately 17 million hosts per network. Class A network number values begin at 1 and end at 127.

Class B: Class B addresses are specified to medium to large sized of networks. Class B allows for 16,384 networks by using the first two octets for the network ID. The two bits in the first octet are always set and fixed to 1 0. The remaining 6 bits, together with the next octet, complete network ID. The 16 bits in the third and fourth octet represent host ID, allowing for approximately 65,000 hosts per network. Class B network number values begin at 128 and end at 191.

Class C: Class C addresses are used in small local area networks (LANs). Class C allows for approximately 2 million networks by using the first three octets for the network ID. In class C address three bits are always set and fixed to 1 1 0. And in the first three octets 21 bits complete the total network ID. The 8 bits of the last octet represent the host ID allowing for 254 hosts per one network. Class C network number values begin at 192 and end at 223.

Class D and E: Classes D and E are not allocated to hosts. Class D addresses are used for multicasting, and class E addresses are not available for general use: they are reserved for future purposes.

Q..6 Define Cryptography. Discuss two cryptographic techniques.

Ans: Cryptography Definition :

The practise and study of encryption and decryption - encoding data so that it can only be decoded by specific individuals. A system for encrypting and decrypting data is a cryptosystem. These usually involve an algorithm for combining the original data "plaintext" with one or more "keys" - numbers or strings of characters known only to the sender and/or recipient. The resulting output is known as "ciphertext".

The security of a cryptosystem usually depends on the secrecy of some of the keys rather than with the supposed secrecy of the algorithm. A strong cryptosystem has a large range of possible keys so that it is not possible to just try all possible keys a "brute force" approach. A strong cryptosystem will produce ciphertext which appears random to all standard statistical tests. A strong cryptosystem will resist all known previous methods for breaking codes "cryptanalysis".

Symmetric-key cryptography

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key (or, less commonly, in which their keys are different, but related in an easily computable way). This was the only kind of encryption publicly known until June 1976.[12]

One round (out of 8.5) of the patented IDEA cipher, used in some versions of PGP for high-speed encryption of, for instance, e-mail

The modern study of symmetric-key ciphers relates mainly to the study of block ciphers and stream ciphers and to their applications. A block cipher is, in a sense, a modern embodiment of Alberti's polyalphabetic cipher: block ciphers take as input a block of plaintext and a key, and output a block of ciphertext of the same size. Since messages are almost always longer than a single block, some method of knitting together successive blocks is required. Several have been developed, some with better security in one aspect or another than others. They are the modes of operation and must be carefully considered when using a block cipher in a cryptosystem.

The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs which have been designated cryptography standards by the US government (though DES's designation was finally withdrawn after the AES was adopted).[14] Despite its deprecation as an official standard, DES (especially its still-approved and much more secure triple-DES variant) remains quite popular; it is used across a wide range of applications, from ATM encryption[15] to e-mail privacy[16] and secure remote access.[17] Many other block ciphers have been designed and released, with considerable variation in quality. Many have been thoroughly broken; see Category:Block ciphers.[13][18]

Stream ciphers, in contrast to the 'block' type, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad. In a stream cipher, the output stream is created based on a hidden internal state which changes as the cipher operates. That internal state is initially set up using the secret key material. RC4 is a widely used stream cipher; see Category:Stream ciphers.[13] Block ciphers can be used as stream ciphers; see Block cipher modes of operation.

Cryptographic hash functions are a third type of cryptographic algorithm. They take a message of any length as input, and output a short, fixed length hash which can be used in (for example) a digital signature. For good hash functions, an attacker cannot find two messages that produce the same hash. MD4 is a long-used hash function which is now broken; MD5, a strengthened variant of MD4, is also widely used but broken in practice. The U.S. National Security Agency developed the Secure Hash Algorithm series of MD5-like hash functions: SHA-0 was a flawed algorithm that the agency withdrew; SHA-1 is widely deployed and more secure than MD5, but cryptanalysts have identified attacks against it; the SHA-2 family improves on SHA-1, but it isn't yet widely deployed, and the U.S. standards authority thought it "prudent" from a security perspective to develop a new standard to "significantly improve the robustness of NIST's overall hash algorithm toolkit."[19] Thus, a hash function design competition is underway and meant to select a new U.S. national standard, to be called SHA-3, by 2012.

Message authentication codes (MACs) are much like cryptographic hash functions, except that a secret key can be used to authenticate the hash value[13] upon receipt.
[edit]

Public-key cryptography

Symmetric-key cryptosystems use the same key for encryption and decryption of a message, though a message or group of messages may have a different key than others. A significant disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key, and perhaps each ciphertext exchanged as well. The number of keys required increases as the square of the number of network members, which very quickly requires complex key management schemes to keep them all straight and secret. The difficulty of securely establishing a secret key between two communicating parties, when a secure channel does not already exist between them, also presents a chicken-and-egg problem which is a considerable practical obstacle for cryptography users in the real world.

Whitfield Diffie and Martin Hellman, authors of the first published paper on public-key cryptography

In a groundbreaking 1976 paper, Whitfield Diffie and Martin Hellman proposed the notion of public-key (also, more generally, called asymmetric key) cryptography in which two different but mathematically related keys are used—a public key and a private key.[A public key system is so constructed that calculation of one key (the 'private key') is computationally infeasible from the other (the 'public key'), even though they are necessarily related. Instead, both keys are generated secretly, as an interrelated pair.[21] The historian David Kahn

described public-key cryptography as "the most revolutionary new concept in the field since polyalphabetic substitution emerged in the Renaissance".

In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret. The public key is typically used for encryption, while the private or secret key is used for decryption. Diffie and Hellman showed that public-key cryptography was possible by presenting the Diffie–Hellman key exchange protocol.

In 1978, Ronald Rivest, Adi Shamir, and Len Adleman invented RSA, another public-key system.

In 1997, it finally became publicly known that asymmetric key cryptography had been invented by James H. Ellis at GCHQ, a British intelligence organization, and that, in the early 1970s, both the Diffie–Hellman and RSA algorithms had been previously developed (by Malcolm J. Williamson and Clifford Cocks, respectively).

The Diffie–Hellman and RSA algorithms, in addition to being the first publicly known examples of high quality public-key algorithms, have been among the most widely used. Others include the Cramer–Shoup cryptosystem, ElGamal encryption, and various elliptic curve techniques. See Category:Asymmetric-key cryptosystems.

Padlock icon from the Firefox Web browser, meant to indicate a page has been sent in SSL or TLS-encrypted protected form. However, such an icon is not a guarantee of security; any subverted browser might mislead a user by displaying such an icon when a transmission is not actually being protected by SSL or TLS.

In addition to encryption, public-key cryptography can be used to implement digital signature schemes. A digital signature is reminiscent of an ordinary signature; they both have the characteristic that they are easy for a user to produce, but difficult for anyone else to forge. Digital signatures can also be permanently tied to the content of the message being signed; they cannot then be 'moved' from one document to another, for any attempt will be detectable. In digital signature schemes, there are two algorithms: one for signing, in which a secret key is used to process the message (or a hash of the message, or both), and one for verification, in which the matching public key is used with the message to check the validity of the signature. RSA and DSA are two of the most popular digital signature schemes. Digital signatures are central to the operation of public key infrastructures and many network security schemes (e.g., SSL/TLS, many VPNs, etc.).

Public-key algorithms are most often based on the computational complexity of "hard" problems, often from number theory. For example, the hardness of RSA is related to the integer factorization problem, while Diffie–Hellman and DSA are related to the discrete logarithm problem. More recently, elliptic curve cryptography has developed in which security is based on number theoretic problems involving elliptic curves. Because of the difficulty of the underlying problems, most public-key algorithms involve operations such as modular multiplication and exponentiation, which are much more computationally expensive than the techniques used in most block ciphers, especially with typical key sizes. As a result, public-key cryptosystems are commonly hybrid cryptosystems, in which a fast high-quality symmetric-key encryption algorithm is used for the message itself, while the relevant symmetric key is sent with the message, but encrypted using a public-key algorithm. Similarly, hybrid signature schemes are often used, in which a cryptographic hash function is computed

Assignment (Set-1)

Subject code: MI0036

Business intelligence & Tools

Q1. Define the term business intelligence tools? Briefly explain how the data from the one end gets transformed into information at the other end?

Ans:

In this section will be familiar with the definition of BI also the application involved in it. Business intelligence (BI) is a wide category of applications and technologies which gathers, stores, analysis, and provides access to data.

It helps enterprise users to make better business decisions. BI applications involve the activities of decision support systems, query and reporting, online analytical processing (OLAP), statistical analysis, forecasting, and data mining.

Business intelligence tools provide information on how trade is presently being conducted and what are the areas to be developed. Business intelligence tool are a kind of application software which is developed to report, evaluate, and present data. The tools generally read data that are stored previously however nor necessarily, in data warehouse. Following are some of the types of Business Intelligence Tools commonly used.

Multiplicity of business intelligence tools:

Multiplicity of business intelligence tools offers past, existing, and observations which can be expected in future which are of business operations. The main features of business intelligence methodologies include reporting, online systematic processing, analytics, withdrawal of information, industry performance administration, benchmarking, text mining, and projecting analytics.

It always encourages enhanced business decision-making processes. Therefore it is also called a decision support system. Even if he word business intelligence is frequently used as a synonym for competitive intelligence, BI uses technologies, processes, and functions to examine mainly internal, planned data and business methods as competitive intelligence, is done by assembling, evaluating and distribute information with or without the encouragement from technology and applications. It mainly concentrates on all-source information and data, which is mostly external and also internal to an organization, which helps in decision making.

Q2. What Do mean by data ware house? What are the major concepts and terminology used in the study of data warehouse?

Ans. In order to survive the market competition, an organization has to monitor the changes within the organization and outside the organization. An organization that cannot study the current trends within the organization and without its operations such as corporate relations will not be able to survive in the world today. This is where data warehousing and its applications comes into play.

Data warehousing: technology is the process by which the historical data of a company (also referred to as corporate memory) is created and utilized. A data warehouse is the database that contains data relevant to corporate information. This includes sales figures, market performances, accounts payables, and leave details

of employees. However, the data warehouse is not limited to the above mentioned data. The data available is useful in making decisions based on past performances of employees, expenses and experiences.

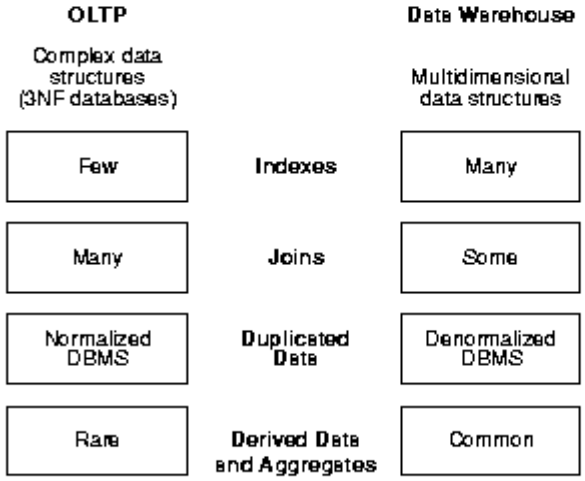
To utilize the data warehousing technology, companies can opt for online transaction processing (OLTP) or online analytical processing (OLAP). The uses of data warehousing are many. Let us consider a bank scenario to analyse the importance of data warehousing. In a bank, the account of several customers have to be maintained. It includes the balance, savings, and deposit details. The particulars of the bank employees and the information regarding their performance have to be maintained. Data warehousing technologies are used for the same.

Data warehousing transforms data to information and enables the organizations to analyse its operations and performances. This task is done by the staging and transformation of data from data sources. The data stores may be stored on disk or memory.

To extract, clean and load data from online transactions processing (OLTP) and the repositories of data, the data warehousing system uses backend tools. Data warehousing consists of the data storage are composed of the data warehouse, the data marts and the data store. It also provides tools like OLAP to organize, partition and summarise data in the data warehouse and data marts. Mining, querying and reporting on data requires front end tools.

Contrasting OLTP and Data Warehousing Environments

Illustrates key differences between an OLTP system and a data warehouse.



One major difference between the types of system is that data warehouses are not usually 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. 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 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. 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.

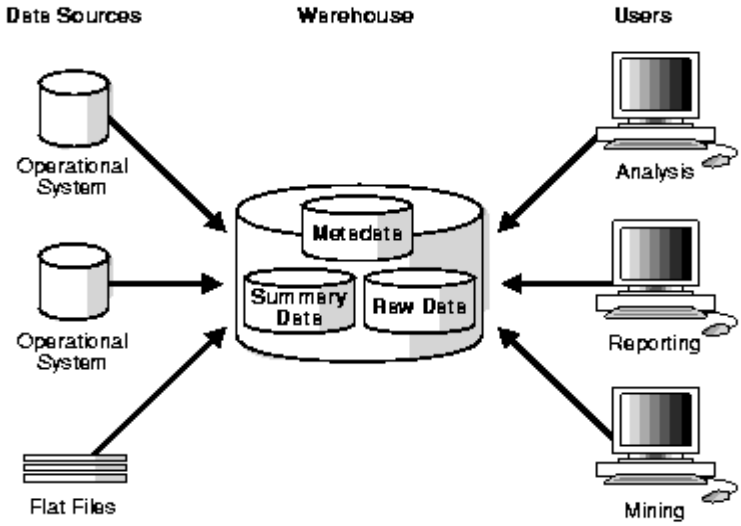
- *Schema design* :Data warehouses often use denormalized or partially denormalized schemas (such as a star schema) to optimize query performance. OLTP systems often use fully normalized schemas to optimize update/insert/delete performance, and to guarantee data consistency.
- *Typical operations :* 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. 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.

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)

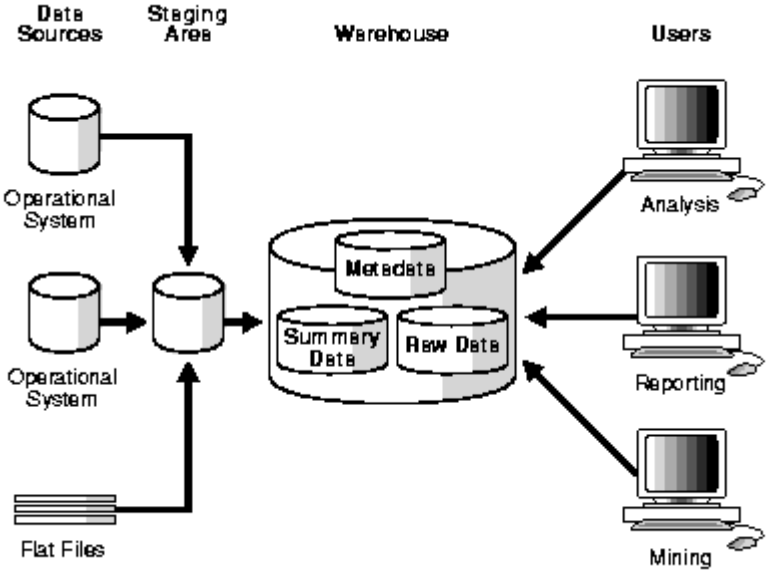
Data Warehouse Architecture (Basic) shows a simple architecture for a data warehouse. End users directly access data derived from several source systems through the data warehouse.



In , the metadata and raw data of a traditional OLTP system is present, as is an additional type of data, summary data. Summaries are very valuable in data warehouses because they pre-compute long operations in advance. For example, a typical data warehouse query is to retrieve something like August sales. A summary in Oracle is called a materialized view.

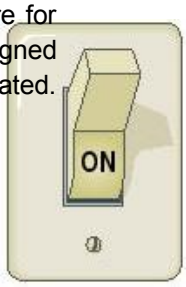
Data Warehouse Architecture (with a Staging Area)

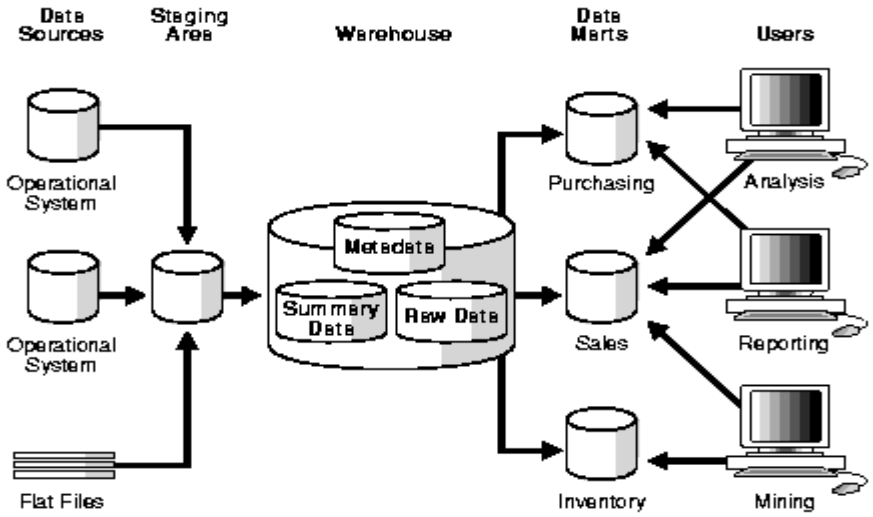
In , you need to clean and process your operational data before putting it into the warehouse. You can do this programmatically, although most data warehouses use a staging area instead. A staging area simplifies building summaries and general warehouse management. illustrates this typical architecture.



Data Warehouse Architecture (with a Staging Area and Data Marts)

Although the architecture in 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. 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.





Q3. What are the data modeling techniques used in data warehousing environment?

Ans: **Data Modelling Multi-fact Star Schema or Snowflake Schema**

Each of the dimension table consists of a single field primary key that has one-to-many relationship with a foreign key in the fact table. Let us look into some facts related to star and snowflake schema.

