MODUL PRAKTIKUM PERTEMUAN 1 - 7

KODE MATA KULIAH CPD121

MATA KULIAH PEMROSESAN DATA TERSEBAR (PDT)

OLEH

Ir. Munawar, MMSI, M.Com, Ph.D
Ir. Nizirwan Anwar, MT

FAKULTAS ILMU KOMPUTER

UNIVERSITAS ESA UNGGUL

2018

DAFTAR ISI

MODUL PRAKTIKUM

DAFTAR PUSTAKA

Ajay D. Kshemkalyani and Mukesh Singhal, (2008), Distributed Computing - Principles, Algorithms, and Systems, Cambridge University Press, ISBN-13 978-0-511-39341-9

CognosTutorial – Simple Easy Learning, (2016), https://www.tutorialspoint.com/cognos/cognos_tutorial.pdf di-akses tanggal 5 Maret 2018 di Jakarta

Couloris et. al. (2012), Distributed Systems Concepts and Design, Fifth Edition. Addison Wesley, ISBN 13: 978-0-13-214301-1

Maarten van Steen and Andrew S. Tanenbaum, (2017), Distributed Systems : Prinsiples and Paradigms. 3 th Pearson Education Inc.

P M. Tamer Özsu and Patrick Valduriez, (2011), Principles of Distributed Database Systems, Third Edition, Springer Publishing ISBN 978-1-4419-8833-1

Modul 1

Introduction to distributed data processing

1. Programming Methodologies ─ Introduction

When programs are developed to solve real-life problems like inventory management, payroll processing, student admissions, examination result processing, etc. they tend to be huge and complex. The approach to analyzing such complex problems, planning for software development and controlling the development process is called **programming methodology**.

Types of Programming Methodologies

There are many types of programming methodologies prevalent among software developers:

Procedural Programming

Problem is broken down into procedures, or blocks of code that perform one task each. All procedures taken together form the whole program. It is suitable only for small programs that have low level of complexity.

Example: For a calculator program that does addition, subtraction, multiplication, division, square root and comparison, each of these operations can be developed as separate procedures. In the main program each procedure would be invoked on the basis of user's choice.

Object-oriented Programming

Here the solution revolves around entities or objects that are part of problem. The solution deals with how to store data related to the entities, how the entities behave and how they interact with each other to give a cohesive solution.

Example: If we have to develop a payroll management system, we will have entities like employees, salary structure, leave rules, etc. around which the solution must be built.

Functional Programming

Here the problem, or the desired solution, is broken down into functional units. Each unit performs its own task and is self-sufficient. These units are then stitched together to form the complete solution.

Example: A payroll processing can have functional units like employee data maintenance, basic salary calculation, gross salary calculation, leave processing, loan repayment processing, etc.

Logical Programming

Here the problem is broken down into logical units rather than functional units. **Example**: In a school management system, users have very defined roles like class teacher, subject teacher, lab assistant, coordinator, academic in-charge, etc. So the software can be divided into units depending on user roles. Each user can have different interface, permissions, etc.

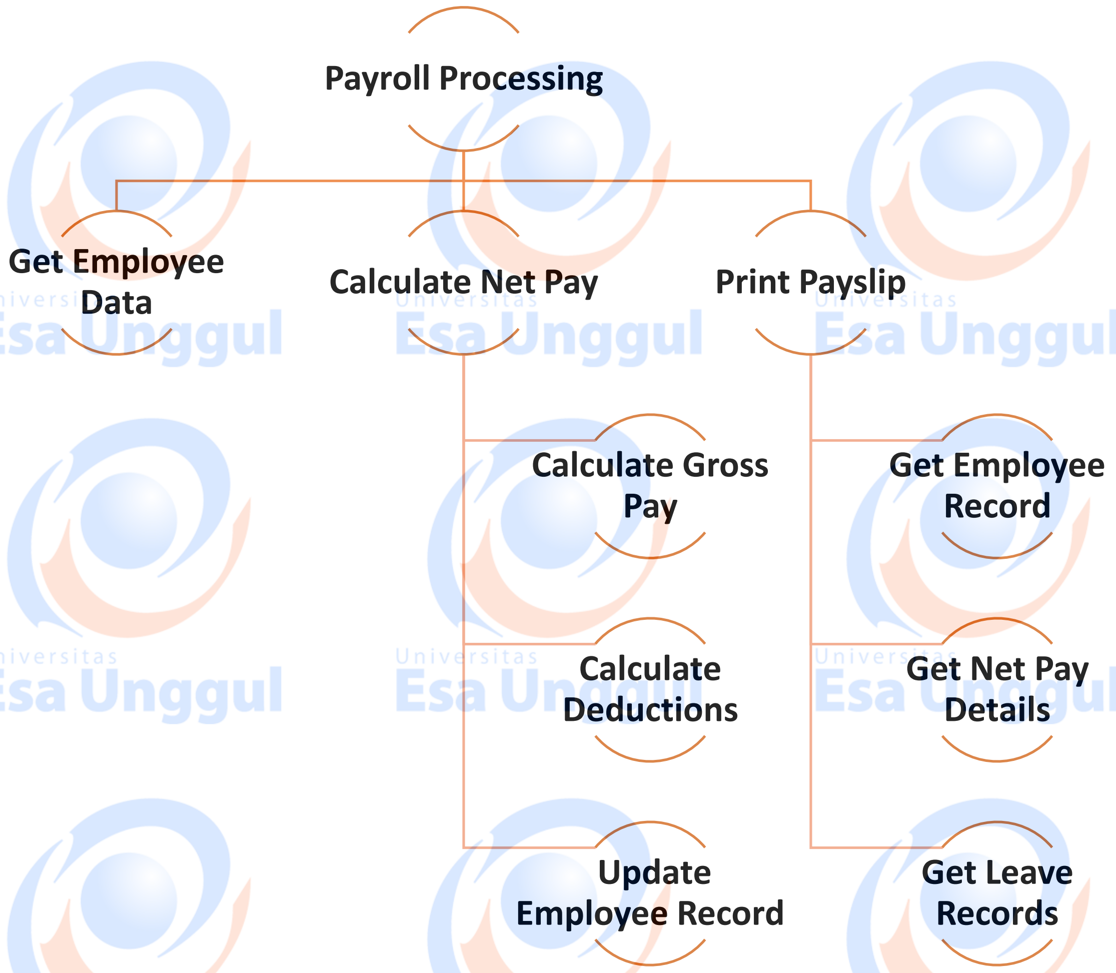
Software developers may choose one or a combination of more than one of these methodologies to develop a software. Note that in each of the methodologies discussed, problem has to be broken down into smaller units. To do this, developers use any of the following two approaches:

- Top-down approach
- Bottom-up approach

Top-down or Modular Approach

The problem is broken down into smaller units, which may be further broken down into even smaller units. Each unit is called a **module.** Each module is a self-sufficient unit that has everything necessary to perform its task.