Model

The fact table consists of the main data and the other smaller dimension tables contain the description for each value in the dimensions. The dimension tables can be connected to the fact table. Fact table consist of a set of foreign keys that makes a complex primary key. Dimension tables consist of a primary key.

One of the reasons for using star schema is because it is simple. The queries are not complex as the joins and conditions involve a fact table and few single level dimension tables. In snowflake schema the queries are complex because of multiple levels of dimension tables.

Uses :

The stat and snowflake schema are used in dimensional data where the speed or retrieval is more important than the efficiency of data management. Therefore, data is not normalized much. The decision as to which schema should be used depends on two factors: the database platform, the query tool to be used. Star schema is relevant in environment where the queries are much simpler and the query tools expose the users to the fundamental table structures. Snowflake schema would be apt for environments with several queries with complex conditions where the user is detached from the fundamental table structures.

Data Normalization and storage:

The data in the database could be repeated. To reduce redundancy we use normalization. Commonly repeated data are moved into a new table. Therefore, the number of tables to be joined to execute a query increases. However, normalization reduces the space required for the storage of redundant data and other places where it has to update. The dimensional tables are smaller compared to fact table when storage is concerned.

What is Data Modeling?

Data modeling is the act of exploring data-oriented structures. Like other modeling artifacts data models can be used for a variety of purposes, from high-level conceptual models to physical data models. From the point of view of an object-oriented developer data modeling is conceptually similar to class modeling. With data modeling you identify entity types whereas with class modeling you identify classes. Data attributes are assigned to entity types just as you would assign attributes and operations to classes. There are associations between entities, similar to the associations between classes – relationships, inheritance, composition, and aggregation are all applicable concepts in data modeling.

Traditional data modeling is different from class modeling because it focuses solely on data – class models allow you to explore both the behavior and data aspects of your domain, with a data model you can only explore data issues. Because of this focus data modelers have a tendency to be much better at getting the data "right" than object modelers. However, some people will model database methods (stored procedures, stored functions, and triggers) when they are physical data modeling. It depends on the situation of course, but I personally think that this is a good idea and promote the concept in my UML data modeling profile (more on this later).

Although the focus of this article is data modeling, there are often alternatives to data-oriented artifacts (never forget Agile Modeling's Multiple Models principle). For example, when it comes to conceptual modeling ORM diagrams aren't your only option – In addition to LDMs it is quite common for people to create UML class diagrams and even Class Responsibility Collaborator (CRC) cards instead. In fact, my experience is that CRC cards are superior to ORM diagrams because it is very easy to get project stakeholders actively involved in the creation of the model. Instead of a traditional, analyst-led drawing session you can instead facilitate stakeholders through the creation of CRC cards.

How are Data Models Used in Practice?

Although methodology issues are covered later, we need to discuss how data models can be used in practice to better understand them. You are likely to see three basic styles of data model:

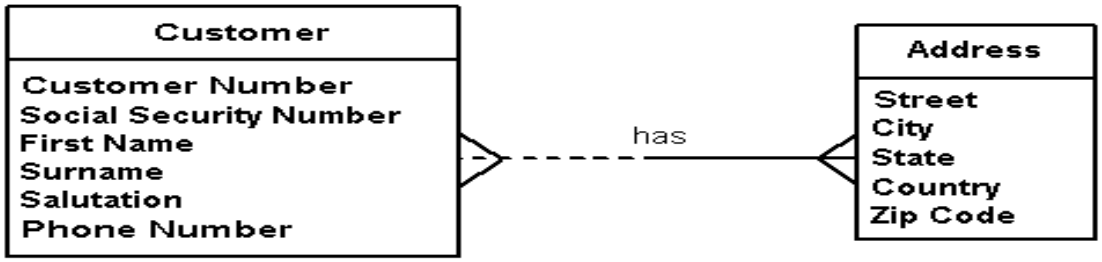
- **Conceptual data models**. These models, sometimes called domain models, are typically used to explore domain concepts with project stakeholders. On Agile teams high-level conceptual models are often created as part of your initial requirements envisioning efforts as they are used to explore the high-level static business structures and concepts. On traditional teams conceptual data models are often created as the precursor to LDMs or as alternatives to LDMs.

- **Logical data models (LDMs)**. LDMs are used to explore the domain concepts, and their relationships, of your problem domain. This could be done for the scope of a single project or for your entire enterprise. LDMs depict the logical entity types, typically referred to simply as entity types, the data attributes describing those entities, and the relationships between the entities. LDMs are rarely used on Agile projects although often are on traditional projects (where they rarely seem to add much value in practice).

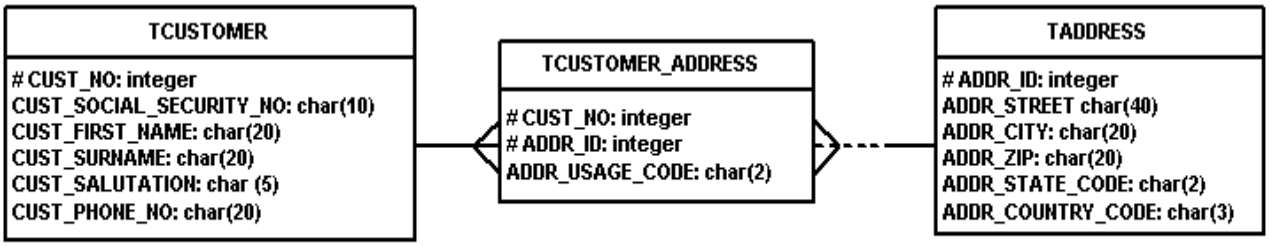
Physical data models (PDMs). PDMs are used to design the internal schema of a database, depicting the data tables, the data columns of those tables, and the relationships between the tables. PDMs often prove to be useful on both Agile and traditional projects and as a result the focus of this article is on physical modeling.

Although LDMs and PDMs sound very similar, and they in fact are, the level of detail that they model can be significantly different. This is because the goals for each diagram is different – you can use an LDM to explore domain concepts with your stakeholders and the PDM to define your database design. Figure 1 presents a simple LDM and Figure 2 a simple PDM, both modeling the concept of customers and addresses as well as the relationship between them. Both diagrams apply the Barker notation, summarized below. Notice how the PDM shows greater detail, including an associative table required to implement the association as well as the keys needed to maintain the relationships. More on these concepts later. PDMs should also reflect your organization's database naming standards, in this case an abbreviation of the entity name is appended to each column name and an abbreviation for "Number" was consistently introduced. A PDM should also indicate the data types for the columns, such as integer and char(5). Although Figure 2 does not show them, lookup tables (also called reference tables or description tables) for how the address is used as well as for states and countries are implied by the attributes *ADDR_USAGE_CODE*, *STATE_CODE*, and *COUNTRY_CODE*.

A simple logical data model.



A simple physical data model.



An important observation about Figures 1 and 2 is that I'm not slavishly following Barker's approach to naming relationships. For example, between *Customer* and *Address* there really should be two names "Each CUSTOMER may be located in one or more ADDRESSES" and "Each ADDRESS may be the site of one or more CUSTOMERS". Although these names explicitly define the relationship I personally think that they're visual noise that clutter the diagram. I prefer simple names such as "has" and then trust my readers to interpret the name in each direction. I'll only add more information where it's needed, in this case I think that it isn't. However, a significant advantage of describing the names the way that Barker suggests is that it's a good test to see if you actually understand the relationship – if you can't name it then you likely don't understand it.

Data models can be used effectively at both the enterprise level and on projects. Enterprise architects will often create one or more high-level LDMs that depict the data structures that support your enterprise, models typically referred to as enterprise data models or enterprise information models. An enterprise data model is one of several views that your organization's enterprise architects may choose to maintain and support – other views may explore your network/hardware infrastructure, your organization structure, your software infrastructure, and your business processes (to name a few). Enterprise data models provide information that

a project team can use both as a set of constraints as well as important insights into the structure of their system.

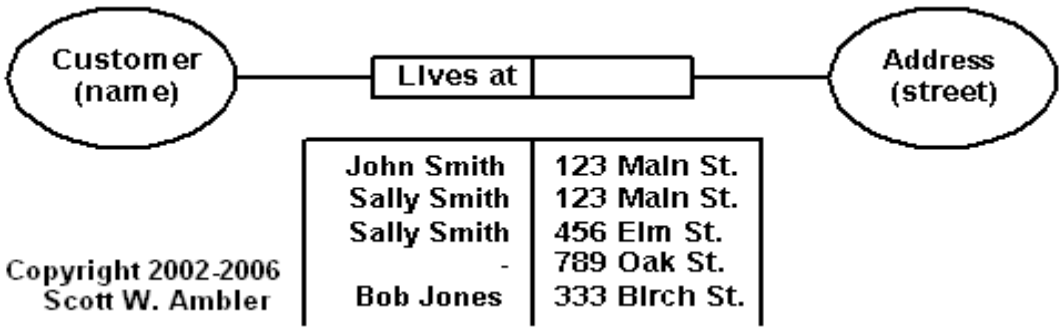
Project teams will typically create LDMs as a primary analysis artifact when their implementation environment is predominantly procedural in nature, for example they are using structured COBOL as an implementation language. LDMs are also a good choice when a project is data-oriented in nature, perhaps a data warehouse or reporting system is being developed (having said that, experience seems to show that usage-centered approaches appear to work even better). However LDMs are often a poor choice when a project team is using object-oriented or component-based technologies because the developers would rather work with UML diagrams or when the project is not data-oriented in nature. As Agile Modeling advises, apply the right artifact(s) for the job. Or, as your grandfather likely advised you, use the right tool for the job. It's important to note that traditional approaches to Master Data Management (MDM) will often motivate the creation and maintenance of detailed LDMs, an effort that is rarely justifiable in practice when you consider the total cost of ownership (TCO) when calculating the return on investment (ROI) of those sorts of efforts.

When a relational database is used for data storage project teams are best advised to create a PDMs to model its internal schema. My experience is that a PDM is often one of the critical design artifacts for business application development projects.

What About Conceptual Models?

Halpin (2001) points out that many data professionals prefer to create an Object-Role Model (ORM), an example is depicted in Figure 3, instead of an LDM for a conceptual model. The advantage is that the notation is very simple, something your project stakeholders can quickly grasp, although the disadvantage is that the models become large very quickly. ORMs enable you to first explore actual data examples instead of simply jumping to a potentially incorrect abstraction – for example Figure 3 examines the relationship between customers and addresses in detail.

A simple Object-Role Model.



It is seen that people will capture information in the best place that they know. As a result I typically discard ORMs after I'm finished with them. I sometimes user ORMs to explore the domain with project stakeholders but later replace them with a more traditional artifact such as an LDM, a class diagram, or even a PDM. As a **generalizing specialist**, someone with one or more specialties who also strives to gain general skills and knowledge, this is an easy decision for me to make; I know that this information that I've just "discarded" will be captured in another artifact – a model, the tests, or even the code – that I understand. A specialist who only understands a limited number of artifacts and therefore "hands-off" their work to other specialists doesn't have this as an option. Not only are they tempted to keep the artifacts that they create but also to invest even more time to enhance the artifacts. Generalizing specialists are more likely than specialists to **travel light**.

Common Data Modeling Notations

Figure presents a summary of the syntax of four common data modeling notations: Information Engineering (IE), Barker, IDEF1X, and the Unified Modeling Language (UML). This diagram isn't meant to be comprehensive, instead its goal is to provide a basic overview. Furthermore, for the sake of brevity I wasn't able to depict the highly-detailed approach to relationship naming that Barker suggests. Although I provide a brief description of each notation in Table 1 I highly suggest David Hay's paper A Comparison of Data Modeling Techniques as he goes into greater detail than I do.

Comparing the syntax of common data modeling notations.

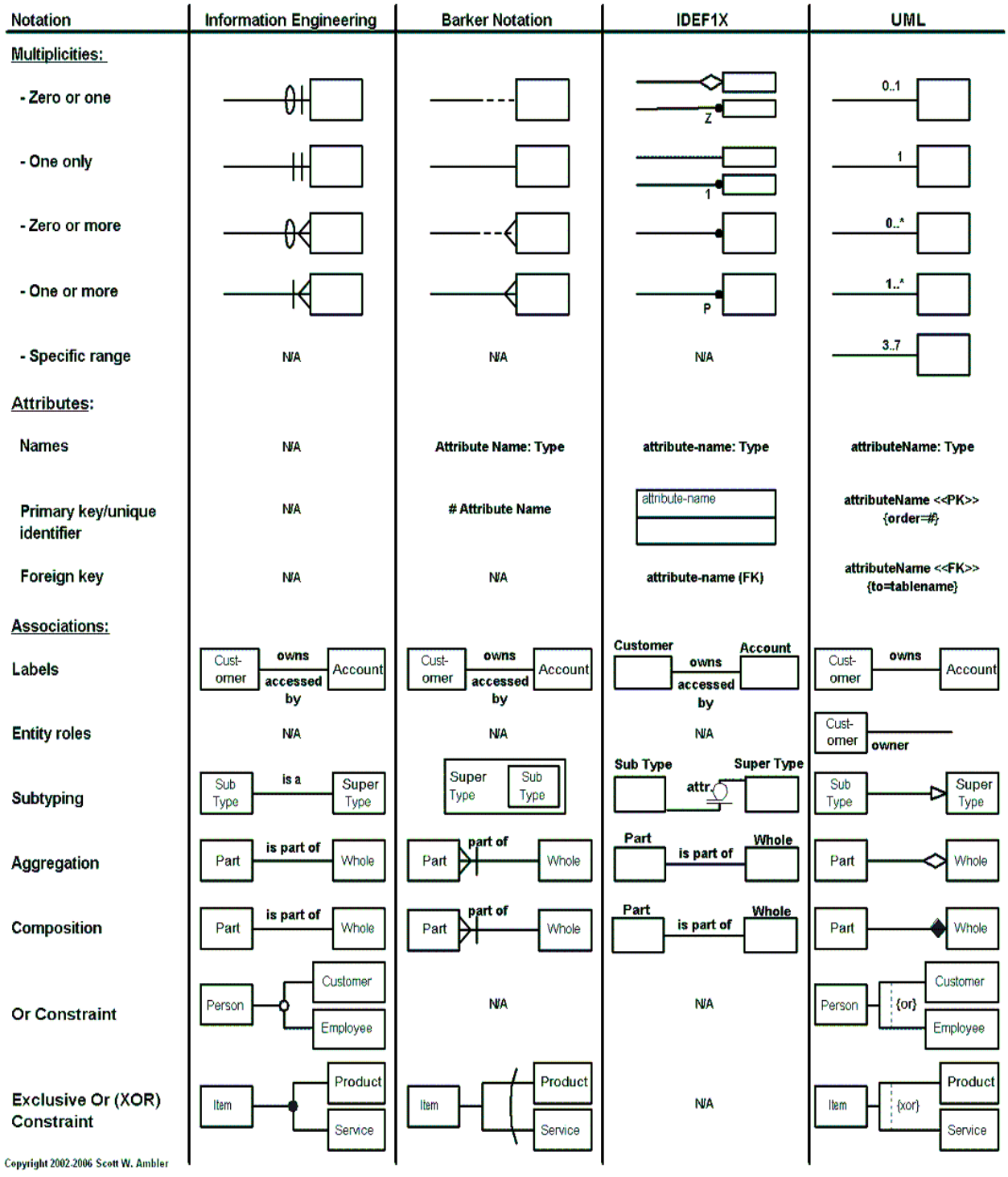


Table . Discussing common data modeling notations.

Notation	Comments
IE	The IE notation (Finkelstein 1989) is simple and easy to read, and is well suited for high-level logical and enterprise data modeling. The only drawback of this notation, arguably an advantage, is that it does not support the identification of attributes of an entity. The assumption is that the attributes will be modeled with another diagram or simply described in the supporting documentation.
Barker	The Barker notation is one of the more popular ones, it is supported by Oracle's toolset, and is well suited for all types of data models. It's approach to subtyping can become clunky with hierarchies that go several levels deep.
IDEF1X	This notation is overly complex. It was originally intended for physical modeling but has been misapplied for logical modeling as well. Although popular within some U.S. government agencies, particularly the Department of Defense (DoD), this notation has been all but abandoned by everyone else. Avoid it if you can.
UML	This is not an official data modeling notation (yet). Although several suggestions for a data modeling profile for the UML exist, none are complete and more importantly are not "official" UML yet. However, the Object Management Group (OMG) in December 2005 announced an RFP for data-oriented models.

How to Model Data

It is critical for an application developer to have a grasp of the fundamentals of data modeling so they can not only read data models but also work effectively with Agile DBAs who are responsible for the data-oriented aspects of your project. Your goal reading this section is not to learn how to become a data modeler, instead it is simply to gain an appreciation of what is involved.

The following tasks are performed in an iterative manner:

- Identify entity types
- Identify attributes
- Apply naming conventions
- Identify relationships
- Apply data model patterns
- Assign keys
- Normalize to reduce data redundancy
- Denormalize to improve performance

Identify Entity Types

An entity type, also simply called entity (not exactly accurate terminology, but very common in practice), is similar conceptually to object-orientation's concept of a class – an entity type represents a collection of similar objects. An entity type could represent a collection of people, places, things, events, or concepts. Examples of entities in an order entry system would include *Customer*, *Address*, *Order*, *Item*, and *Tax*. If you were class modeling you would expect to discover classes with the exact same names. However, the difference between a class and an entity type is that classes have both data and behavior whereas entity types just have data.

Ideally an entity should be normal, the data modeling world's version of cohesive. A normal entity depicts one concept, just like a cohesive class models one concept. For example, customer and order are clearly two different concepts; therefore it makes sense to model them as separate entities.

Identify Attributes

Each entity type will have one or more data attributes. For example, in Figure 1 you saw that the *Customer* entity has attributes such as *First Name* and *Surname* and in Figure 2 that the *TCUSTOMER* table had corresponding data columns *CUST_FIRST_NAME* and *CUST_SURNAME* (a column is the implementation of a data attribute within a relational database).

Attributes should also be cohesive from the point of view of your domain, something that is often a judgment call. – in Figure 1 we decided that we wanted to model the fact that people had both first and last names instead of just a name (e.g. "Scott" and "Ambler" vs. "Scott Ambler") whereas we did not distinguish between the sections of an American zip code (e.g. 90210-1234-5678). Getting the level of detail right can have a significant impact on your development and maintenance efforts. Refactoring a single data column into several columns can be difficult, database refactoring is described in detail in Database Refactoring, although over-specifying an attribute (e.g. having three attributes for zip code when you only needed one) can result in overbuilding your system and hence you incur greater development and maintenance costs than you actually needed.

Apply Data Naming Conventions

Your organization should have standards and guidelines applicable to data modeling, something you should be able to obtain from your enterprise administrators (if they don't exist you should lobby to have some put in place). These guidelines should include naming conventions for both logical and physical modeling, the logical naming conventions should be focused on human readability whereas the physical naming conventions will reflect technical considerations. You can clearly see that different naming conventions were applied in Figures.

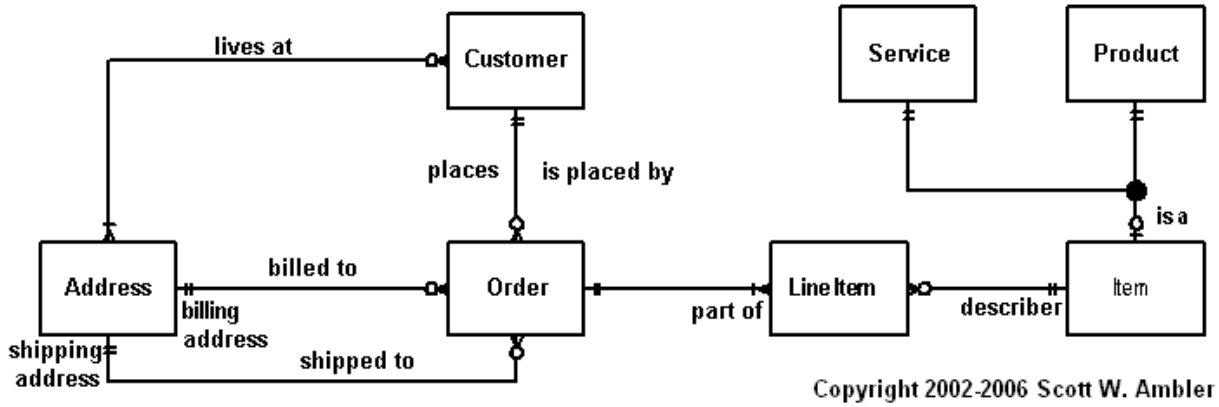
As you saw in Introduction to Agile Modeling, AM includes the Apply Modeling Standards practice. The basic idea is that developers should agree to and follow a common set of modeling standards on a software project. Just like there is value in following common coding conventions, clean code that follows your chosen coding guidelines is easier to understand and evolve than code that doesn't, there is similar value in following common modeling conventions.

Identify Relationships

In the real world entities have relationships with other entities. For example, customers PLACE orders, customers LIVE AT addresses, and line items ARE PART OF orders. Place, live at, and are part of are all terms that define relationships between entities. The relationships between entities are conceptually identical to the relationships (associations) between objects.

Figure depicts a partial LDM for an online ordering system. The first thing to notice is the various styles applied to relationship names and roles – different relationships require different approaches. For example the relationship between *Customer* and *Order* has two names, *places* and *is placed by*, whereas the relationship between *Customer* and *Address* has one. In this example having a second name on the relationship, the idea being that you want to specify how to read the relationship in each direction, is redundant – you're better off to find a clear wording for a single relationship name, decreasing the clutter on your diagram. Similarly you will often find that by specifying the roles that an entity plays in a relationship will often negate the need to give the relationship a name (although some CASE tools may inadvertently force you to do this). For example the role of *billing address* and the label *billed to* are clearly redundant, you really only need one. For example the role *part of* that *Line Item* has in its relationship with *Order* is sufficiently obvious without a relationship name.

A logical data model (Information Engineering notation).

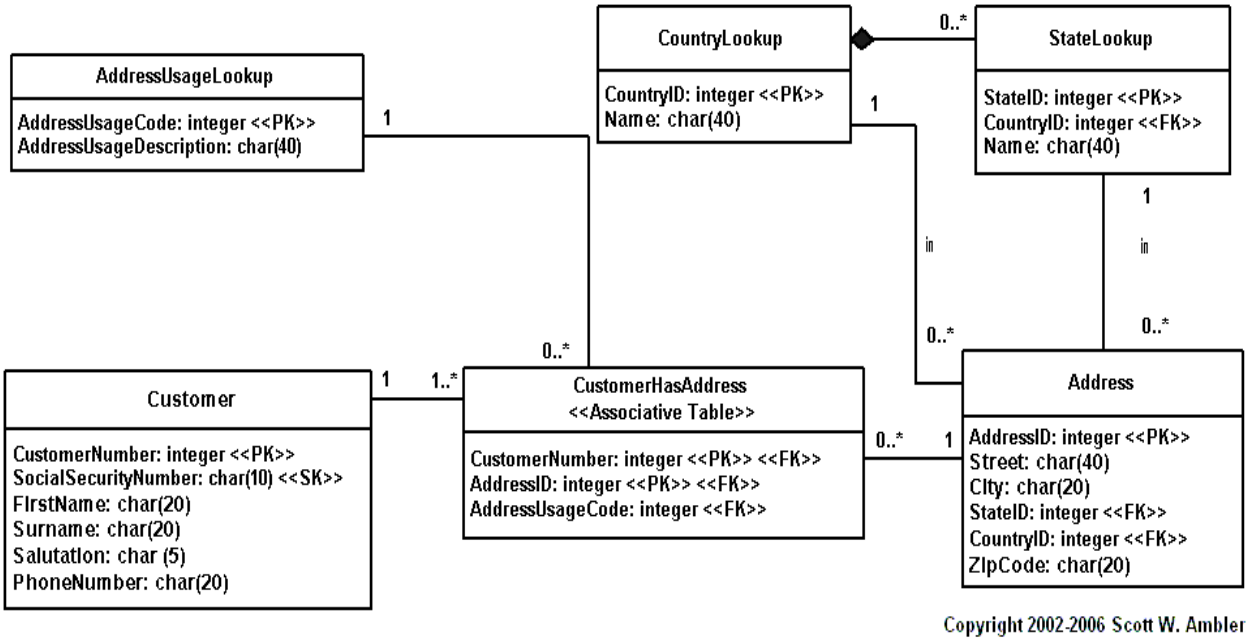


It is also need to identify the cardinality and optionality of a relationship (the UML combines the concepts of optionality and cardinality into the single concept of multiplicity). Cardinality represents the concept of "how many" whereas optionality represents the concept of "whether you must have something." For example, it is not enough to know that customers place orders. How many orders can a customer place? None, one, or several? Furthermore, relationships are two-way streets: not only do customers place orders, but orders are placed by customers. This leads to questions like: how many customers can be enrolled in any given order and is it possible to have an order with no customer involved? Figure 5 shows that customers place one or more orders and that any given order is placed by one customer and one customer only. It also shows that a customer lives at one or more addresses and that any given address has zero or more customers living at it.
Although the UML distinguishes between different types of relationships – associations, inheritance, aggregation, composition, and dependency – data modelers often aren't as concerned with this issue as much as object modelers are. Subtyping, one application of inheritance, is often found in data models, an example of which is the *is a* relationship between *Item* and it's two "sub entities" *Service* and *Product*. Aggregation and composition are much less common and typically must be implied from the data model, as you see with the *part of* role that *Line Item* takes with *Order*. UML dependencies are typically a software construct and therefore wouldn't appear on a data model, unless of course it was a very highly detailed physical model that showed how views, triggers, or stored procedures depended on other aspects of the database schema.

Assign Keys

There are two fundamental strategies for assigning keys to tables. First, you could assign a natural key which is one or more existing data attributes that are unique to the business concept. The *Customer* table of Figure 6 there was two candidate keys, in this case *CustomerNumber* and *SocialSecurityNumber*. Second, you could introduce a new column, called a surrogate key, which is a key that has no business meaning. An example of which is the *AddressID* column of the *Address* table in Figure 6. Addresses don't have an "easy" natural key because you would need to use all of the columns of the *Address* table to form a key for itself (you might be able to get away with just the combination of *Street* and *ZipCode* depending on your problem domain), therefore introducing a surrogate key is a much better option in this case.

Figure . Customer and Address revisited (UML notation).



Let's consider Figure in more detail. Figure presents an alternative design to that presented in Figure, a different naming convention was adopted and the model itself is more extensive. In Figure 6 the *Customer* table has the *CustomerNumber* column as its primary key and *SocialSecurityNumber* as an alternate key. This indicates that the preferred way to access customer information is through the value of a person's customer number although your software can get at the same information if it has the person's social security number. The *CustomerHasAddress* table has a composite primary key, the combination of *CustomerNumber* and *AddressID*. A foreign key is one or more attributes in an entity type that represents a key, either primary or secondary, in another entity type. Foreign keys are used to maintain relationships between rows. For example, the relationships between rows in the *CustomerHasAddress* table and the *Customer* table is maintained by the *CustomerNumber* column within the *CustomerHasAddress* table. The interesting thing about the *CustomerNumber* column is the fact that it is part of the primary key for *CustomerHasAddress* as well as the foreign key to the *Customer* table. Similarly, the *AddressID* column is part of the primary key of *CustomerHasAddress* as well as a foreign key to the *Address* table to maintain the relationship with rows of *Address*.

Although the "natural vs. surrogate" debate is one of the great religious issues within the data community, the fact is that neither strategy is perfect and you'll discover that in practice (as we see in Figure 6) sometimes it makes sense to use natural keys and sometimes it makes sense to use surrogate keys. In Choosing a Primary Key: Natural or Surrogate? I describe the relevant issues in detail.

Normalize to Reduce Data Redundancy

Data normalization is a process in which data attributes within a data model are organized to increase the cohesion of entity types. In other words, the goal of data normalization is to reduce and even eliminate data redundancy, an important consideration for application developers because it is incredibly difficult to stores objects in a relational database that maintains the same information in several places. Table 2 summarizes the three most common normalization rules describing how to put entity types into a series of increasing levels of normalization. Higher levels of data normalization (Date 2000) are beyond the scope of this book. With respect to terminology, a data schema is considered to be at the level of normalization of its least normalized entity type. For example, if all of your entity types are at second normal form (2NF) or higher then we say that your data schema is at 2NF.

Table 2. Data Normalization Rules.

Level	Rule
First normal form (1NF)	An entity type is in 1NF when it contains no repeating groups of data.
Second normal form (2NF)	An entity type is in 2NF when it is in 1NF and when all of its non-key attributes are fully dependent on its primary key.
Third normal form (3NF)	An entity type is in 3NF when it is in 2NF and when all of its attributes are directly dependent on the primary key.

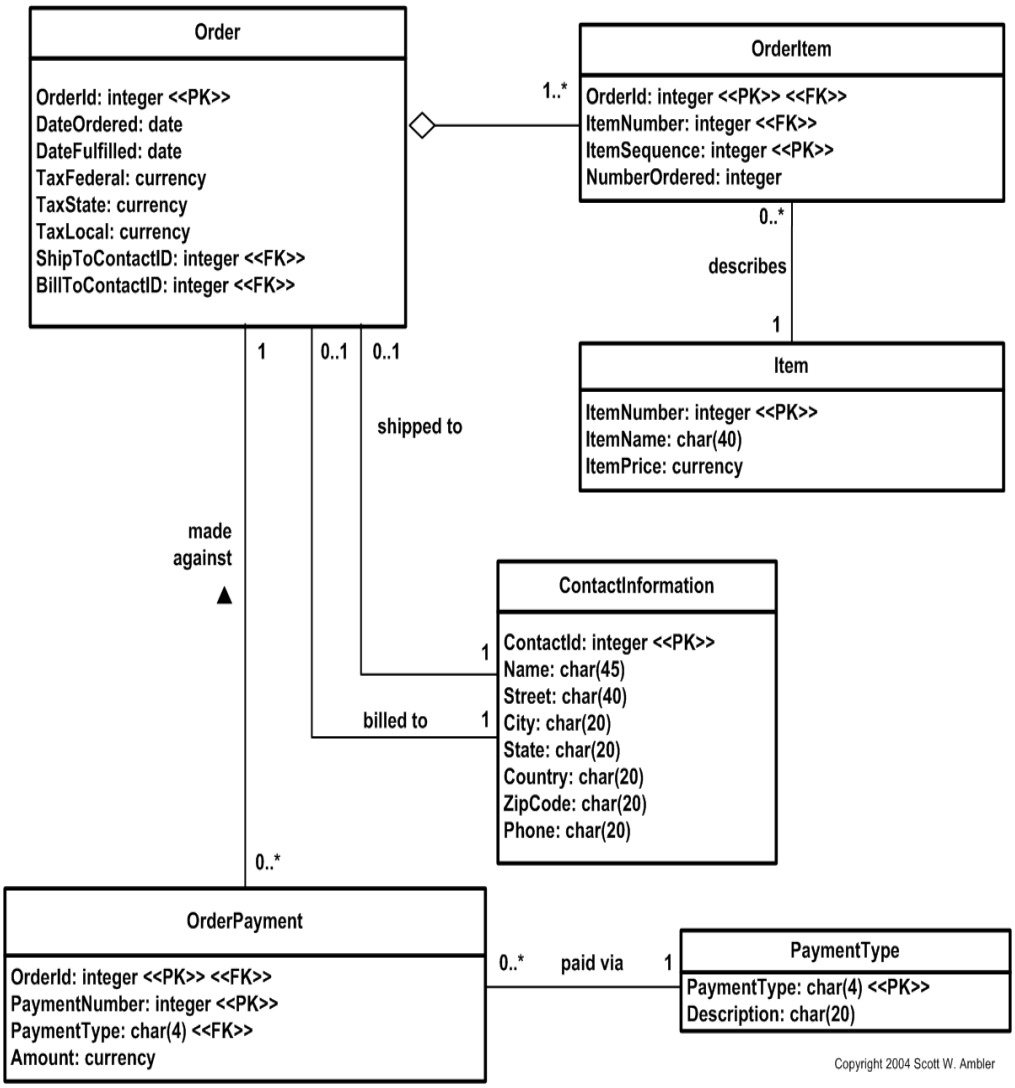
 depicts a database schema in ONF whereas Figure 8 depicts a normalized schema in 3NF. Read the Introduction to Data Normalization essay for details.

Why data normalization? The advantage of having a highly normalized data schema is that information is stored in one place and one place only, reducing the possibility of inconsistent data. Furthermore, highly-normalized data schemas in general are closer conceptually to object-oriented schemas because the object-oriented goals of promoting high cohesion and loose coupling between classes results in similar solutions (at least from a data point of view). This generally makes it easier to map your objects to your data schema. Unfortunately, normalization usually comes at a performance cost. With the data schema of Figure 7 all the data for a single order is stored in one row (assuming orders of up to nine order items), making it very easy to access. With the data schema of Figure 7 you could quickly determine the total amount of an order by reading

the single row from the *Order0NF* table. To do so with the data schema of Figure 8 you would need to read data from a row in the *Order* table, data from all the rows from the *OrderItem* table for that order and data from the corresponding rows in the *Item* table for each order item. For this query, the data schema of Figure 7 very likely provides better performance.

An Initial Data Schema for Order (UML Notation).

Normalized schema in 3NF (UML Notation).