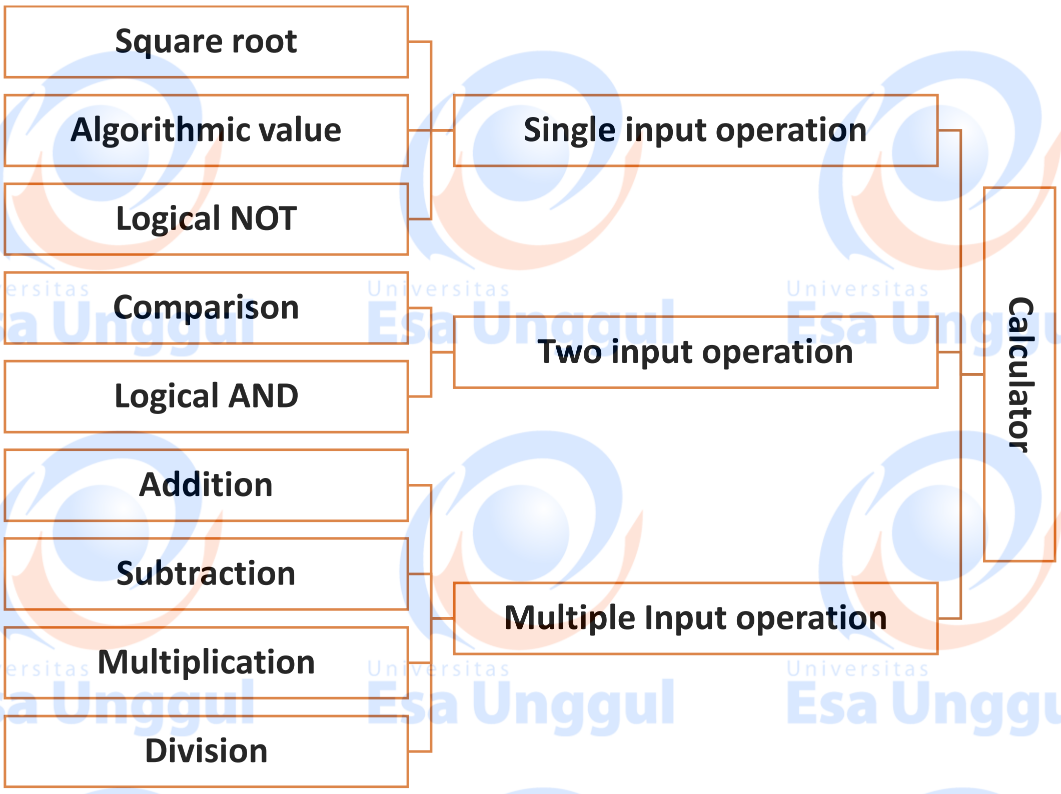
The following illustration shows an example of how you can follow modular approach to create different modules while developing a payroll processing program.



Bottom-up Approach

In bottom-up approach, system design starts with the lowest level of components, which are then interconnected to get higher level components. This process continues till a hierarchy of all system components is generated. However, in real-life scenario it is very difficult to know all lowest level components at the outset. So bottoms up approach is used only for very simple problems.

Let us look at the components of a calculator program.



2. Programming Methodologies ― Understanding the Problem

A typical software development process follows these steps:

- Requirement gathering
- Problem definition
- System design
- Implementation
- Testing
- Documentation
- Training and support
- Maintenance

The first two steps assist the team in understanding the problem, the most crucial first step towards getting a solution. Person responsible for gathering requirement, defining the problem and designing the system is called **system analyst**.

Requirement Gathering

Usually, clients or users are not able to clearly define their problems or requirements. They have a vague idea of what they want. So system developers need to gather client requirements to understand the problem that needs to be resolved, or what needs to be delivered. Detailed understanding of the problem is possible only by first understanding the business area for which the solution is being developed. Some key questions that help in understanding a business include:

- What is being done?
- How is it being done?
- What is the frequency of a task?
- What is the volume of decisions or transactions?
- What are the problems being encountered?

Some techniques that help in gathering this information are:

- Interviews
- Questionnaires
- Studying existing system documents
- Analyzing business data

System analysts needs to create clear and concise but thorough requirements document in order to identify SMART – specific, measurable, agreed upon, realistic and time-based – requirements. A failure to do so results in:

- Incomplete problem definition
- Incorrect program goals
- Re-work to deliver required outcome to client
- Increased costs
- Delayed delivery

Due to the depth of information required, requirement gathering is also known as **detailed investigation**.

Problem Definition

After gathering requirements and analyzing them, problem statement must be stated clearly. Problem definition should unambiguously state what problem or problems need to be solved. Having a clear problem statement is necessary to:

- Define project scope
- Keep the team focused
- Keep the project on track
- Validate that desired outcome was achieved at the end of project

1. XP – Introduction

This chapter gives an overview of Extreme Programming.

What is Agile?

The word 'agile' means-

- Able to move your body quickly and easily.
- Able to think quickly and clearly.

In business, 'agile' is used for describing ways of planning and doing work wherein it is understood that making changes as needed is an important part of the job. Business 'agililty' means that a company is always in a position to take account of the market changes.

Ref: Cambridge Dictionaries online.

In software development, the term 'agile' is adapted to mean 'the ability to respond to changes – changes from Requirements, Technology and People.'

Agile Manifesto

A team of software developers published the Agile Manifesto in 2001, highlighting the importance of the development team, accommodating changing requirements and customer involvement.

The Agile Manifesto states that-

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value-

- **Individuals and interactions** over processes and tools.
- **Working software** over comprehensive documentation.
- **Customer collaboration** over contract negotiation.
- **Responding to change** over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

Characteristics of Agility

Following are the characteristics of Agility-

- Agility in Agile Software Development focuses on the culture of the whole team with multi-discipline, cross-functional teams that are empowered and self-organizing.

- It fosters shared responsibility and accountability.

- Facilitates effective communication and continuous collaboration.
- The whole-team approach avoids delays and wait times.
- Frequent and continuous deliveries ensure quick feedback that in in turn enable the team align to the requirements.
- Collaboration facilitates combining different perspectives timely in implementation, defect fixes and accommodating changes.
- Progress is constant, sustainable, and predictable emphasizing transparency.

Software Engineering Trends

The following trends are observed in software engineering-

- Gather requirements before development starts. However, if the requirements are to be changed later, then following is usually noticed-
    - Resistance to the changes at a later stage of development.
    - There is a requirement of a rigorous change process that involves a change control board that may even push the changes to later releases.
    - The delivery of a product with obsolete requirements, not meeting the customer's expectations.
    - Inability to accommodate the inevitable domain changes and technology changes within the budget.
- Find and eliminate defects early in the development life cycle in order to cut the defect-fix costs.
    - Testing starts only after coding is complete and testing is considered as a tester's responsibility though the tester is not involved in development.
    - Measure and track the process itself. This becomes expensive because of-
    - Monitoring and tracking at the task level and at the resource level.
    - Defining measurements to guide the development and measuring every activity in the development.
    - Management intervention.
- Elaborate, analyze, and verify the models before development.
    - A model is supposed to be used as a framework. However, focus on the model and not on the development that is crucial will not yield the expected results.