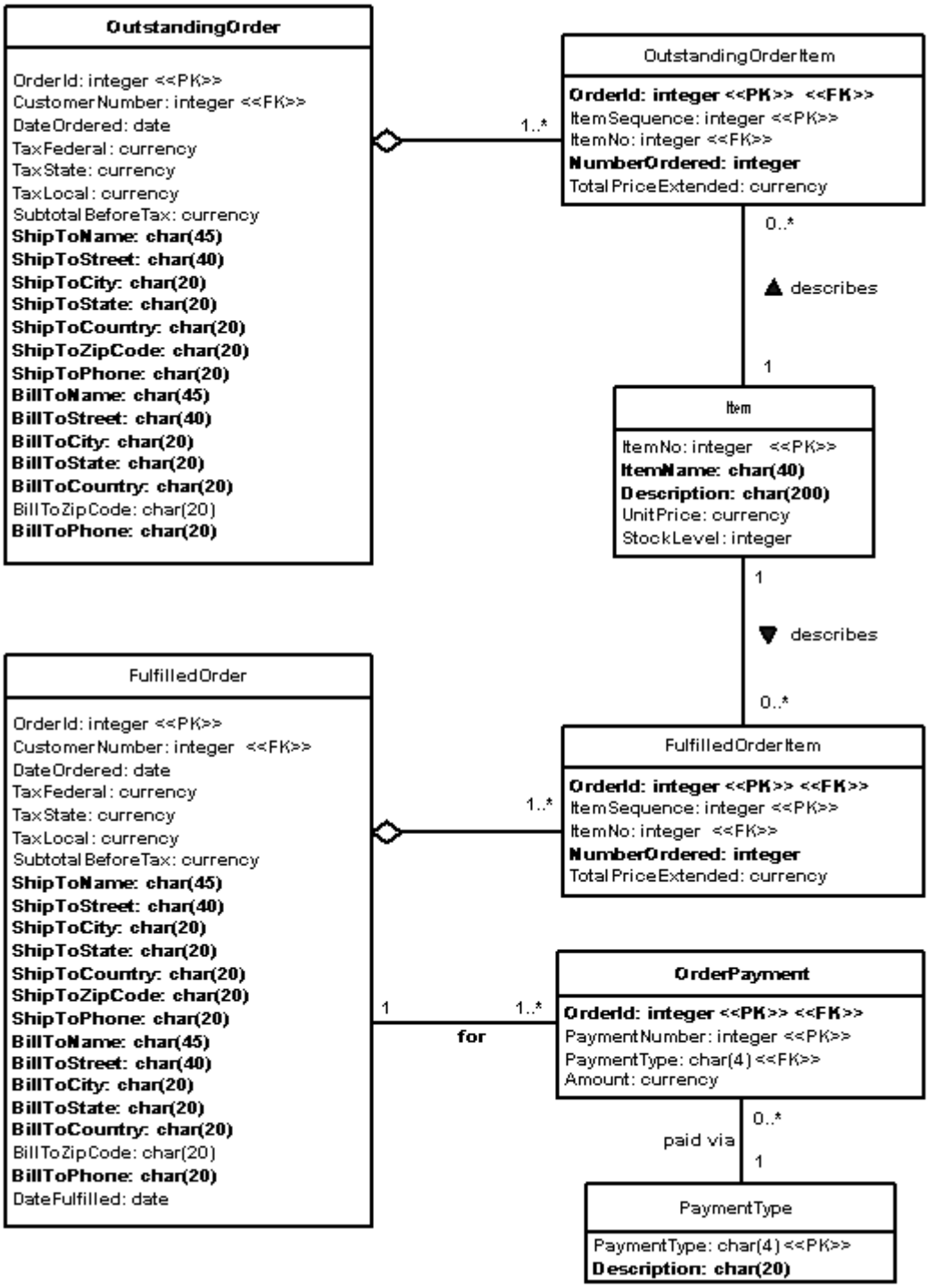


In class modeling, there is a similar concept called Class Normalization although that is beyond the scope of this article.

Denormalize to Improve Performance

Normalized data schemas, when put into production, often suffer from performance problems. This makes sense – the rules of data normalization focus on reducing data redundancy, not on improving performance of data access. An important part of data modeling is to denormalize portions of your data schema to improve database access times. For example, the data model of Figure 9 looks nothing like the normalized schema of Figure 8. To understand why the differences between the schemas exist you must consider the performance needs of the application. The primary goal of this system is to process new orders from online customers as quickly as possible. To do this customers need to be able to search for items and add them to their order quickly, remove items from their order if need be, then have their final order totaled and recorded quickly. The secondary goal of the system is to the process, ship, and bills the orders afterwards.

A Denormalized Order Data Schema (UML notation).



To denormalize the data schema the following decisions were made:

1. To support quick searching of item information the *Item* table was left alone.

2. To support the addition and removal of order items to an order the concept of an *OrderItem* table was kept, albeit split in two to support outstanding orders and fulfilled orders. New order items can easily be inserted into the *OutstandingOrderItem* table, or removed from it, as needed.

3. To support order processing the *Order* and *OrderItem* tables were reworked into pairs to handle outstanding and fulfilled orders respectively. Basic order information is first stored in the *OutstandingOrder* and *OutstandingOrderItem* tables and then when the order has been shipped and paid for the data is then removed from those tables and copied into the *FulfilledOrder* and *FulfilledOrderItem* tables respectively. Data access time to the two tables for outstanding orders is reduced because only the active orders are being stored there. On average an order may be outstanding for a couple of days, whereas for financial reporting reasons may be stored in the fulfilled order tables for several years until archived. There is a performance penalty under this scheme because of the need to delete outstanding orders and then resave them as fulfilled orders, clearly something that would need to be processed as a transaction.

4. The contact information for the person(s) the order is being shipped and billed to was also denormalized back into the *Order* table, reducing the time it takes to write an order to the database because there is now one write instead of two or three. The retrieval and deletion times for that data would also be similarly improved.

Note that if your initial, normalized data design meets the performance needs of your application then it is fine as is. Denormalization should be resorted to only when performance testing shows that you have a problem with your objects and subsequent profiling reveals that you need to improve database access time. As my grandfather said, if it ain't broke don't fix it.

Evolutionary/Agile Data Modeling

Evolutionary data modeling is data modeling performed in an iterative and incremental manner. The article Evolutionary Development explores evolutionary software development in greater detail. Agile data modeling is evolutionary data modeling done in a collaborative manner. The article Agile Data Modeling: From Domain Modeling to Physical Modeling works through a case study which shows how to take an agile approach to data modeling.

Although you wouldn't think it, data modeling can be one of the most challenging tasks that an Agile DBA can be involved with on an agile software development project. Your approach to data modeling will often be at the center of any controversy between the agile software developers and the traditional data professionals within your organization. Agile software developers will lean towards an evolutionary approach where data modeling is just one of many activities whereas traditional data professionals will often lean towards a big design up front (BDUF) approach where data models are the primary artifacts, if not THE artifacts. This problem results from a combination of the cultural impedance mismatch, a misguided need to enforce the "one truth", and "normal" political maneuvering within your organization. As a result Agile DBAs often find that navigating the political waters is an important part of their data modeling efforts.

Q4. Discuss the categories in which data is divided before structuring it into data ware house?

Ans. **Data Warehouse Testing Categories**

Categories of Data Warehouse testing includes different stages of the process. The testing is done on individual and end to end basis.

Good part of the testing of data warehouse testing can be linked to 'Data Warehouse Quality Assurance'. Data Warehouse Testing will include the following chapters:

Extraction Testing

This testing checks the following:

- Data is able to extract the required fields.
- The Extraction logic for each source system is working
- Extraction scripts are granted security access to the source systems.
- Updating of extract audit log and time stamping is happening.
- Source to Extraction destination is working in terms of completeness and accuracy.
- Extraction is getting completed with in the expected window.

Transformation Testing

- Transaction scripts are transforming the data as per the expected logic.
- The one time Transformation for historical snap-shots are working.
- Detailed and aggregated data sets are created and are matching.
- Transaction Audit Log and time stamping is happening.
- There is no pilferage of data during Transformation process.
- Transformation is getting completed with in the given window

Loading Testing

- There is no pilferage during the Loading process.
- Any Transformations during Loading process is working.
- Data sets in staging to Loading destination is working.
- One time historical snap-shots are working.
- Both incremental and total refresh are working.
- Loading is happening with in the expected window.

End User Browsing and OLAP Testing

- The Business views and dashboard are displaying the data as expected.
- The scheduled reports are accurate and complete.
- The scheduled reports and other batch operations like view refresh etc. is happening in the expected window.
- 'Analysis Functions' and 'Data Analysis' are working.
- There is no pilferage of data between the source systems and the views.

Ad-hoc Query Testing

- Ad-hoc queries creation is as per the expected functionalities.
- Ad-hoc queries output response time is as expected.

Down Stream Flow Testing

- Data is extracted from the data warehouse and updated in the down-stream systems/data marts.
- There is no pilferage.

One Time Population testing

- The one time ETL for the production data is working
- The production reports and the data warehouse reports are matching
- T he time taken for one time processing will be manageable within the conversion weekend.

End-to-End Integrated Testing

- End to end data flow from the source system to the down stream system is complete and accurate.

Stress and volume Testing

This part of testing will involve, placing maximum volume OR failure points to check the robustness and capacity of the system. The level of stress testing depends upon the configuration of the test environment and the level of capacity planning done. Here are some examples from the ideal world:

- Server shutdown during batch process.
- Extraction, Transformation and Loading with two to three times of maximum possible imagined data (for which the capacity is planned)
- Having 2 to 3 times more users placing large numbers of ad-hoc queries.
- Running large number of scheduled reports.

Parallel Testing
Parallel testing is done where the Data Warehouse is run on the production data as it would have done in real life and its outputs are compared with the existing set of reports to ensure that they are in synch OR have the explained mismatches.

Q5. Discuss the purpose of executive information system in an organization?

Ans: Not a piece of hardware or software, but an infrastructure that supplies to a firm's executives the up-to-the-minute operational data, gathered and sifted from various databases. The typical information mix presented to the executive may include financial information, work in process, inventory figures, sales figures, market trends, industry statistics, and market price of the firm's shares. It may even suggest what needs to be done, but differs from a decision support system (DSS) in that it is targeted at executives and not managers.

An **Executive Information System** (EIS) is a computer-based system intended to facilitate and support the information and decision making needs of senior executives by providing easy access to both internal and external information relevant to meeting the strategic goals of the organization. It is commonly considered as a specialized form of Decision Support System (DSS).

The emphasis of EIS in on graphical displays and easy-to-use user interfaces. They offer strong reporting and drill-down capabilities. In general, EIS are enterprise-wide DSS that help top-level executives analyze, compare, and highlight trends in important variables so that they can monitor performance and identify opportunities and problems. EIS and data warehousing technologies are converging in the marketplace.

Executive Information System (EC-EIS)

Purpose
An **executive information system (EIS)** provides information about all the factors that influence the business activities of a company. It combines relevant data from external and internal sources and provides the user with important current data which can be analyzed quickly.
The EC-Executive Information System (EC-EIS) is a system which is used to collect and evaluate information from different areas of a business and its environment. Among others, sources of this information can be the Financial Information System (meaning external accounting and cost accounting), the Human Resources Information System and the Logistics Information System.
The information provided serves both management and the employees in Accounting.

Implementation Considerations
EC-EIS is the information system for upper management. It is generally suitable for the collection and evaluation of data from different functional information systems in one uniform view.

Integration
The Executive Information System is based on the same data basis, and has the same data collection facilities as Business Planning . In EC-EIS you can report on the

data planned in EC-BP.

Features
When customizing your Executive Information System you set up an individual EIS database for your business and have this supplied with data from various sub-information systems (Financial Information System, Human Resources Information System, Logistics Information System, cost accounting, etc.) or with external data. Since this data is structured heterogeneously, you can structure the data basis into separate EIS data areas for different business purposes. These data areas are called aspects. You can define various aspects for your enterprise containing, for example, information on the financial situation, logistics, human resources, the market situation, and stock prices. For each aspect you can create reports to evaluate the data. You can either carry out your own basic evaluations in the EIS presentation (reporting) system or analyze the data using certain report groups created specifically for your requirements. To access the EIS presentation functions, choose *Information systems →EIS.*

In this documentation the application functions are described in detail and the customizing functions in brief. It is intended for the EC-EIS user but also those responsible for managing the system. To access the EC-EIS application menu, choose *Accounting →Enterprise control. →Executive InfoSystem.*

To call up the presentation functions from the application menu, choose *Environment → Executive menu.*
Necessary preliminary tasks and settings are carried out in Customizing. You can find a detailed description of the customizing functions in the implementation guidelines.

Setting Up the Data Basis
An **aspect** consists of **characteristics** and **key figures**. Characteristics are classification terms such as division, region, department, or company. A combination of characteristic values for certain characteristics (such as Division: Pharmaceuticals, Region: Northwest) is called an evaluation object. Key figures are numerical values such as revenue, fixed costs, variable costs, number of employees and quantity produced. They also form part of the structure of the aspect.

The key figure data is stored according to the characteristics in an aspect. Besides these key figures stored in the database, you can also define calculated key figures in EC-EIS and EC-BP. Calculated key figures are calculated with a formula and the basic key figures of the aspect (for example: CM1 per employee = (sales - sales deductions – variable costs) / Number of employees).

Determining characteristics and key figures when setting up the system provides the framework for the possible evaluations. You make these settings in customizing. When the structure of the aspect and the data basis have been defined in Customizing, you can evaluate data.

Presentation of the Data
Drilldown reporting and the report portfolio help you to evaluate and present your data. You can evaluate EC-EIS data interactively using drilldown reporting. You a make a selection of the characteristics and key figures from the data basis. You can analyze many types of variance (plan/actual comparisons, time comparisons, object comparisons). Drilldown reporting contains easy-to-use functions for navigating through the dataset. In addition, there are a variety of functions for interactively processing a report (selection conditions, exceptions, sort, top n and so on). You can also access SAPgraphics and SAPmail and print using Microsoft Word for Windows and Microsoft Excel.

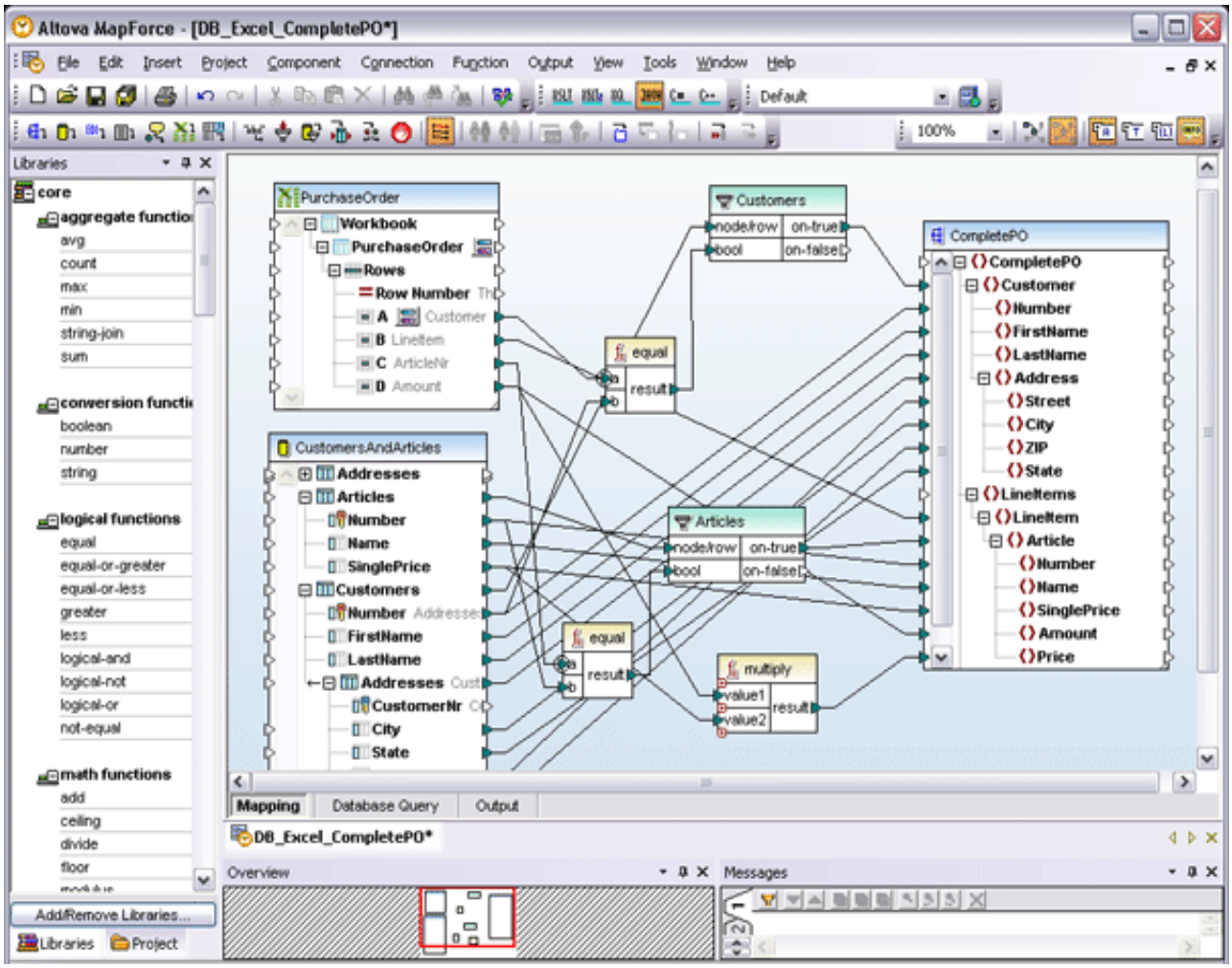
Drilldown reporting, with its numerous functions, is aimed at trained users, especially financial controllers and managers. By using the various function levels appropriately, other users can execute reports without needing extensive training. The reports created in drilldown reporting can be combined for specific user groups and stored in the graphical report portfolio. The report portfolio is aimed at users with basic knowledge of the system who wish to access information put together for their specific needs. You can call up report portfolio reports via a graphical menu. This menu can be set up individually for different user groups. The navigation function in the report portfolio is limited to scrolling through reports created by the relevant department.

Q6. Discuss the challenges involved in data integration and coordination process?

Ans:

Simple Data Integration

The data integration process can often seem overwhelming, and this is often compounded by the vast number of large-scale, complex, and costly enterprise integration applications available on the market. MapForce seeks to alleviate this burden with powerful data integration capabilities built into a straightforward graphical user interface.