- Coding, which is the heart of development is not given enough emphasis. The reasons being-
    - Developers, who are responsible for the production, are usually not in constant communication with the customers.
    - Coding is viewed as a translation of design and the effective implementation in code is hardly ever looped back into the design.
- Testing is considered to be the gateway to check for defects before delivery.
    - Schedule overruns of the earlier stages of development are compensated by overlooking the test requirements to ensure timely deliveries.
    - This results in cost overruns fixing defects after delivery.
    - Testers are made responsible and accountable for the product quality though they were not involved during the entire course of development.
- Limiting resources (mainly team) to accommodate budget leads to-
    - Resource over allocation.
    - Team burnout.
    - Loss in effective utilization of team competencies.
    - Attrition.

Extreme Programming – A way to handle the common shortcomings

Software Engineering involves-

- Creativity
- Learning and improving through trials and errors
- Iterations

Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.

Extreme Programming is based on the following values-

- Communication
- Simplicity
- Feedback

- Courage
- Respect

What is Extreme Programming?

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where-

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.

Embrace Change

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.

This can be achieved with-

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

Extreme Programming in a Nutshell

Extreme Programming involves-

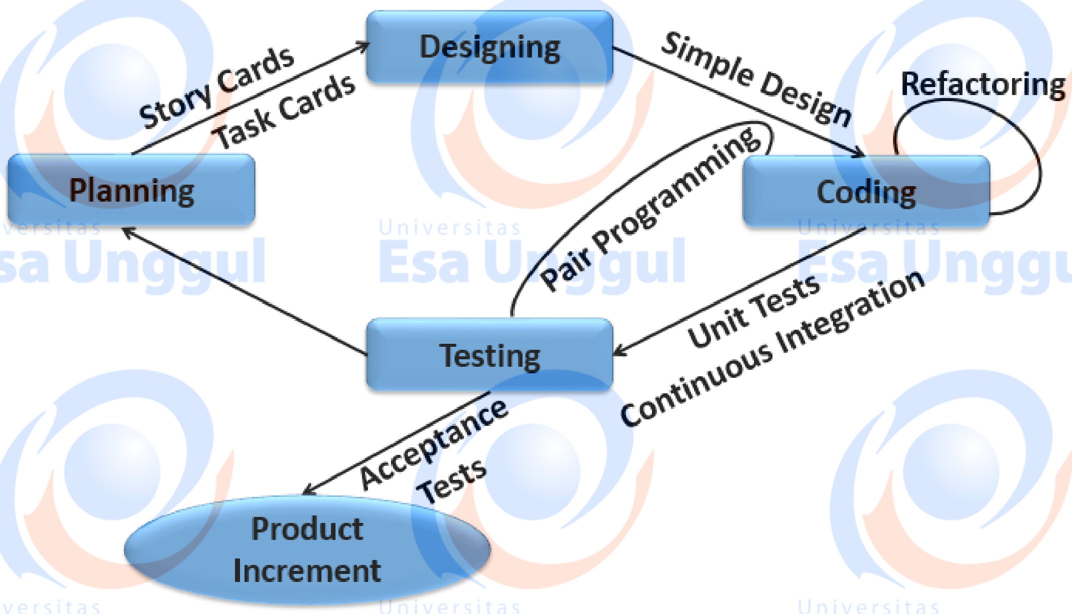
- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.

- Starting with a simple design just enough to code the features at hand and redesigning when required.

- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.

- Integrating and testing the whole system several times a day.
- Putting a minimal working system into the production quickly and upgrading it whenever required.
- Keeping the customer involved all the time and obtaining constant feedback.

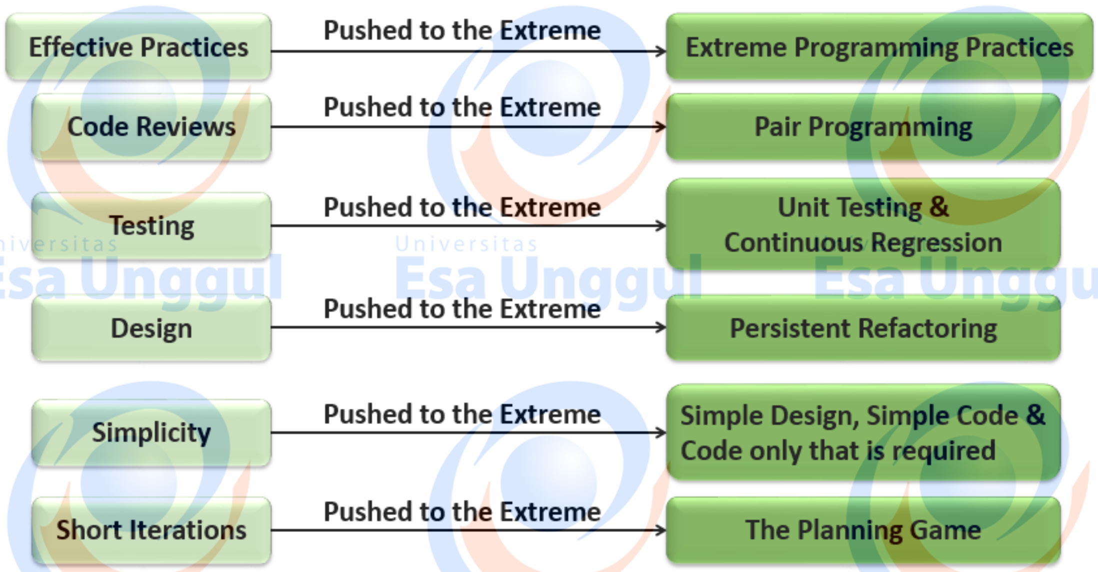
Iterating facilitates the accommodating changes as the software evolves with the changing requirements.



Why is it called "Extreme?"

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.



History of Extreme Programming

Kent Beck, Ward Cunningham and Ron Jeffries formulated extreme Programming in 1999. The other contributors are Robert Martin and Martin Fowler.

In Mid-80s, Kent Beck and Ward Cunningham initiated Pair Programming at Tektronix. In the 80s and 90s, Smalltalk Culture produced Refactoring, Continuous Integration, constant testing, and close customer involvement. This culture was later generalized to the other environments.

In the Early 90s, Core Values were developed within the Patterns Community, Hillside Group. In 1995, Kent summarized these in Smalltalk Best Practices, and in 1996, Ward summarized it in episodes.

In 1996, Kent added unit testing and metaphor at Hewitt. In 1996, Kent had taken the Chrysler C3 project, to which Ron Jeffries was added as a coach. The practices were refined on C3 and published on Wiki.

Scrum practices were incorporated and adapted as the planning game. In 1999, Kent published his book, 'Extreme Programming Explained'. In the same year, Fowler published his book, Refactoring.

Extreme Programming has been evolving since then, and the evolution continues through today.

Success in Industry

The success of projects, which follow Extreme Programming practices, is due to-

- Rapid development.
- Immediate responsiveness to the customer's changing requirements.
- Focus on low defect rates.