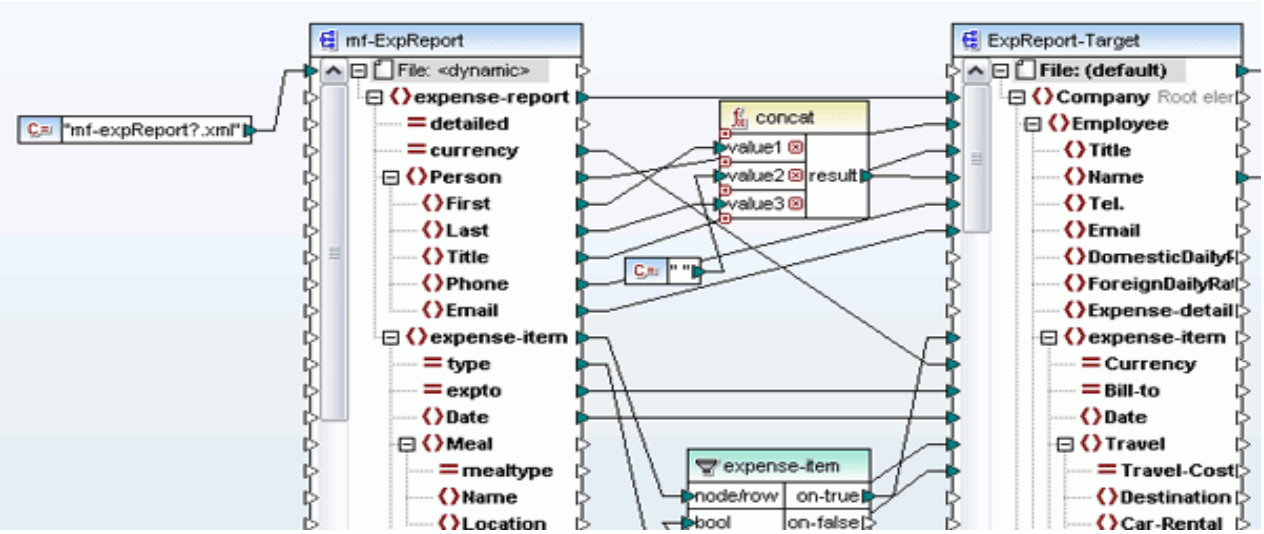


MapForce allows you to easily associate target and source data structures using drag and drop functionality. Advanced data processing filters and functions can be added via a built-in function library, and you can use the visual function builder to combine multiple inline and/or recursive operations in more complex data integration scenarios.

Integrating Data from/into Multiple Files

MapForce lets you easily integrate data from multiple files or split data from one file into many. Multiple files can be specified through support for wildcard characters (e.g., ? or *), a database table, auto-number sequences, or other methods. This feature is very useful in a wide variety of data integration scenarios; for example, it may be necessary to integrate data from a file collection or to generate individual XML files for each

main table record in a large database. The screenshot below shows an example in which two files from a directory are integrated into a single target file.

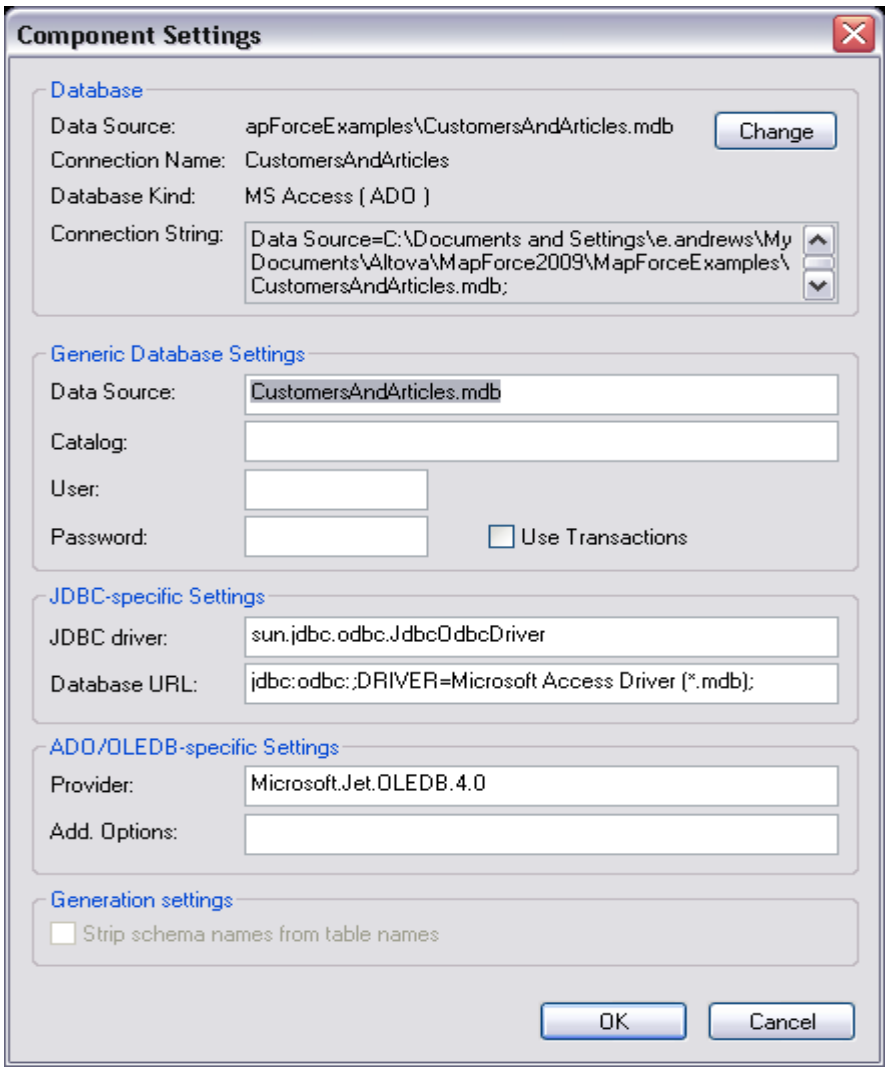


As a complement to this feature, MapForce also allows you to use file names as parameters in your data integration projects. This lets you create dynamic mappings in which this information is defined at run-time.

Re-usable Data Mappings

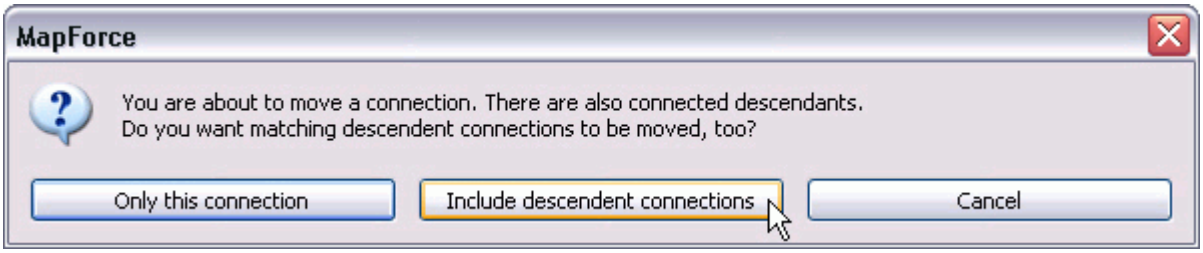
Whether it is an XML or database schema, EDI configuration file, or XBRL taxonomy and beyond, MapForce integrates data based on data structures regardless of the underlying content. This means that you can re-use your data integration mappings again and again as your business data changes.

Simply right click the data structure and choose Properties to access the component settings dialog to change your data source and, consequently, the output of your data integration project.

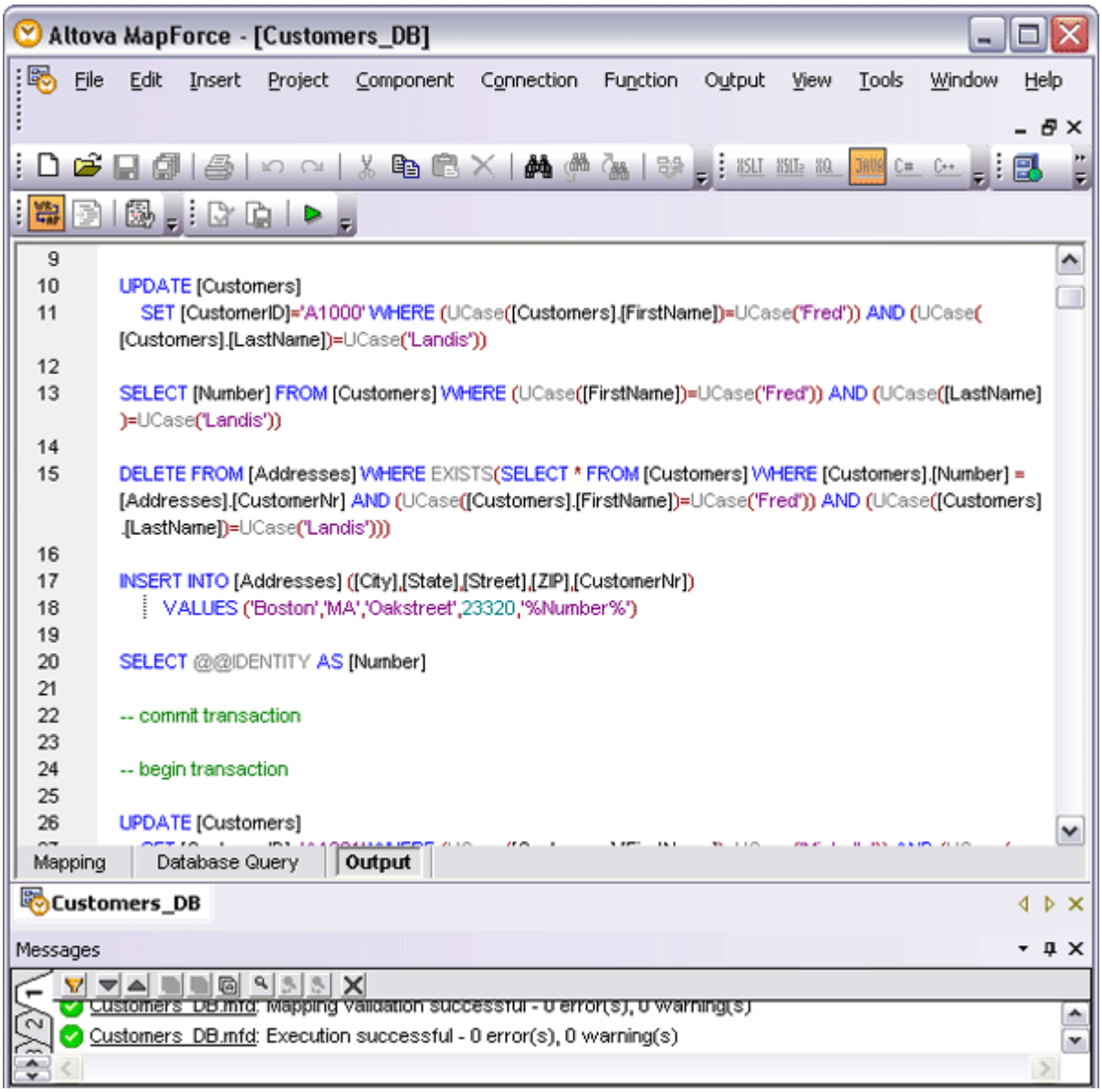


If you need to make some changes to your mapping along the way - to accommodate for underlying schema changes, for instance - MapForce offers a variety of automation features that help ease this process. For

example, when you re-map a parent element, you will be asked if you would like to automatically reassign child elements or any other descendent connections accordingly.

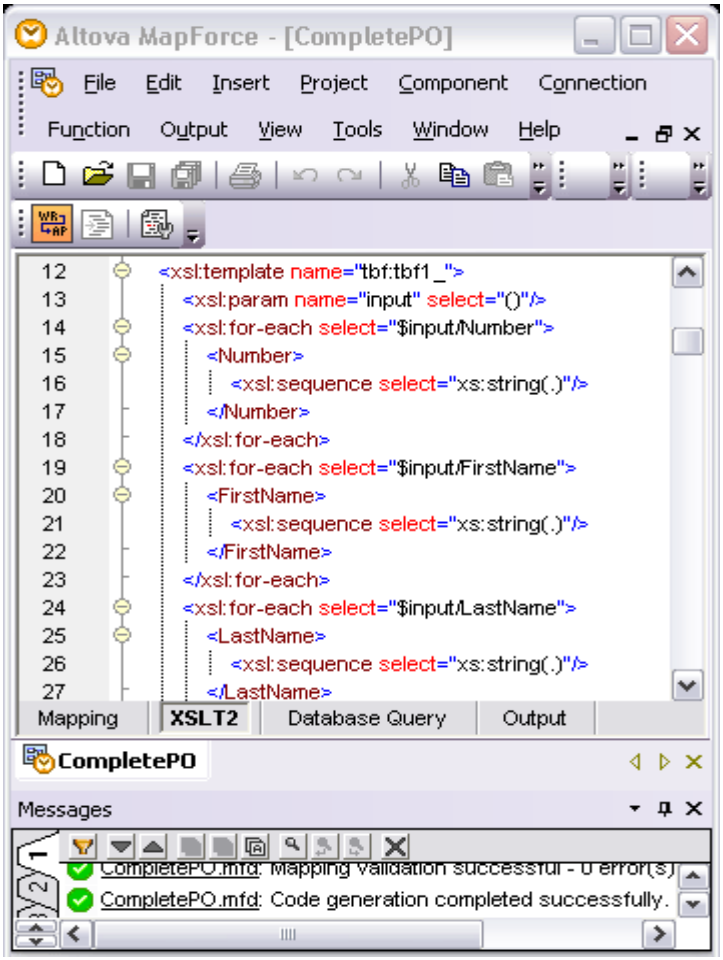


Data integration output is created on-the-fly, and can be viewed at any time by simply clicking the Output tab in the design pane.

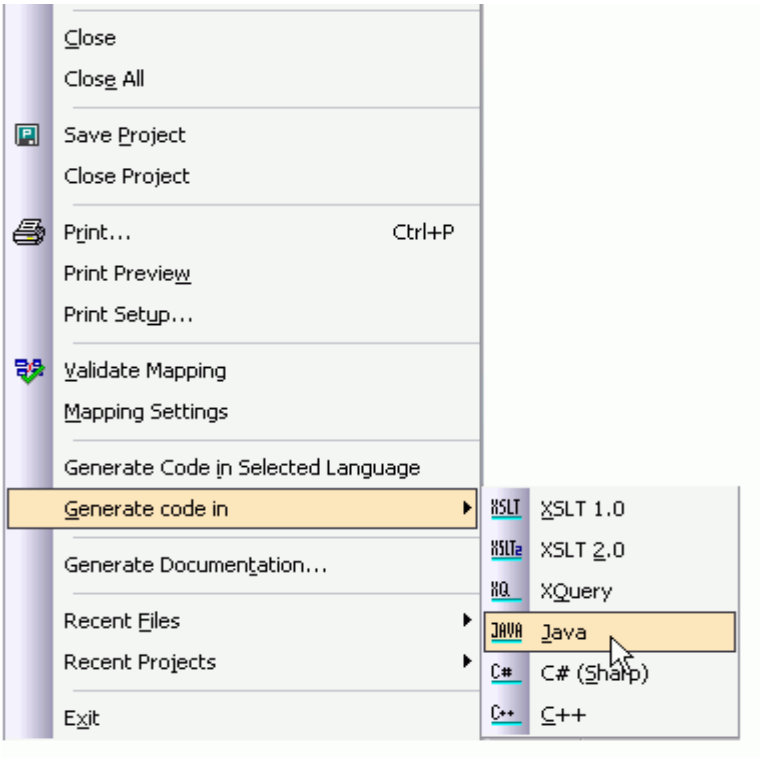


Automated Data Integration

For XML mappings, MapForce automatically generates data integration code on-the-fly in XSLT 1.0/2.0 or XQuery, based on your selection.



MapForce data mappings can also be fully automated through the generation of royalty-free data integration application code in Java, C#, or C++. This enables you to implement scheduled or event-triggered data integration/migration operations for inclusion in any reporting, e-commerce, or SOA-based applications.



MapForce data integration operations can also be automated via data integration API, ActiveX control, or the command line.

Full integration with the Visual Studio and Eclipse IDEs helps developers use MapForce data integration functionality as part of large-scale enterprise projects, without the hefty price tag.

Legacy Data Integration

As technology rapidly advances in the information age, organizations are often left burdened with legacy data repositories that are no longer supported, making the data difficult to access and impossible to edit in its native format. Here, MapForce provides the unique FlexText utility for parsing flat file output so that it can easily be integrated with any other target structure.

FlexText enables you to create reusable legacy data integration templates for mapping flat files to modern data formats like XML, databases, Excel 2007+, XBRL, Web services, and more.

In addition, legacy data formats like EDI can easily be integrated with modern accounting systems like ERP and relational databases, or even translated to modern formats like XML.

Data Coordination Process:

1. A data coordination method of coordinating data between a source application program and a destination application program in an information processing terminal, the information processing terminal including a data storage unit configured to store therein data, a virus pattern file describing characteristics of a computer virus, and a data string pattern file describing a detecting data string; and an applications storage unit that stores therein a plurality of application programs each capable of creating data and storing the data in the data storage unit, and a virus detection program configured to detect a virus contained in the data created by any one of the application programs based on the virus pattern file before storing the data in the data storage unit, the application programs including a source application program that creates a specific data and a destination

application program that makes use of the specific data, the data coordination method comprising: executing the virus detection program whereby the virus detection program looks for a data string in the specific data based on the detecting data string in the data string pattern file in the data storage unit and extracts the data string if such a data string is present in the specific data; and notifying the data string extracted by the virus detection program at the executing and path information that specifies path of the specific data as data coordination information to the destination application program.

2. The data coordination method according to claim 1, further comprising creating and storing the data string pattern file in the storage unit.

3. The data coordination method according to claim 1, wherein the virus pattern file and the data string pattern file being separate files.

4. The data coordination method according to claim 1, wherein the data string pattern file includes pattern information for detecting a data string relating to any one of date information and position information or both included in the data created by any one of the application programs.

5. The data coordination method according to claim 1, wherein the executing includes looking for a data string in the specific data each time the destination application program requests the virus detection program to detect a virus contained in the data.

6. The data coordination method according to claim 1, wherein the executing includes looking for a data string in the specific data each time the destination application program is activated.

7. The data coordination method according to claim 1, wherein the executing includes looking for a data string in the specific data at a timing specified by a user.