- System returning constant and consistent value to the customer.
- High customer satisfaction.
- Reduced costs.
- Team cohesion and employee satisfaction.

Extreme Programming Advantages

Extreme Programming solves the following problems often faced in the software development projects-

- **Slipped schedules:** Short and achievable development cycles ensure timely deliveries.
- **Cancelled projects:** Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues.
- **Costs incurred in changes:** Extensive and ongoing testing makes sure the changes do not break the existing functionality. A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected.
- **Production and post-delivery defects: Emphasis is on** the unit tests to detect and fix the defects early.
- **Misunderstanding the business and/or domain:** Making the customer a part of the team ensures constant communication and clarifications.
- **Business changes:** Changes are considered to be inevitable and are accommodated at any point of time.
- **Staff turnover:** Intensive team collaboration ensures enthusiasm and good will. Cohesion of multi-disciplines fosters the team spirit.

1. Functional Programming – Introduction

Functional programming languages are specially designed to handle symbolic computation and list processing applications. Functional programming is based on mathematical functions. Some of the popular functional programming languages include: Lisp, Python, Erlang, Haskell, Clojure, etc.

Functional programming languages are categorized into two groups, i.e.:

- **Pure Functional Languages:** These types of functional languages support only the functional paradigms. For example: Haskell.

- **Impure Functional Languages:** These types of functional languages support the functional paradigms and imperative style programming. For example: LISP.

Functional Programming – Characteristics

The most prominent characteristics of functional programming are as follows:

- Functional programming languages are designed on the concept of mathematical functions that use conditional expressions and recursion to perform computation.

- Functional programming supports **higher-order functions** and **lazy evaluation** features.

- Functional programming languages don't support flow Controls like loop statements and conditional statements like If-Else and Switch Statements. They directly use the functions and functional calls.

- Like OOP, functional programming languages support popular concepts such as Abstraction, Encapsulation, Inheritance, and Polymorphism.

Functional Programming – Advantages

Functional programming offers the following advantages:

- **Bugs-Free Code:** Functional programming does not support **state**, so there are no side-effect results and we can write error-free codes.

- **Efficient Parallel Programming:** Functional programming languages have NO Mutable state, so there are no state-change issues. One can program "Functions" to work parallel as "instructions". Such codes support easy reusability and testability.

- **Efficiency:** Functional programs consist of independent units that can run concurrently. As a result, such programs are more efficient.

- **Supports Nested Functions:** Functional programming supports Nested Functions.

- **Lazy Evaluation:** Functional programming supports Lazy Functional Constructs like Lazy Lists, Lazy Maps, etc.

As a downside, functional programming requires a large memory space. As it does not have state, you need to create new objects every time to perform actions.

Functional Programming is used in situations where we have to perform lots of different operations on the same set of data.

- Lisp is used for artificial intelligence applications like Machine learning, language processing, Modeling of speech and vision, etc.

- Embedded Lisp interpreters add programmability to some systems like Emacs.

Functional Programming vs. Object-oriented Programming

The following table highlights the major differences between functional programming and object-oriented programming:

Functional Programming	OOP
Uses Immutable data.	Uses Mutable data.
Follows Declarative Programming Model.	Follows Imperative Programming Model.
Focus is on: “What you are doing”	Focus is on “How you are doing”
Supports Parallel Programming	Not suitable for Parallel Programming
Its functions have no-side effects	Its methods can produce serious side-effects.
Flow Control is done using function calls & function calls with recursion	Flow control is done using loops and conditional statements.
It uses "Recursion" concept to iterate Collection Data.	It uses "Loop" concept to iterate Collection Data. For example: For-each loop in Java
Execution order of statements is not so important.	Execution order of statements is very important.
Supports both "Abstraction over Data" and "Abstraction over Behavior".	Supports only "Abstraction over Data".

Efficiency of a Program Code

The efficiency of a programming code is directly proportional to the algorithmic efficiency and the execution speed. Good efficiency ensures higher performance.

The factors that affect the efficiency of a program includes:

- The speed of the machine
- Compiler speed
- Operating system
- Choosing right Programming language
- The way of data in a program is organized
- Algorithm used to solve the problem

The efficiency of a programming language can be improved by performing the following tasks:

- By removing unnecessary code or the code that goes to redundant processing.
- By making use of optimal memory and nonvolatile storage
- By making the use of reusable components wherever applicable.
- By making the use of error & exception handling at all layers of program.
- By creating programming code that ensures data integrity and consistency.
- By developing the program code that's compliant with the design logic and flow.

An efficient programming code can reduce resource consumption and completion time as much as possible with minimum risk to the operating environment.

Modul 2

Distributed-System

2. OOAD – Object Model

The object model visualizes the elements in a software application in terms of objects. In this chapter, we will look into the basic concepts and terminologies of object–oriented systems.

Objects and Classes

The concepts of objects and classes are intrinsically linked with each other and form the foundation of object–oriented paradigm.

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has:

- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, object is an instance of a class.

The constituents of a class are:

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows:

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows:

- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

During instantiation, values are assigned for at least some of the attributes. If we create an object my_circle, we can assign values like x-coord : 2, y-coord : 3, and a : 4 to depict its state. Now, if the operation scale() is performed on my_circle with a scaling factor of 2, the value of the variable a will become 8. This operation brings a change in the state of my_circle, i.e., the object has exhibited certain behavior.

Encapsulation and Data Hiding

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Data Hiding

Typically, a class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access. This process of insulating an object's data is called data hiding or information hiding.

Example

In the class Circle, data hiding can be incorporated by making attributes invisible from outside the class and adding two more methods to the class for accessing class data, namely:

- setValues(), method to assign values to x-coord, y-coord, and a
- getValues(), method to retrieve values of x-coord, y-coord, and a

Here the private data of the object my_circle cannot be accessed directly by any method that is not encapsulated within the class Circle. It should instead be accessed through the methods setValues() and getValues().

Message Passing

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other using message passing. Suppose a system has two objects: obj1 and obj2. The object obj1 sends a message to object obj2, if obj1 wants obj2 to execute one of its methods.

The features of message passing are:

- Message passing between two objects is generally unidirectional.
- Message passing enables all interactions between objects.
- Message passing essentially involves invoking class methods.

- Objects in different processes can be involved in message passing.

Inheritance

Inheritance is the mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses. The subclass can inherit or derive the attributes and methods of the super-class(es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines an "is – a" relationship.

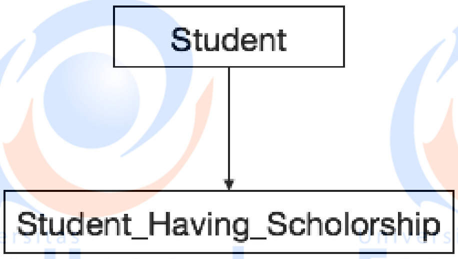
Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow "is – a" mammal.