8. The data coordination method according to claim 1, wherein the destination application program is a schedule management program that manages schedule by using at least one of calendar information and map information.

9. The data coordination method according to claim 8, wherein the data is e-mail data, and the schedule management program extracts e-mail data from the data storage unit based on the storage destination information, and handles extracted e-mail data in association with the calendar information based on the date information.

10. The data coordination method according to claim 8, wherein the data is image data, and the schedule management program extracts image data from the data storage unit based on the storage destination information, and handles extracted image data in association with the calendar information based on the date information.

11. The data coordination method according to claim 8, wherein the data is e-mail data, and the schedule management program extracts e-mail data from the data storage unit based on the storage destination information, and handles extracted e-mail data in association with the map information.

12. The data coordination method according to claim 8, wherein the data is image data, and the schedule management program extracts image data from the data storage unit based on the storage destination information, and handles extracted image data in association with the map information.

13. A computer-readable recording medium that stores therein a computer program that implements on a computer a data coordination method of coordinating data between a source application program and a destination application program in an information processing terminal, the information processing terminal including a data storage unit configured to store therein data, a virus pattern file describing characteristics of a computer virus, and a data string pattern file describing a detecting data string; and an applications storage unit

that stores therein a plurality of application programs each capable of creating data and storing the data in the data storage unit, and a virus detection program configured to detect a virus contained in the data created by any one of the application programs based on the virus pattern file before storing the data in the data storage unit, the application programs including a source application program that creates a specific data and a destination application program that makes use of the specific data, the computer program causing the computer to execute: executing the virus detection program whereby the virus detection program looks for a data string in the specific data based on the detecting data string in the data string pattern file in the data storage unit and extracts the data string if such a data string is present in the specific data; and notifying the data string extracted by the virus detection program at the executing and path information that specifies path of the specific data as data coordination information to the destination application program.

14. An information processing terminal comprising: a data storage unit configured to store therein data, a virus pattern file describing characteristics of a computer virus, and a data string pattern file describing a detecting data string; an applications storage unit that stores therein a plurality of application programs each capable of creating data and storing the data in the data storage unit, and a virus detection program configured to detect a virus contained in the data created by any one of the application programs based on the virus pattern file before storing the data in the data storage unit, the application programs including a source application program that creates a specific data and a destination application program that makes use of the specific data; an executing unit that executes the virus detection program whereby the virus detection program looks for a data string in the specific data based on the detecting data string in the data string pattern file in the data storage unit and extracts the data string if such a data string is present in the specific data; and a notifying unit that notifies the data string extracted by the virus detection program and path information that specifies path of the specific data as data coordination information to the destination application program.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a technology for coordinating data between various application programs that handle storage of data in a storage unit and a virus detection program that detects a virus contained in the data based on a virus pattern file describing characteristics of viruses.

2. Description of the Related Art

Mobile phones are becoming multifunctional. Most mobile phones now have an e-mail function, a web page browsing function, a music reproducing function, and a photograph function.

In a mobile phone, data files are generally stored in a storage device within the mobile phone. Along with the increase of the functions of mobile phones, types and number of the data files that need to be stored have increased. As a result, there is a need to efficiently group and manage the data files. Various techniques have been proposed to achieve this. One approach includes adding unique classification information to data files. Another approach includes using a data file name to facilitate classification.

For example, Japanese Patent Application Laid-Open (JP-A) No. 2003-134454 discloses a technique that appends a date on which image data was photographed to be included in a data file indicative of the image data. On the contrary, JP-A No. 2001-34632 discloses a technique that appends the name of the place where image data was photographed to be included in a data file indicative of the image data.

Thus, the techniques disclosed in JP-A Nos. 2003-134454 and 2001-34632 include appending unique information to data files. However, some application programs (AP) cannot handle such data files that are

appended with additional information. In other words, there is a limitation on where the conventional techniques can be employed.

Many APs (for example, an e-mail AP, a camera AP, and a web browser AP) are installed in a multifunctional mobile phone, and generally, file formats that are handled by these APs are not the same. Accordingly, although some APs can handle the data files that are appended with additional information, others cannot.

For example, when a user of a mobile phone wishes to rearrange data files stored in a predetermined directory by the camera AP, it is necessary to shift the data file by creating a new directory. However, such an operation is troublesome, and is not convenient for the user.

Data files such as e-mail data files, music files, web page data files, image data files that are handled by mobile phones are in conformity with a standardized format. That is, these data files already include information such as the date when the file is created.

One approach could be to create a search program that can extract information, such as date, from these data files, and install the program to mobile phones. However, creation of a new search program increases the costs.

Therefore, there is a need of a technique that can easily and at low costs coordinate data files among various APs. This issue is not particularly limited to the mobile phones, and applies likewise to information processing terminals such as a personal digital assistant (PDA)

Assignment (Set-2)

Subject code: MI0036

Business intelligence & Tools

Q.1 Explain business development life cycle in detail?

Ans: The **Systems Development Life Cycle (SDLC)**, or *Software Development Life Cycle* in systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems. The concept generally refers to computer or information systems.

In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system: the software development process.

ystems Development Life Cycle (SDLC) is a process used by a systems analyst to develop an information system, including requirements, validation, training, and user (stakeholder) ownership. Any SDLC should result in a high quality system that meets or exceeds customer expectations, reaches completion within time and cost estimates, works effectively and efficiently in the current and planned Information Technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.[2]

Computer systems are complex and often (especially with the recent rise of Service-Oriented Architecture) link multiple traditional systems potentially supplied by different software vendors. To manage this level of

complexity, a number of SDLC models have been created: "waterfall"; "fountain"; "spiral"; "build and fix"; "rapid prototyping"; "incremental"; and "synchronize and stabilize". [3]

SDLC models can be described along a spectrum of agile to iterative to sequential. Agile methodologies, such as XP and Scrum, focus on light-weight processes which allow for rapid changes along the development cycle. Iterative methodologies, such as Rational Unified Process and Dynamic Systems Development Method, focus on limited project scopes and expanding or improving products by multiple iterations. Sequential or big-design-upfront (BDUF) models, such as Waterfall, focus on complete and correct planning to guide large projects and risks to successful and predictable results[*citation needed*]. Other models, such as Anamorphic Development, tend to focus on a form of development that is guided by project scope and adaptive iterations of feature development.

In project management a project can be defined both with a project life cycle (PLC) and an SDLC, during which slightly different activities occur. According to Taylor (2004) "the project life cycle encompasses all the activities of the project, while the systems development life cycle focuses on realizing the product requirements".[4]

Systems development phases

The System Development Life Cycle framework provides system designers and developers to follow a sequence of activities. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one.

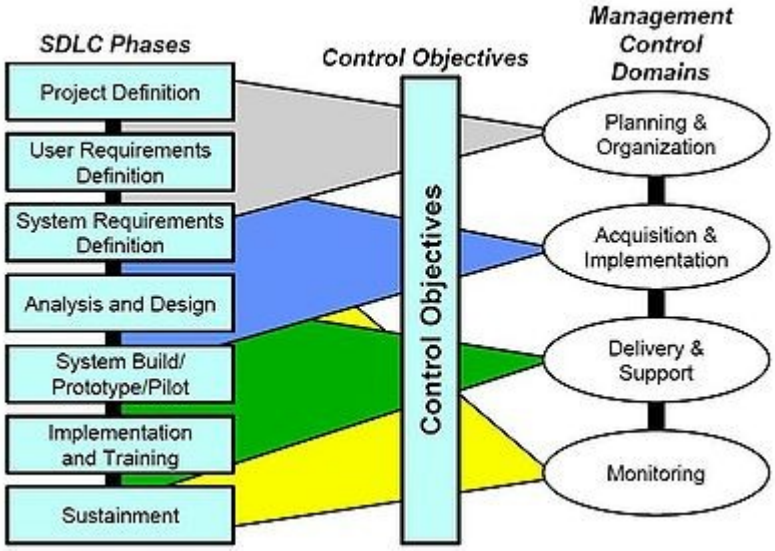
A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation, and are explained in the section below. A number of system development life cycle (SDLC) models have been created: waterfall, fountain, spiral, build and fix, rapid prototyping, incremental, and synchronize and stabilize. The oldest of these, and the best known, is the waterfall model: a sequence of stages in which the output of each stage becomes the input for the next. These stages can be characterized and divided up in different ways, including the following[6]:

- **Project planning, feasibility study**: Establishes a high-level view of the intended project and determines its goals.
- **Systems analysis, requirements definition**: Refines project goals into defined functions and operation of the intended application. Analyzes end-user information needs.
- **Systems design**: Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation.
- **Implementation**: The real code is written here.
- **Integration and testing**: Brings all the pieces together into a special testing environment, then checks for errors, bugs and interoperability.
- **Acceptance, installation, deployment**: The final stage of initial development, where the software is put into production and runs actual business.
- **Maintenance**: What happens during the rest of the software's life: changes, correction, additions, moves to a different computing platform and more. This, the least glamorous and perhaps most important step of all, goes on seemingly forever.

In the following example (see picture) these stage of the Systems Development Life Cycle are divided in ten steps from definition to creation and modification of IT work products:

Systems development life cycle topics

Management and control

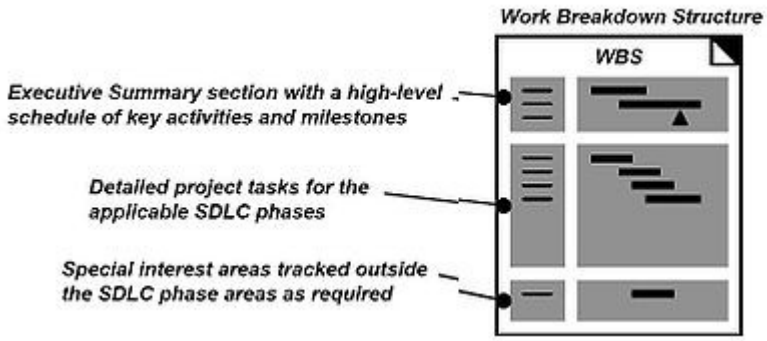


SDLC Phases Related to Management Controls.

The Systems Development Life Cycle (SDLC) phases serve as a programmatic guide to project activity and provide a flexible but consistent way to conduct projects to a depth matching the scope of the project. Each of the SDLC phase objectives are described in this section with key deliverables, a description of recommended tasks, and a summary of related control objectives for effective management. It is critical for the project manager to establish and monitor control objectives during each SDLC phase while executing projects. Control objectives help to provide a clear statement of the desired result or purpose and should be used throughout the entire SDLC process. Control objectives can be grouped into major categories (Domains), and relate to the SDLC phases as shown in the figure.

To manage and control any SDLC initiative, each project will be required to establish some degree of a Work Breakdown Structure (WBS) to capture and schedule the work necessary to complete the project. The WBS and all programmatic material should be kept in the "Project Description" section of the project notebook. The WBS format is mostly left to the project manager to establish in a way that best describes the project work. There are some key areas that must be defined in the WBS as part of the SDLC policy. The following diagram describes three key areas that will be addressed in the WBS in a manner established by the project manager.[8]

Work breakdown structured organization



Work Breakdown Structure.

The upper section of the Work Breakdown Structure (WBS) should identify the major phases and milestones of the project in a summary fashion. In addition, the upper section should provide an overview of the full scope

and timeline of the project and will be part of the initial project description effort leading to project approval. The middle section of the WBS is based on the seven Systems Development Life Cycle (SDLC) phases as a guide for WBS task development. The WBS elements should consist of milestones and "tasks" as opposed to "activities" and have a definitive period (usually two weeks or more). Each task must have a measurable output (e.g. document, decision, or analysis). A WBS task may rely on one or more activities (e.g. software engineering, systems engineering) and may require close coordination with other tasks, either internal or external to the project. Any part of the project needing support from contractors should have a Statement of work (SOW) written to include the appropriate tasks from the SDLC phases. The development of a SOW does not occur during a specific phase of SDLC but is developed to include the work from the SDLC process that may be conducted by external resources such as contractors and struct.

Q.2 Discuss the various components of data ware house?

Ans: The data warehouse architecture is based on a relational database management system server that functions as the central repository for informational data. Operational data and processing is completely separated from data warehouse processing. This central information repository is surrounded by a number of key components designed to make the entire environment functional, manageable and accessible by both the operational systems that source data into the warehouse and by end-user query and analysis tools.

Typically, the source data for the warehouse is coming from the operational applications. As the data enters the warehouse, it is cleaned up and transformed into an integrated structure and format. The transformation process may involve conversion, summarization, filtering and condensation of data. Because the data contains a historical component, the warehouse must be capable of holding and managing large volumes of data as well as different data structures for the same database over time.

The next sections look at the seven major components of data warehousing:

Data Warehouse Database

The central data warehouse database is the cornerstone of the data warehousing environment. This database is almost always implemented on the relational database management system (RDBMS) technology. However, this kind of implementation is often constrained by the fact that traditional RDBMS products are optimized for transactional database processing. Certain data warehouse attributes, such as very large database size, ad hoc query processing and the need for flexible user view creation including aggregates, multi-table joins and drill-downs, have become drivers for different technological approaches to the data warehouse database. These approaches include:

- Parallel relational database designs for scalability that include shared-memory, shared disk, or shared-nothing models implemented on various multiprocessor configurations (symmetric multiprocessors or SMP, massively parallel processors or MPP, and/or clusters of uni- or multiprocessors).
- An innovative approach to speed up a traditional RDBMS by using new index structures to bypass relational table scans.
- Multidimensional databases (MDDBs) that are based on proprietary database technology; conversely, a dimensional data model can be implemented using a familiar RDBMS. Multi-dimensional databases are designed to overcome any limitations placed on the warehouse by the nature of the relational data model. MDDBs enable on-line analytical processing (OLAP) tools that architecturally belong to a group of data warehousing components jointly categorized as the data query, reporting, analysis and mining tools.

Sourcing, Acquisition, Cleanup and Transformation Tools

A significant portion of the implementation effort is spent extracting data from operational systems and putting it in a format suitable for informational applications that run off the data warehouse.

The data sourcing, cleanup, transformation and migration tools perform all of the conversions, summarizations, key changes, structural changes and condensations needed to transform disparate data into information that can be used by the decision support tool. They produce the programs and control statements, including the COBOL programs, MVS job-control language (JCL), UNIX scripts, and SQL data definition language (DDL) needed to move data into the data warehouse for multiple operational systems. These tools also maintain the meta data. The functionality includes:

- Removing unwanted data from operational databases
- Converting to common data names and definitions
- Establishing defaults for missing data
- Accommodating source data definition changes

The data sourcing, cleanup, extract, transformation and migration tools have to deal with some significant issues including:

- Database heterogeneity. DBMSs are very different in data models, data access language, data navigation, operations, concurrency, integrity, recovery etc.
- Data heterogeneity. This is the difference in the way data is defined and used in different models - homonyms, synonyms, unit compatibility (U.S. vs metric), different attributes for the same entity and different ways of modeling the same fact.

These tools can save a considerable amount of time and effort. However, significant shortcomings do exist. For example, many available tools are generally useful for simpler data extracts. Frequently, customized extract routines need to be developed for the more complicated data extraction procedures.

Meta data

Meta data is data about data that describes the data warehouse. It is used for building, maintaining, managing and using the data warehouse. Meta data can be classified into:

- Technical meta data, which contains information about warehouse data for use by warehouse designers and administrators when carrying out warehouse development and management tasks.
- Business meta data, which contains information that gives users an easy-to-understand perspective of the information stored in the data warehouse.

Equally important, meta data provides interactive access to users to help understand content and find data. One of the issues dealing with meta data relates to the fact that many data extraction tool capabilities to gather meta data remain fairly immature. Therefore, there is often the need to create a meta data interface for users, which may involve some duplication of effort.

Meta data management is provided via a meta data repository and accompanying software. Meta data repository management software, which typically runs on a workstation, can be used to map the source data to the target database; generate code for data transformations; integrate and transform the data; and control moving data to the warehouse.

As user's interactions with the data warehouse increase, their approaches to reviewing the results of their requests for information can be expected to evolve from relatively simple manual analysis for trends and exceptions to agent-driven initiation of the analysis based on user-defined thresholds. The definition of these thresholds, configuration parameters for the software agents using them, and the information directory indicating where the appropriate sources for the information can be found are all stored in the meta data repository as well.

Access Tools

The principal purpose of data warehousing is to provide information to business users for strategic decision-making. These users interact with the data warehouse using front-end tools. Many of these tools require an information specialist, although many end users develop expertise in the tools. Tools fall into four main

categories: query and reporting tools, application development tools, online analytical processing tools, and data mining tools.

Query and Reporting tools can be divided into two groups: reporting tools and managed query tools. Reporting tools can be further divided into production reporting tools and report writers. Production reporting tools let companies generate regular operational reports or support high-volume batch jobs such as calculating and printing paychecks. Report writers, on the other hand, are inexpensive desktop tools designed for end-users.

Managed query tools shield end users from the complexities of SQL and database structures by inserting a metalayer between users and the database. These tools are designed for easy-to-use, point-and-click operations that either accept SQL or generate SQL database queries.

Often, the analytical needs of the data warehouse user community exceed the built-in capabilities of query and reporting tools. In these cases, organizations will often rely on the tried-and-true approach of in-house application development using graphical development environments such as PowerBuilder, Visual Basic and Forte. These application development platforms integrate well with popular OLAP tools and access all major database systems including Oracle, Sybase, and Informix.

OLAP tools are based on the concepts of dimensional data models and corresponding databases, and allow users to analyze the data using elaborate, multidimensional views. Typical business applications include product performance and profitability, effectiveness of a sales program or marketing campaign, sales forecasting and capacity planning. These tools assume that the data is organized in a multidimensional model. A critical success factor for any business today is the ability to use information effectively. Data mining is the process of discovering meaningful new correlations, patterns and trends by digging into large amounts of data stored in the warehouse using artificial intelligence, statistical and mathematical techniques.