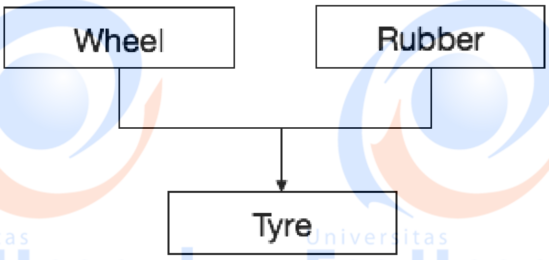
Types of Inheritance

- **Single Inheritance** : A subclass derives from a single super-class.
- **Multiple Inheritance** : A subclass derives from more than one super-classes.
- **Multilevel Inheritance** : A subclass derives from a super-class which in turn is derived from another class and so on.
- **Hierarchical Inheritance** : A class has a number of subclasses each of which may have subsequent subclasses, continuing for a number of levels, so as to form a tree structure.
- **Hybrid Inheritance** : A combination of multiple and multilevel inheritance so as to form a lattice structure.

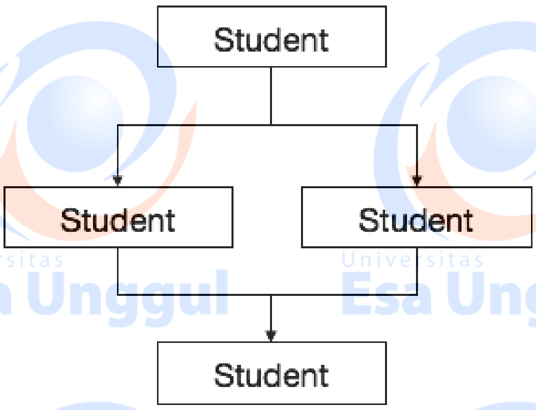
The following figure depicts the examples of different types of inheritance.



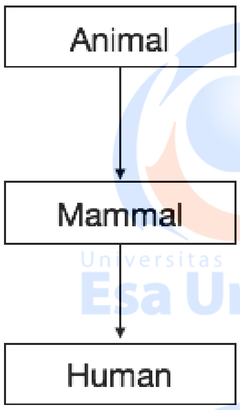
Single Inheritance



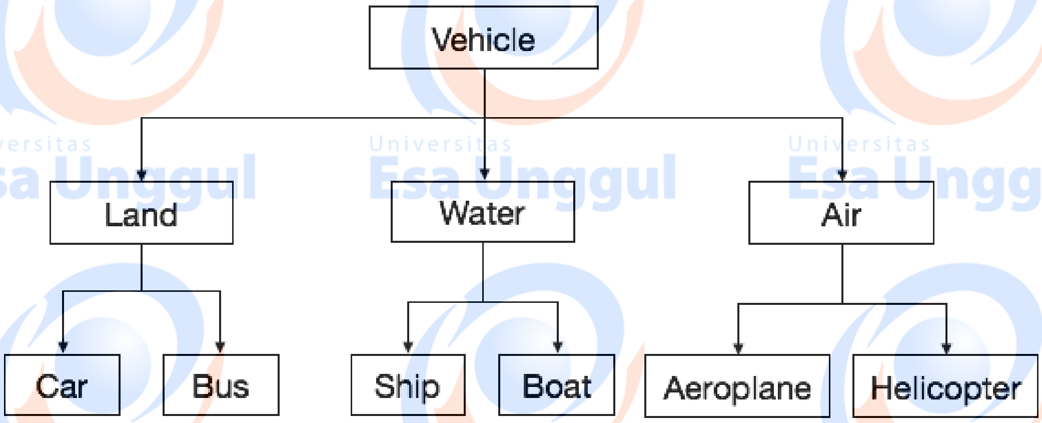
Multiple Inheritance



Hybrid Inheritance



Multi-level Inheritance



Hierarchical Inheritance

Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instance they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example

Let us consider two classes, Circle and Square, each with a method findArea(). Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating area is different for each class. When an object of class Circle invokes its findArea() method, the operation finds the area of the circle without any conflict with the findArea() method of the Square class.

Generalization and Specialization

Generalization and specialization represent a hierarchy of relationships between classes, where subclasses inherit from super-classes.

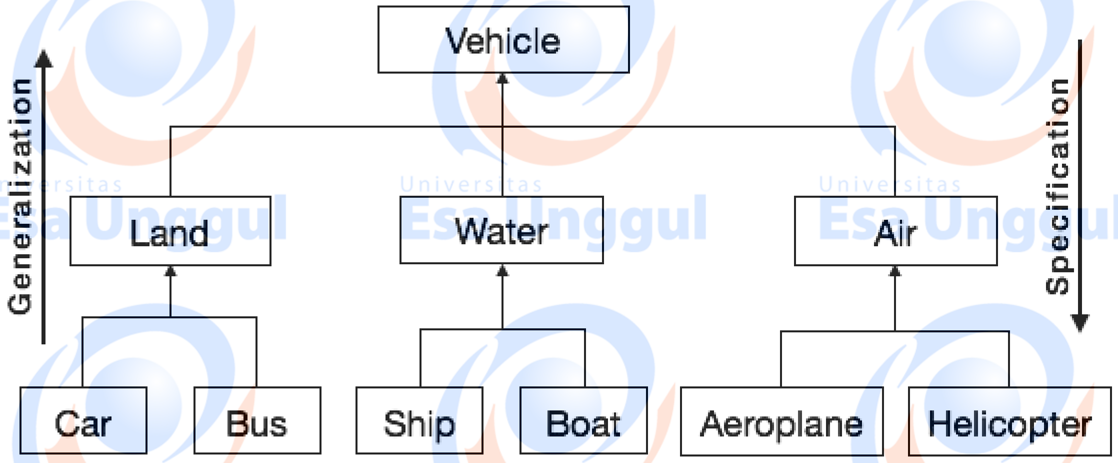
Generalization

In the generalization process, the common characteristics of classes are combined to form a class in a higher level of hierarchy, i.e., subclasses are combined to form a generalized super-class. It represents an "is – a – kind – of" relationship. For example, "car is a kind of land vehicle", or "ship is a kind of water vehicle".

Specialization

Specialization is the reverse process of generalization. Here, the distinguishing features of groups of objects are used to form specialized classes from existing classes. It can be said that the subclasses are the specialized versions of the super-class.

The following figure shows an example of generalization and specialization.



Links and Association

Link

A link represents a connection through which an object collaborates with other objects. Rumbaugh has defined it as "a physical or conceptual connection between objects". Through a link, one object may invoke the methods or navigate through another object. A link depicts the relationship between two or more objects.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association.

Degree of an Association

Degree of an association denotes the number of classes involved in a connection. Degree may be unary, binary, or ternary.

- A **unary relationship** connects objects of the same class.
- A **binary relationship** connects objects of two classes.
- A **ternary relationship** connects objects of three or more classes.

Cardinality Ratios of Associations

Cardinality of a binary association denotes the number of instances participating in an association. There are three types of cardinality ratios, namely:

- **One–to–One** : A single object of class A is associated with a single object of class B.
- **One–to–Many** : A single object of class A is associated with many objects of class B.
- **Many–to–Many** : An object of class A may be associated with many objects of class B and conversely an object of class B may be associated with many objects of class A.