Data Marts

The concept of a data mart is causing a lot of excitement and attracts much attention in the data warehouse industry. Mostly, data marts are presented as an alternative to a data warehouse that takes significantly less time and money to build. However, the term data mart means different things to different people. A rigorous definition of this term is a data store that is subsidiary to a data warehouse of integrated data. The data mart is directed at a partition of data (often called a subject area) that is created for the use of a dedicated group of users. A data mart might, in fact, be a set of denormalized, summarized, or aggregated data. Sometimes, such a set could be placed on the data warehouse rather than a physically separate store of data. In most instances, however, the data mart is a physically separate store of data and is resident on separate database server, often a local area network serving a dedicated user group. Sometimes the data mart simply comprises relational OLAP technology which creates highly denormalized dimensional model (e.g., star schema) implemented on a relational database. The resulting hypercubes of data are used for analysis by groups of users with a common interest in a limited portion of the database.

These types of data marts, called dependent data marts because their data is sourced from the data warehouse, have a high value because no matter how they are deployed and how many different enabling technologies are used, different users are all accessing the information views derived from the single integrated version of the data.

Unfortunately, the misleading statements about the simplicity and low cost of data marts sometimes result in organizations or vendors incorrectly positioning them as an alternative to the data warehouse. This viewpoint defines independent data marts that in fact, represent fragmented point solutions to a range of business problems in the enterprise. This type of implementation should be rarely deployed in the context of an overall technology or applications architecture. Indeed, it is missing the ingredient that is at the heart of the data warehousing concept -- that of data integration. Each independent data mart makes its own assumptions about how to consolidate the data, and the data across several data marts may not be consistent.

Moreover, the concept of an independent data mart is dangerous -- as soon as the first data mart is created, other organizations, groups, and subject areas within the enterprise embark on the task of building their own data marts. As a result, you create an environment where multiple operational systems feed multiple non-integrated data marts that are often overlapping in data content, job scheduling, connectivity and management.

In other words, you have transformed a complex many-to-one problem of building a data warehouse from operational and external data sources to a many-to-many sourcing and management nightmare.

Data Warehouse Administration and Management

Data warehouses tend to be as much as 4 times as large as related operational databases, reaching terabytes in size depending on how much history needs to be saved. They are not synchronized in real time to the associated operational data but are updated as often as once a day if the application requires it.

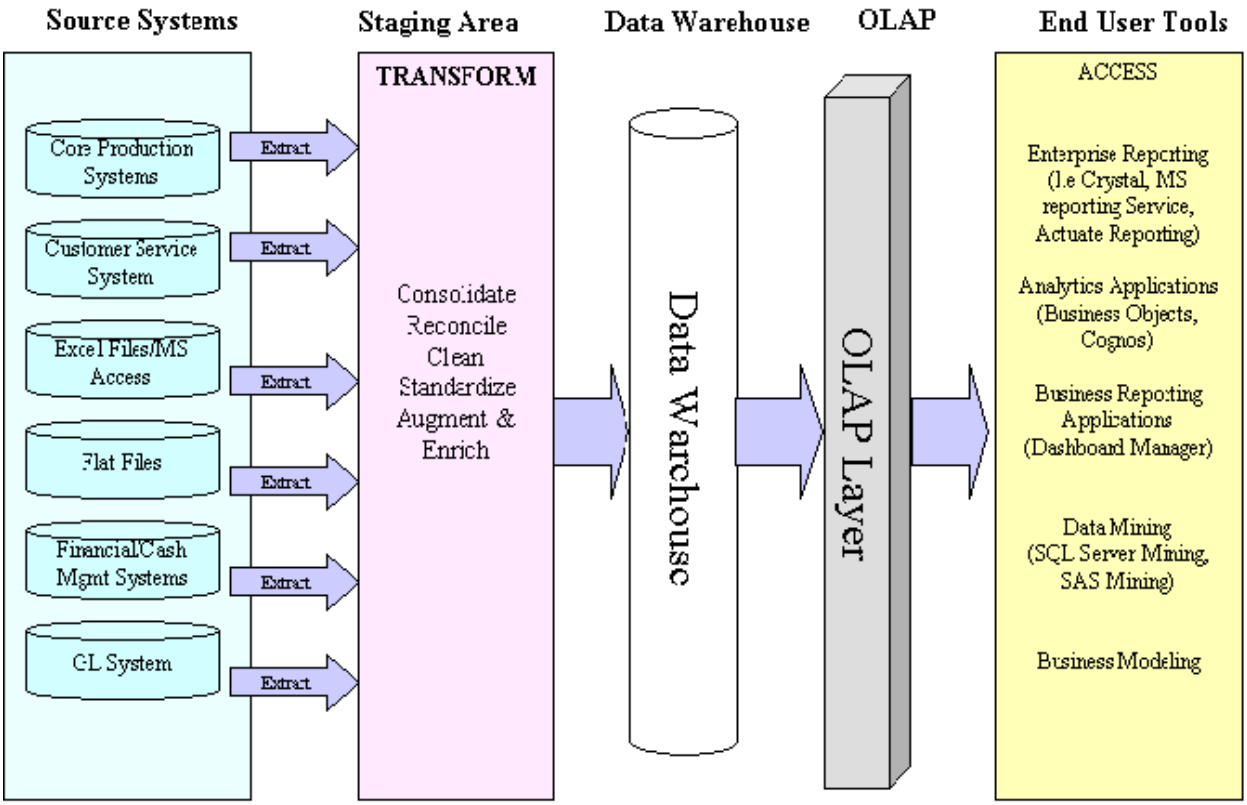
In addition, almost all data warehouse products include gateways to transparently access multiple enterprise data sources without having to rewrite applications to interpret and utilize the data. Furthermore, in a heterogeneous data warehouse environment, the various databases reside on disparate systems, thus requiring inter-networking tools. The need to manage this environment is obvious.

Managing data warehouses includes security and priority management; monitoring updates from the multiple sources; data quality checks; managing and updating meta data; auditing and reporting data warehouse usage and status; purging data; replicating, subsetting and distributing data; backup and recovery and data warehouse storage management.

Information Delivery System

The information delivery component is used to enable the process of subscribing for data warehouse information and having it delivered to one or more destinations according to some user-specified scheduling algorithm. In other words, the information delivery system distributes warehouse-stored data and other information objects to other data warehouses and end-user products such as spreadsheets and local databases. Delivery of information may be based on time of day or on the completion of an external event. The rationale for the delivery systems component is based on the fact that once the data warehouse is installed and operational, its users don't have to be aware of its location and maintenance. All they need is the report or an analytical view of data at a specific point in time. With the proliferation of the Internet and the World Wide Web such a delivery system may leverage the convenience of the Internet by delivering warehouse-enabled information to thousands of end-users via the ubiquitous world wide network.

In fact, the Web is changing the data warehousing landscape since at the very high level the goals of both the Web and data warehousing are the same: easy access to information. The value of data warehousing is maximized when the right information gets into the hands of those individuals who need it, where they need it and they need it most. However, many corporations have struggled with complex client/server systems to give end users the access they need. The issues become even more difficult to resolve when the users are physically remote from the data warehouse location. The Web removes a lot of these issues by giving users universal and relatively inexpensive access to data. Couple this access with the ability to deliver required information on demand and the result is a web-enabled information delivery system that allows users dispersed across continents to perform a sophisticated business-critical analysis and to engage in collective decision-making.



Q.3 Discuss data extraction process? What are the various methods being used for data extraction?

Ans: **Data extract**: Data extract is the output of the data extraction process, a very important aspect of data warehouse implementation.

A data warehouse gathers data from several sources and utilizes these data to serve as vital information for the company. These data will be used to spot patterns and trends both in the business operations as well as in industry standards.

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 of 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.

Data extracts are loaded into the staging area of a relational database which for future manipulation in the ETL methodology.

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. Some database professionals implement data extraction using extraction logic in the data warehouse staging area and query the source system for data using applications programming interface (API).

Data extraction is a complex process but there are various software applications that have been developed to handle this process.

Some generic extraction applications can be found free on the internet. A CD extraction software can create digital copies of audio CDs on the hard drive. There also email extraction tools which can extract email addresses from different websites including results from Google searches. These emails can be exported to text, html or XML formats.

Another data extracting tool is a web data or link extractor which can extra URLs, meta tags (like keywords, title and descriptions), body texts, email addresses, phone and fax numbers and many other data from a website.

There is a wide array of data extracting tools. Some are used for individual purposes such as extracting data for entertainment while some are used for big projects like data warehousing.

Since data warehouses need to do other processes and not just extracting alone, database managers or programmers usually write programs that repetitively checks on many different sites or new data updates. This way, the code just sits in one area of the data warehouse sensing new updates from the data sources. Whenever an new data is detected, the program automatically does its function to update and transfer the data to the ETL process.

Three common methods for data extraction

Probably the most common technique used traditionally to do this is to cook up some regular expressions that match the pieces you want (e.g., URL's and link titles). Our screen-scraper software actually started out as an application written in Perl for this very reason. In addition to regular expressions, you might also use some code written in something like Java or Active Server Pages to parse out larger chunks of text. Using raw regular expressions to pull out the data can be a little intimidating to the uninitiated, and can get a bit messy when a script contains a lot of them. At the same time, if you're already familiar with regular expressions, and your scraping project is relatively small, they can be a great solution.

Other techniques for getting the data out can get very sophisticated as algorithms that make use of artificial intelligence and such are applied to the page. Some programs will actually analyze the semantic content of an HTML page, then intelligently pull out the pieces that are of interest. Still other approaches deal with developing "ontologies", or hierarchical vocabularies intended to represent the content domain.

There are a number of companies (including our own) that offer commercial applications specifically intended to do screen-scraping. The applications vary quite a bit, but for medium to large-sized projects they're often a good solution. Each one will have its own learning curve, so you should plan on taking time to learn the ins and outs of a new application. Especially if you plan on doing a fair amount of screen-scraping it's probably a good idea to at least shop around for a screen-scraping application, as it will likely save you time and money in the long run.

So what's the best approach to data extraction? It really depends on what your needs are, and what resources you have at your disposal. Here are some of the pros and cons of the various approaches, as well as suggestions on when you might use each one:

Raw regular expressions and code

Advantages:

- If you're already familiar with regular expressions and at least one programming language, this can be a quick solution.
- Regular expressions allow for a fair amount of "fuzziness" in the matching such that minor changes to the content won't break them.
- You likely don't need to learn any new languages or tools (again, assuming you're already familiar with regular expressions and a programming language).
- Regular expressions are supported in almost all modern programming languages. Heck, even VBScript has a regular expression engine. It's also nice because the various regular expression implementations don't vary too significantly in their syntax.

Disadvantages:

- They can be complex for those that don't have a lot of experience with them. Learning regular expressions isn't like going from Perl to Java. It's more like going from Perl to XSLT, where you have to wrap your mind around a completely different way of viewing the problem.
- They're often confusing to analyze. Take a look through some of the regular expressions people have created to match something as simple as an email address and you'll see what I mean.
- If the content you're trying to match changes (e.g., they change the web page by adding a new "font" tag) you'll likely need to update your regular expressions to account for the change.
- The data discovery portion of the process (traversing various web pages to get to the page containing the data you want) will still need to be handled, and can get fairly complex if you need to deal with cookies and such.

When to use this approach: You'll most likely use straight regular expressions in screen-scraping when you have a small job you want to get done quickly. Especially if you already know regular expressions, there's no sense in getting into other tools if all you need to do is pull some news headlines off of a site.

Ontologies and artificial intelligence

Advantages:

- You create it once and it can more or less extract the data from any page within the content domain you're targeting.
- The data model is generally built in. For example, if you're extracting data about cars from web sites the extraction engine already knows what the make, model, and price are, so it can easily map them to existing data structures (e.g., insert the data into the correct locations in your database).
- There is relatively little long-term maintenance required. As web sites change you likely will need to do very little to your extraction engine in order to account for the changes.

Disadvantages:

- It's relatively complex to create and work with such an engine. The level of expertise required to even understand an extraction engine that uses artificial intelligence and ontologies is much higher than what is required to deal with regular expressions.
- These types of engines are expensive to build. There are commercial offerings that will give you the basis for doing this type of data extraction, but you still need to configure them to work with the specific content domain you're targeting.
- You still have to deal with the data discovery portion of the process, which may not fit as well with this approach (meaning you may have to create an entirely separate engine to handle data discovery). Data discovery is the process of crawling web sites such that you arrive at the pages where you want to extract data.

When to use this approach: Typically you'll only get into ontologies and artificial intelligence when you're planning on extracting information from a very large number of sources. It also makes sense to do this when the data you're trying to extract is in a very unstructured format (e.g., newspaper classified ads). In cases where the data is very structured (meaning there are clear labels identifying the various data fields), it may make more sense to go with regular expressions or a screen-scraping application.

Screen-scraping software

Advantages:

- Abstracts most of the complicated stuff away. You can do some pretty sophisticated things in most screen-scraping applications without knowing anything about regular expressions, HTTP, or cookies.
- Dramatically reduces the amount of time required to set up a site to be scraped. Once you learn a particular screen-scraping application the amount of time it requires to scrape sites vs. other methods is significantly lowered.
- Support from a commercial company. If you run into trouble while using a commercial screen-scraping application, chances are there are support forums and help lines where you can get assistance.

Disadvantages:

- The learning curve. Each screen-scraping application has its own way of going about things. This may imply learning a new scripting language in addition to familiarizing yourself with how the core application works.
- A potential cost. Most ready-to-go screen-scraping applications are commercial, so you'll likely be paying in dollars as well as time for this solution.
- A proprietary approach. Any time you use a proprietary application to solve a computing problem (and proprietary is obviously a matter of degree) you're locking yourself into using that approach. This may or may not be a big deal, but you should at least consider how well the application you're using will integrate with other software applications you currently have. For example, once the screen-scraping application has extracted the data how easy is it for you to get to that data from your own code?

When to use this approach: Screen-scraping applications vary widely in their ease-of-use, price, and suitability to tackle a broad range of scenarios. Chances are, though, that if you don't mind paying a bit, you can save yourself a significant amount of time by using one. If you're doing a quick scrape of a single page you can use just about any language with regular expressions. If you want to extract data from hundreds of web sites that are all formatted differently you're probably better off investing in a complex system that uses ontologies and/or artificial intelligence. For just about everything else, though, you may want to consider investing in an application specifically designed for screen-scraping.

As an aside, I thought I should also mention a recent project we've been involved with that has actually required a hybrid approach of two of the aforementioned methods. We're currently working on a project that deals with extracting newspaper classified ads. The data in classifieds is about as unstructured as you can get. For example, in a real estate ad the term "number of bedrooms" can be written about 25 different ways. The data extraction portion of the process is one that lends itself well to an ontologies-based approach, which is what we've done. However, we still had to handle the data discovery portion. We decided to use screen-scraper for that, and it's handling it just great. The basic process is that screen-scraper traverses the various pages of the site, pulling out raw chunks of data that constitute the classified ads. These ads then get passed to code we've written that uses ontologies in order to extract out the individual pieces we're after. Once the data has been extracted we then insert it into a database.

Q.4 Discuss the needs of developing OLAP tools in details?

Ans: OLAP

Short for ***On****line* ***A****nalytical* ***P****rocessing,* a category of software tools that provides analysis of data stored in a database. OLAP tools enable users to analyze different dimensions of multidimensional data. For example, it provides time series and trend analysis views. OLAP often is used in data mining.

The chief component of OLAP is the OLAP server, which sits between a client and a database management systems (DBMS). The OLAP server understands how data is organized in the database and has special functions for analyzing the data. There are OLAP servers available for nearly all the major database systems.

The first commercial multidimensional (OLAP) products appeared approximately 30 years ago (Express). When Edgar Codd introduced the OLAP definition in his 1993 white paper, there were already dozens of OLAP products for client/server and desktop/file server environments. Usually those products were expensive, proprietary, standalone systems afforded only by large corporations, and performed only OLAP functions.

After Codd's research appeared, the software industry began appreciating OLAP functionality and many companies have integrated OLAP features into their products (RDBMS, integrated business intelligence suites, reporting tools, portals, etc.). In addition, for the last decade, pure OLAP tools have considerably improved and become cheaper and more user-friendly.

These developments brought OLAP functionality to a much broader range of users and organizations. Now OLAP is used not only for strategic decision-making in large corporations, but also to make daily tactical decisions about how to better streamline business operations in organizations of all sizes and shapes.

However, the acceptance of OLAP is far from maxi mized. For example, one year ago, The OLAP Survey 2 found that only thirty percent of its participants actually used OLAP.

General purpose tools with OLAP capabilities

Organizations do not want to use pure OLAP tools or integrated business intelligence suites for different reasons. But many of organizations may want to use OLAP capabilities integrated into popular general purpose applications development tools which they already use. In this case, the organizations do not need to buy and deploy new software products, train staff to use them or hire new people.

There is another argument for creating general purpose tools with OLAP capabilities. End users work with the information they need via applications. The effectiveness of this work depends very much on the number of applications (and the interfaces, data formats, etc. associated with them). So it is very desirable to reduce the number of applications (ideally to one application). General purpose tools with OLAP capabilities allow us to reach the goal. In other words, there's no need to use separate applications based on pure OLAP tools.

The advantages of such an approach to developers and end users are clear, and Microsoft and Oracle have recognized this. Both corporations have steadily integrated OLAP into their RDBMSs and general purpose database application development tools.