Aggregation or Composition

Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a "part–of" or "has–a" relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Example

In the relationship, "a car has–a motor", car is the whole object or the aggregate, and the motor is a "part–of" the car. Aggregation may denote:

- **Physical containment** : Example, a computer is composed of monitor, CPU, mouse, keyboard, and so on.
- **Conceptual containment** : Example, shareholder has–a share.

Benefits of Object Model

Now that we have gone through the core concepts pertaining to object orientation, it would be worthwhile to note the advantages that this model has to offer.

The benefits of using the object model are:

- It helps in faster development of software.

- It is easy to maintain. Suppose a module develops an error, then a programmer can fix that particular module, while the other parts of the software are still up and running.
- It supports relatively hassle-free upgrades.
- It enables reuse of objects, designs, and functions.
- It reduces development risks, particularly in integration of complex systems.

3. OOAD – Object-Oriented System

We know that the Object-Oriented Modelling (OOM) technique visualizes things in an application by using models organized around objects. Any software development approach goes through the following stages:

- Analysis,
- Design, and
- Implementation.

In object-oriented software engineering, the software developer identifies and organizes the application in terms of object-oriented concepts, prior to their final representation in any specific programming language or software tools.

Phases in Object-Oriented Software Development

The major phases of software development using object–oriented methodology are object-oriented analysis, object-oriented design, and object-oriented implementation.

Object–Oriented Analysis

In this stage, the problem is formulated, user requirements are identified, and then a model is built based upon real–world objects. The analysis produces models on how the desired system should function and how it must be developed. The models do not include any implementation details so that it can be understood and examined by any non–technical application expert.

Object–Oriented Design

Object-oriented design includes two main stages, namely, system design and object design.

System Design

In this stage, the complete architecture of the desired system is designed. The system is conceived as a set of interacting subsystems that in turn is composed of a hierarchy of interacting objects, grouped into classes. System design is done according to both the system analysis model and the proposed system architecture. Here, the emphasis is on the objects comprising the system rather than the processes in the system.

Object Design

In this phase, a design model is developed based on both the models developed in the system analysis phase and the architecture designed in the system design phase. All the classes required are identified. The designer decides whether:

- new classes are to be created from scratch,
- any existing classes can be used in their original form, or
- new classes should be inherited from the existing classes.

The associations between the identified classes are established and the hierarchies of classes are identified. Besides, the developer designs the internal details of the classes and their associations, i.e., the data structure for each attribute and the algorithms for the operations.

Object–Oriented Implementation and Testing

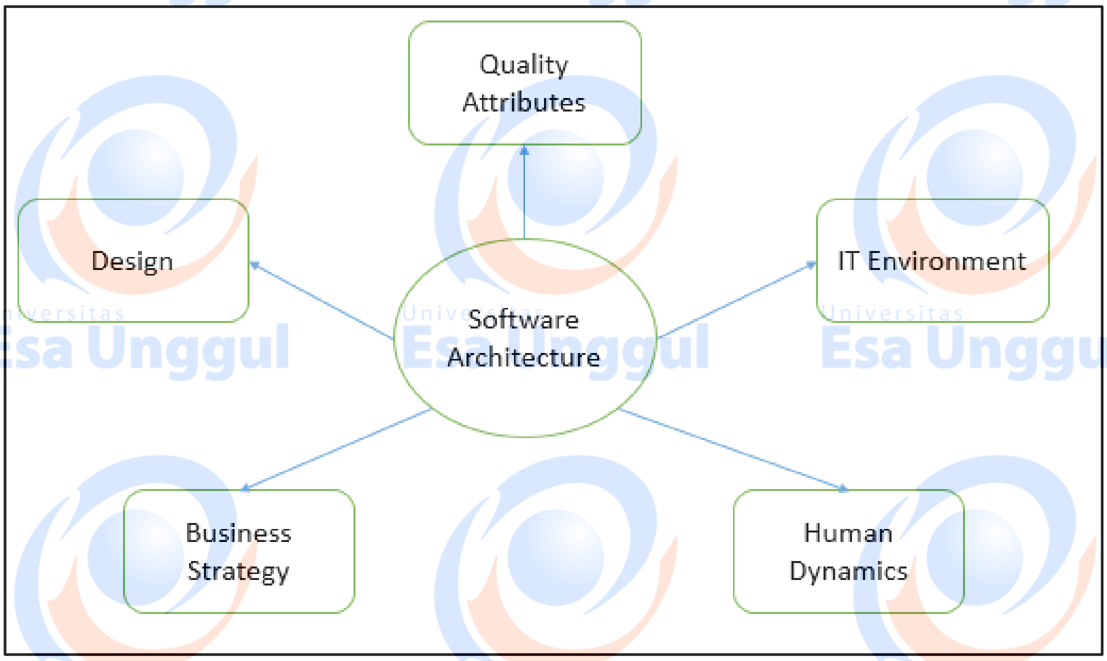
In this stage, the design model developed in the object design is translated into code in an appropriate programming language or software tool. The databases are created and the specific hardware requirements are ascertained. Once the code is in shape, it is tested using specialized techniques to identify and remove the errors in the code.

Modul 3

Architectur Database

1. Software Architecture and Design – Introduction

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design is a process that includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In **Design**, functional requirements are accomplished.

Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.

- It involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of:

    - Selection of structural elements and their interfaces by which the system is composed.

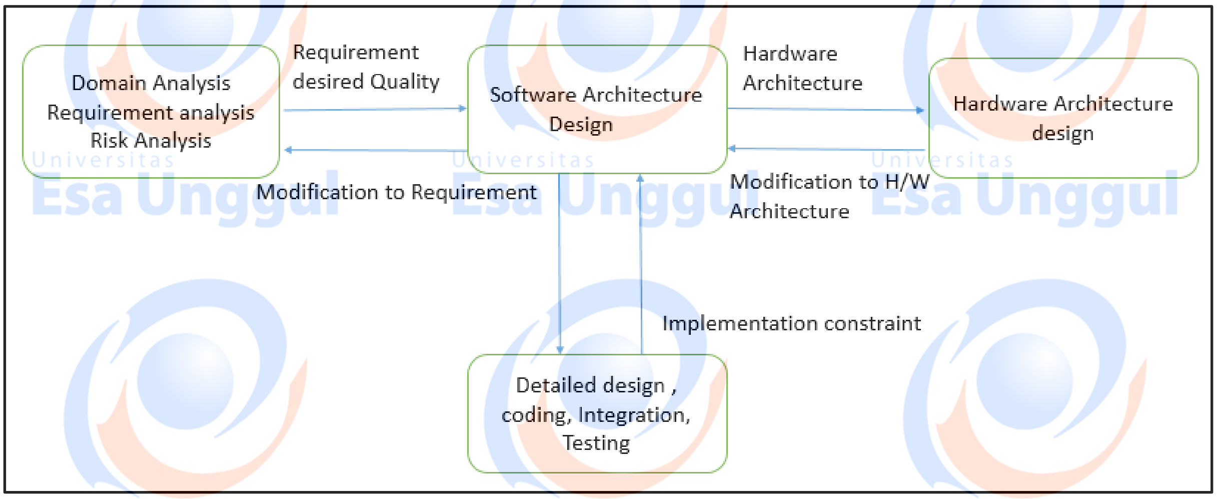
- Behavior as specified in collaborations among those elements.
- Composition of these structural and behavioral elements into large subsystem.
- Architectural decisions align with business objectives.
- Architectural styles that guide the organization.

Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows:

- To negotiate system requirements, and to set expectations with customers, marketing and management personnel.
- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.


Domain Analysis
Requirement analysis
Risk Analysis
Requirement desired Quality
Modification to Requirement
Software Architecture Design
Hardware Architecture
Modification to H/W Architecture
Hardware Architecture design
Implementation constraint
Detailed design , coding, Integration, Testing

Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements. Some of the other goals are as follows:

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.

- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations:

- Lack of tools and standardized ways to represent architecture
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development
- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the design process, design experience and evaluation of design

Role of Software Architect

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas:

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.
- Should be able to review design proposals and tradeoffs among them.

Domain Expertise

- Expert on the system being developed and plan for software evolution.
- Assist in the requirement investigation process assuring completeness and consistency.
- Coordinate the definition of domain model for the system being developed.

Technology Expertise

- Expert on available technologies that helps in the implementation of the system.
- Coordinate the selection of programming language, framework, platforms, databases, etc.

Methodological Expertise

- Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).
- Choose the appropriate approaches for development that helps the entire team.

Hidden Role of Software Architect

- Facilitates the technical work among team members and reinforcing the trust relationship in the team.
- Information specialist who shares knowledge and has vast experience.
- Protect the team members from external forces that would distract them and bring less value to the project.

Deliverables of the Architect

- A clear, complete, consistent, and achievable set of functional goals
- A functional description of the system, with at least two layers of decomposition
- A concept for the system
- A design in the form of the system, with at least two layers of decomposition
- A notion of the timing, operator attributes, and the implementation and operation plans
- A document or process which ensures functional decomposition is followed, and the form of interfaces is controlled

Quality Attributes

Quality is a measure of excellence or the state of being free from deficiencies or defects. Quality attributes are system properties that are separate from the functionality of the system.

Implementing quality attributes makes it is easier to differentiate a good system from a bad one. Attributes are overall factors that affect runtime behavior, system design, and user experience. They can be classified as follows:

Static Quality Attributes

Reflect the structure of system and organization, directly related to architecture, design, source code. They are invisible to end-user, but affect the development and maintenance cost, e.g.: modularity, testability, maintainability, etc.

Dynamic Quality Attributes

Reflect the behavior of the system during its execution. They are directly related to system's architecture, design, source code and also the configuration, deployment parameters, environment, and platform.

They are visible to the end-user and exist at runtime, e.g.: throughput, robustness, scalability, etc.

Quality Scenarios

Quality scenarios specify how to prevent a fault from becoming a failure. They can be divided into six parts based on their attribute specifications:

- **Source** An internal or external entity such as people, hardware, software, or physical infrastructure that generates the stimulus.
- **Stimulus** A condition that needs to be considered when it arrives on a system.
- **Environment** The stimulus occurs within certain conditions.
- **Artifact** A whole system or some part of it such as processors, communication channel, persistent storage, processes etc.
- **Response** An activity undertaken after the arrival of stimulus such as detect faults, recover from fault, disable event source etc.
- **Response measure** Should measure the occurred responses so that the requirements can be tested.

Common Quality Attributes

The following table lists the common quality attributes a software architecture must have.

Category	Quality Attribute	Description
Design Qualities	Conceptual Integrity	Defines the consistency and coherence of the overall design. This includes the way components or modules are designed
	Maintainability	Ability of the system to undergo changes with a degree of ease.
	Reusability	Defines the capability for components and subsystems to be suitable for use in other applications
	Interoperability	Ability of a system or different systems to operate successfully by communicating and exchanging

Run-time Qualities		information with other external systems written and run by external parties.
	Manageability	Defines how easy it is for system administrators to manage the application
	Reliability	Ability of a system to remain operational over time
	Scalability	Ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
	Security	Capability of a system to prevent malicious or accidental actions outside of the designed usages
	Performance	Indication of the responsiveness of a system to execute any action within a given time interval.
	Availability	Defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period.
System Qualities	Supportability	Ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
	Testability	Measure of how easy it is to create test criteria for the system and its components
User Qualities	Usability	Defines how well the application meets the requirements of the user and consumer by being intuitive
Architecture Quality	Correctness	Accountability for satisfying all the requirements of the system.
Non-runtime Quality	Portability	Ability of the system to run under different computing environment.
	Integrality	Ability to make separately developed components of the system work correctly together.
	Modifiability	Ease with which each software system can accommodate changes to its software.
Business quality attributes	Cost and schedule	Cost of the system with respect to time to market, expected project lifetime & utilization of legacy
	Marketability	Use of system with respect to market competition.

2. Software Architecture and Design – Key Principles

Software architecture is described as the organization of a system, where the system represents a collection of components that accomplish a specific set of functions.

Architectural Style

The **architectural style**, also called as **architectural pattern,** is a set of principles which shapes an application. It defines an abstract framework for a family of system in terms of the pattern of structural organization. The architectural style

- Provides a lexicon of components and connectors with rules on how they can be combined.
- Improves partitioning and allows the reuse of design by giving solutions to frequently occurring problems.
- Describes a particular way to configure a collection of components (a module with well-defined interfaces, reusable, replaceable) and connectors (communication link between modules).

The software that is built for computer-based systems exhibit one of many architectural styles. Each style describes a system category that encompasses:

- A set of component types which perform a required function by the system
- A set of connectors (subroutine call, remote procedure call, data stream, socket) that enable communication, coordination, and cooperation among components
- Semantic constraints which define how components can be integrated to form the system
- A topological layout of the components indicating their runtime interrelationships

Common Architectural Design

The following table lists architectural styles that can be organized by their key focus area.

Category	Architectural Design	Description
	Message bus	Prescribes use of a software system that can receive and send messages using one or more communication channels

Communication	Service–Oriented Architecture (SOA)	Defines the applications that expose and consume functionality as a service using contracts and messages.
Deployment	Client/server	Separate the system into two applications, where the client makes requests to the server
	3-tier or N-tier	Separates the functionality into separate segments with each segment being a tier located on a physically separate computer.
Domain	Domain Driven Design	Focused on modeling a business domain and defining business objects based on entities within the business domain.
Structure	Component Based	Breakdown the application design into reusable functional or logical components that expose well-defined communication interfaces.
	Layered	Divide the concerns of the application into stacked groups (layers).
	Object oriented	Based on the division of responsibilities of an application or system into objects, each containing the data and the behavior relevant to the object.