Microsoft provides SQL Server to handle a relational view of data and the Analysis Services OLAP engine to handle a multidimensional cube view of data. Analysis Services provides the OLE DB for OLAP API and the MDX language for processing multidimensional cubes, which can be physically stored in relational tables or a multidimensional store. Microsoft Excel and Microsoft Office both provide access to Analysis Services data.

Oracle has finally incorporated the Express OLAP engine into the Oracle9i Database Enterprise Edition Release 2 (OLAP Option). Multidimensional cubes are stored in analytical workspaces, which are managed in an Oracle database using an abstract data type. The existing Oracle tools such as PL/SQL, Oracle Reports, Oracle Discoverer and Oracle BI Beans can query and analyze analytical workspaces. The OLAP API is Java-based and supports a rich OLAP manipulation language, which can be considered to be the multidimensional equivalent of Oracle PL/SQL.

Granulated OLAP

There is yet another type of OLAP tool, different from pure OLAP tools and general purpose tools with OLAP capabilities: OLAP components. It seems that this sort of OLAP is not as appreciated.

The OLAP component is the minimal and elementary tool (granula) for developers to embed OLAP functionality in applications. So we can say that OLAP components are granulated OLAP.

Each OLAP component is used within some application development environment. At present, almost all known OLAP components are ActiveX or VCL ones.

All OLAP components are divided into two classes: OLAP components without an OLAP engine (MOLAP components) and OLAP components with this engine (ROLAP components).

The OLAP components without OLAP engines allow an application to access existing multidimensional cubes on MOLAP server that performs a required operation and returns results to the application. At present all the OLAP components of this kind are designed for access to MS Analytical Services (more precisely, they use OLE DB for OLAP) and were developed by Microsoft partners (Knosys, Matrix, etc.).

The ROLAP components (with OLAP engine) are of much greater interest than the MOLAP ones and general purpose tools with OLAP capabilities because they allow you to create flexible and efficient applications that cannot be developed with other tools.

The ROLAP component with an OLAP engine has three interfaces:

- A data access mechanism interface through which it gets access to data sources
- APIs through which a developer uses a language like MDX to define data access and processing to build a multidimensional table (cube). The developer manages properties and behavior of the component so it fully conforms to the application in which the component is embedded
- End-user GUI that can pivot, filter, drill down and drill up data and generate numbers of views from a multidimensional table (cube).

As a rule, ROLAP components are used in client-side applications, so the OLAP engine functions on a client PC. Data access mechanisms like BDE (Borland Database Engines) or ADO.NET allow you to get source data from relational tables and flat files of an enterprise. PC performance nowadays allows the best ROLAP components to quickly process hundreds of thousands or even millions of records from these data sources, and dynamically build multidimensional cubes and perform operations with them. So very effective ROLAP and DOLAP are realized -- and in many cases, they are more preferable than MOLAP.

For example, the low price and simplicity of using a ROLAP component is the obvious (and possibly the only) choice for developers to create mass-deployed small and cheap applications, especially single-user DOLAP applications. Another field in which these components may be preferable is real-time analytical applications (no need to create and maintain MOLAP server, load cubes).

The most widely-known OLAP component is the Microsoft Pivot Table, which has an OLAP engine and access to MS Analytical Services so the component is both MOLAP and ROLAP. Another well-known OLAP (ROLAP) component is DesicionCube of Borland corporation.

Some ROLAP components have the ability to store dynamically-built multidimensional cubes which are usually named microcubes. This feature deserves attention from applications architects and designers because it allows them to develop flexible and cheap applications like enterprise-wide distributed corporate reporting system or Web-based applications.

For example, the ContourCube component stores a microcube with all associated metadata (in fact, it is a container of an analytical application like an Excel workbook) in compressed form (from 10 up to 100 times).

So this microcube is optimized for use on the Internet and can be transferred through HTTP and FTP protocols, and via e-mail. An end user with ContourCube is able to fully analyze the microcube. InterSoft Lab, developer of ContourCube component, has also developed several additional tools to facilitate the development, deployment and use of such distributed applications.

At present, there is a broad range of pure OLAP tools, general purpose tools with OLAP capabilities and OLAP components. To make the best choice, developers must understand the benefits and disadvantages of all sorts of OLAP.

Q.5 what do you understand by the term statistical analysis? Discuss the most important statistical techniques?

Ans: Developments in the field of statistical data analysis often parallel or follow advancements in other fields to which statistical methods are fruitfully applied. Because practitioners of the statistical analysis often address particular applied decision problems, methods developments is consequently motivated by the search to a better decision making under uncertainties.

Decision making process under uncertainty is largely based on application of statistical data analysis for probabilistic risk assessment of your decision. Managers need to understand variation for two key reasons. First, so that they can lead others to apply statistical thinking in day to day activities and secondly, to apply the concept for the purpose of continuous improvement. This course will provide you with hands-on experience to promote the use of statistical thinking and techniques to apply them to make educated decisions whenever there is variation in business data. Therefore, it is a course in statistical thinking via a data-oriented approach.

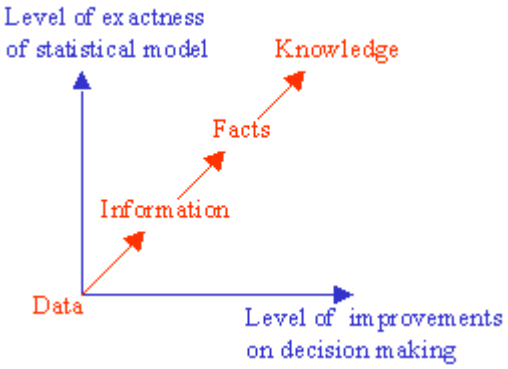
Statistical models are currently used in various fields of business and science. However, the terminology differs from field to field. For example, the fitting of models to data, called calibration, history matching, and data assimilation, are all synonymous with parameter estimation.

Your organization database contains a wealth of information, yet the decision technology group members tap a fraction of it. Employees waste time scouring multiple sources for a database. The decision-makers are frustrated because they cannot get business-critical data exactly when they need it. Therefore, too many decisions are based on guesswork, not facts. Many opportunities are also missed, if they are even noticed at all.

Knowledge is what we know well. Information is the communication of knowledge. In every knowledge exchange, there is a sender and a receiver. The sender make common what is private, does the informing, the communicating. Information can be classified as **explicit and tacit** forms. The explicit information can be explained in structured form, while tacit information is inconsistent and fuzzy to explain. Know that data are only crude information and not knowledge by themselves.

Data is known to be crude information and not knowledge by itself. The sequence from data to knowledge is: **from Data to Information, from Information to Facts, and finally, from Facts to Knowledge**. Data becomes information, when it becomes relevant to your decision problem. Information becomes fact, when the data can support it. Facts are what the data reveals. However the decisive instrumental (i.e., applied) knowledge is expressed together with some statistical degree of confidence.

Fact becomes knowledge, when it is used in the successful completion of a decision process. Once you have a massive amount of facts integrated as knowledge, then your mind will be superhuman in the same sense that mankind with writing is superhuman compared to mankind before writing. The following figure illustrates the statistical thinking process based on data in constructing statistical models for decision making under uncertainties.



The above figure depicts the fact that as the exactness of a statistical model increases, the level of improvements in decision-making increases. That's why we need statistical data analysis. Statistical data analysis arose from the need to place knowledge on a systematic evidence base. This required a study of the laws of probability, the development of measures of data properties and relationships, and so on.

Statistical inference aims at determining whether any statistical significance can be attached that results after due allowance is made for any random variation as a source of error. Intelligent and critical inferences cannot be made by those who do not understand the purpose, the conditions, and applicability of the various techniques for judging significance.

Considering the uncertain environment, the chance that "good decisions" are made increases with the availability of "good information." The chance that "good information" is available increases with the level of structuring the process of Knowledge Management. The above figure also illustrates the fact that as the exactness of a statistical model increases, the level of improvements in decision-making increases.

Knowledge is more than knowing something technical. Knowledge needs wisdom. Wisdom is the power to put our time and our knowledge to the proper use. Wisdom comes with age and experience. Wisdom is the accurate application of accurate knowledge and its key component is to knowing the limits of your knowledge. Wisdom is about knowing how something technical can be best used to meet the needs of the decision-maker. Wisdom, for example, creates statistical software that is useful, rather than technically brilliant. For example, ever since the Web entered the popular consciousness, observers have noted that it puts information at your fingertips but tends to keep wisdom out of reach.

Almost every professionals need a statistical toolkit. Statistical skills enable you to intelligently collect, analyze and interpret data relevant to their decision-making. Statistical concepts enable us to solve problems in a diversity of contexts. Statistical thinking enables you to add substance to your decisions.

The appearance of computer software, JavaScript Applets, Statistical Demonstrations Applets, and Online Computation are the most important events in the process of teaching and learning concepts in model-based statistical decision making courses. These tools allow you to construct numerical examples to understand the concepts, and to find their significance for yourself.

We will apply the basic concepts and methods of statistics you've already learned in the previous statistics course to the real world problems. The course is tailored to meet your needs in the statistical business-data analysis using widely available *commercial* statistical computer packages such as SAS and SPSS. By doing this, you will inevitably find yourself asking questions about the data and the method proposed, and you will have the means at your disposal to settle these questions to your own satisfaction. Accordingly, all the applications problems are borrowed from business and economics. By the end of this course you'll be able to think statistically while performing any data analysis.

There are two general views of teaching/learning statistics: Greater and Lesser Statistics. Greater statistics is everything related to learning from data, from the first planning or collection, to the last presentation or report. Lesser statistics is the body of statistical methodology. This is *a Greater Statistics course*.

There are basically two kinds of "statistics" courses. The real kind shows you how to make sense out of data. These courses would include all the recent developments and all share a deep respect for data and truth. The imitation kind involves plugging numbers into statistics formulas. The emphasis is on doing the arithmetic correctly. These courses generally have no interest in data or truth, and the problems are generally arithmetic exercises. If a certain assumption is needed to justify a procedure, they will simply tell you to "assume the ... are normally distributed" -- no matter how unlikely that might be. It seems like you all are suffering from an overdose of the latter. This course will bring out the joy of statistics in you.

Statistics is a science assisting you to make decisions under uncertainties (based on some numerical and measurable scales). Decision making process must be based on data neither on personal opinion nor on belief.

It is already an accepted fact that "Statistical thinking will one day be as necessary for efficient citizenship as the ability to read and write." So, let us be ahead of our time.

Analysis Of Variance

An important technique for analyzing the effect of categorical factors on a response is to perform an Analysis of Variance. An ANOVA decomposes the variability in the response variable amongst the different factors. Depending upon the type of analysis, it may be important to determine: (a) which factors have a significant effect on the response, and/or (b) how much of the variability in the response variable is attributable to each factor.

STATGRAPHICS Centurion provides several procedures for performing an analysis of variance:

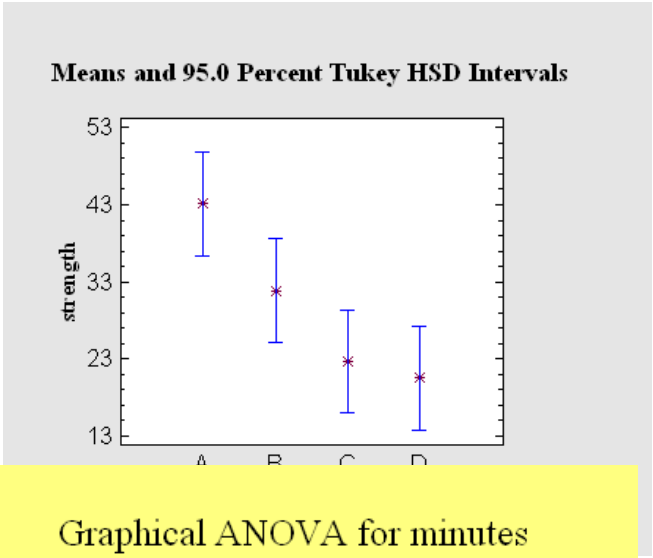
1. *One-Way ANOVA* - used when there is only a single categorical factor. This is equivalent to comparing multiple groups of data.

2. *Multifactor ANOVA* - used when there is more than one categorical factor, arranged in a crossed pattern. When factors are crossed, the levels of one factor appear at more than one level of the other factors.

3. *Variance Components Analysis* - used when there are multiple factors, arranged in a hierarchical manner. In such a design, each factor is nested in the factor above it.

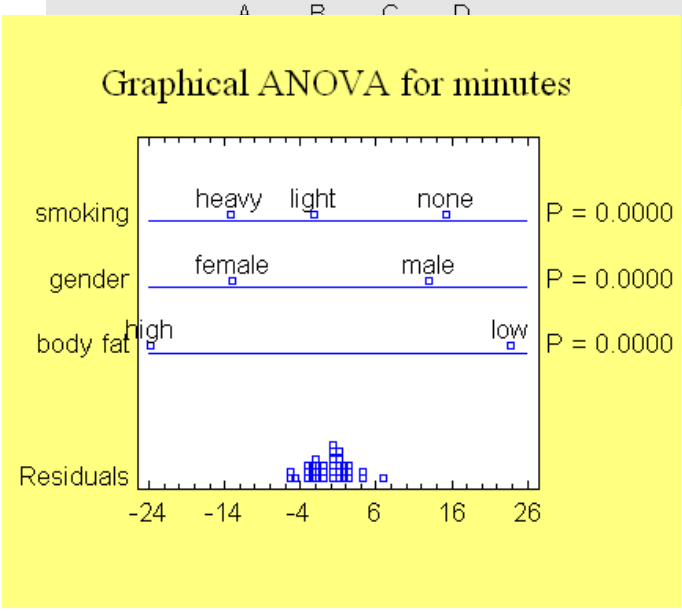
4. *General Linear Models* - used whenever there are both crossed and nested factors, when some factors are fixed and some are random, and when both categorical and quantitative factors are present.

One-Way ANOVA



A one-way analysis of variance is used when the data are divided into groups according to only one factor. The questions of interest are usually: (a) Is there a significant difference between the groups?, and (b) If so, which groups are significantly different from which others? Statistical tests are provided to compare group means, group medians, and group standard deviations. When comparing means, multiple range tests are used, the most popular of which is Tukey's HSD procedure. For equal size samples, significant group differences can be determined by examining the means plot and identifying those intervals that do not overlap.

Multifactor ANOVA

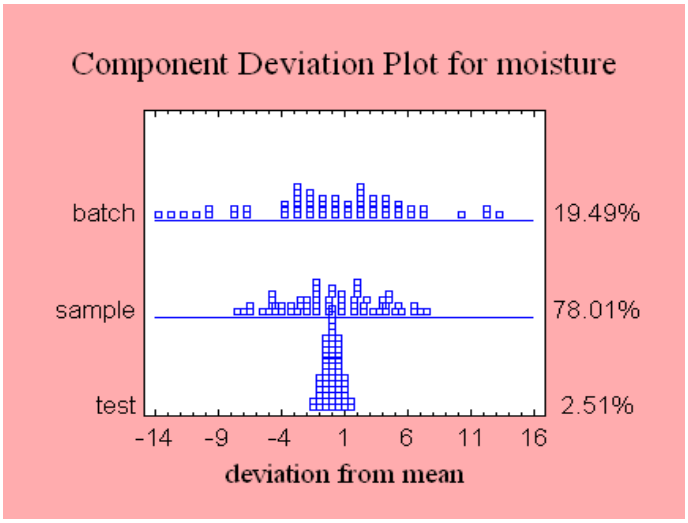


When more than one factor is present and the factors are crossed, a multifactor ANOVA is appropriate. Both main effects and interactions between the factors may be estimated. The output includes an ANOVA table and a new graphical ANOVA from the latest edition of Statistics for Experimenters by Box, Hunter and Hunter (Wiley, 2005). In a graphical ANOVA, the points are scaled so that any levels that differ by more than exhibited in the distribution of the

residuals are significantly different.

Variance Components Analysis

A *Variance Components Analysis* is most commonly used to determine the level at which variability is being introduced into a product. A typical experiment might select several batches, several samples from each batch, and then run replicates tests on each sample. The goal is to determine the relative percentages of the overall process variability that is being introduced at each level.



General Linear Model

The *General Linear Models* procedure is used whenever the above procedures are not appropriate. It can be used for models with both crossed and nested factors, models in which one or more of the variables is random rather than fixed, and when quantitative factors are to be combined with categorical ones. Designs that can be analyzed with the GLM procedure include partially nested designs, repeated measures experiments, split plots, and many others. For example, pages 536-540 of the book Design and Analysis of Experiments (sixth edition) by Douglas Montgomery (Wiley, 2005) contains an example of an experimental design with both crossed and nested factors. For that data, the GLM procedure produces several important tables, including estimates of the variance components for the random factors.

Analysis of Variance for Assembly Time

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
Model	243.7	23	10.59	4.54	0.0002
Residual	56.0	24	2.333		
Total (Corr.)	299.7	47			

Type III Sums of Squares

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-Ratio*	*P-Value*
Layout	4.083	1	4.083	0.34	0.5807
Operator(Layout)	71.92	6	11.99	2.18	0.1174
Fixture	82.79	2	41.4	7.55	0.0076
Layout*Fixture	19.04	2	9.521	1.74	0.2178
Fixture*Operator(Layout)	65.83	12	5.486	2.35	0.0360
Residual	56.0	24	2.333		
Total (corrected)	299.7	47			

Expected Mean Squares

Source	*EMS*
Layout	(6)+2.0(5)+6.0(2)+Q1
Operator(Layout)	(6)+2.0(5)+6.0(2)
Fixture	(6)+2.0(5)+Q2
Layout*Fixture	(6)+2.0(5)+Q3
Fixture*Operator(Layout)	(6)+2.0(5)
Residual	(6)

Variance Components

Source	*Estimate*
Operator(Layout)	1.083
Fixture*Operator(Layout)	1.576
Residual	2.333