Types of Architecture

There are four types of architecture from the viewpoint of an enterprise and collectively, these architectures are referred to as **enterprise architecture**.

- **Business architecture**: Defines the strategy of business, governance, organization, and key business processes within an enterprise and focuses on the analysis and design of business processes.
- **Application (software) architecture**: Serves as the blueprint for individual application systems, their interactions, and their relationships to the business processes of the organization
- **Information architecture**: Defines the logical and physical data assets and data management resources.
- **Information technology (IT) architecture**: Defines the hardware and software building blocks that make up the overall information system of the organization.

Architecture Design Process

The architecture design process focuses on the decomposition of a system into components and their interactions to satisfy functional and nonfunctional requirements. The key inputs to software architecture design are:

- The requirements produced by the analysis tasks
- The hardware architecture (the software architect in turn provides requirements to the system architect, who configures the hardware architecture)

The result or output of the architecture design process is an **architectural description**. The basic architecture design process is composed of the following steps:

Understand the Problem

- This is the most crucial step because it affects the quality of the design that follows.
- Without a clear understanding of the problem, it is not possible to create an effective solution.
- Many software projects and products are considered failures because they did not actually solve a valid business problem or have a recognizable return on investment (ROI).

Identify Design Elements and their Relationships

- In this phase, build a baseline for defining the boundaries and context of the system.
- Decomposition of the system into its main components based on functional requirements. The decomposition can be modeled using a design structure matrix (DSM), which shows the dependencies between design elements without specifying the granularity of the elements.
- In this step, the first validation of the architecture is done by describing a number of system instances and this step is referred as functionality based architectural design.

Evaluate the Architecture Design

- Each quality attribute is given an estimate so in order to gather qualitative measures or quantitative data, the design is evaluated.
- It involves evaluating the architecture for conformance to architectural quality attributes requirements.
- If all estimated quality attributes are as per the required standard, the architectural design process is finished.
- If not, the third phase of software architecture design is entered: architecture transformation. If the observed quality attribute does not meet its requirements, then a new design must be created.

Modul 4

Communication and Network

1. Data Warehouse – Overview

A Data Warehouse consists of data from **multiple heterogeneous data sources** and is used for analytical reporting and decision making. Data Warehouse is a central place where data is stored from different data sources and applications.

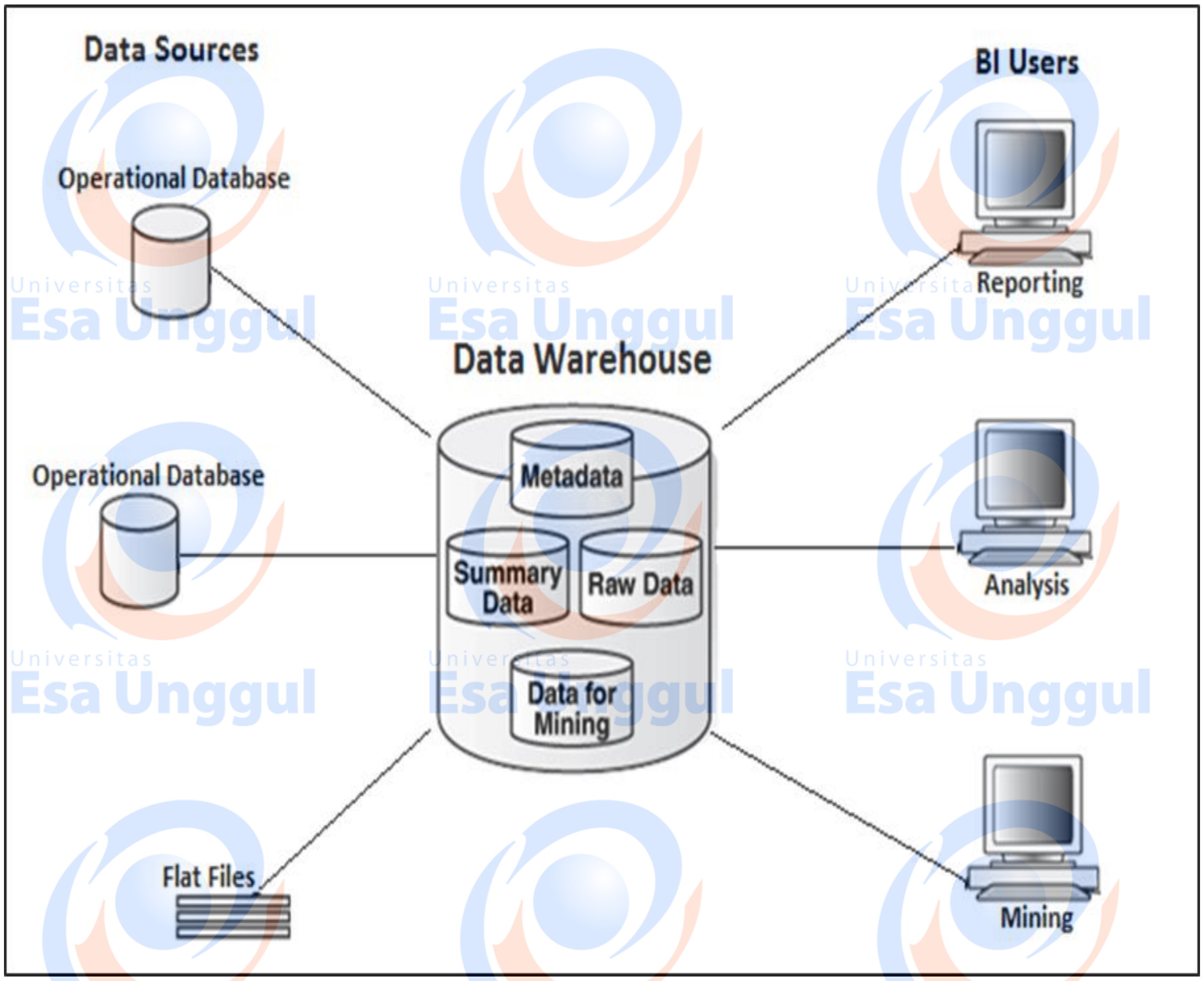
The term Data Warehouse was first invented by Bill Inmom in 1990. A Data Warehouse is always kept separate from an Operational Database.

The data in a DW system is loaded from operational transaction systems like –

- Sales
- Marketing
- HR
- SCM, etc.

It may pass through operational data store or other transformations before it is loaded to the DW system for information processing.

A Data Warehouse is used for reporting and analyzing of information and stores both historical and current data. The data in DW system is used for Analytical reporting, which is later used by Business Analysts, Sales Managers or Knowledge workers for decision-making.



In the above image, you can see that the data is coming from **multiple heterogeneous data** sources to a Data Warehouse. Common data sources for a data warehouse includes:

- Operational databases
- SAP and non-SAP Applications
- Flat Files (xls, csv, txt files)

Data in data warehouse is accessed by BI (Business Intelligence) users for Analytical Reporting, Data Mining and Analysis. This is used for decision making by Business Users, Sales Manager, Analysts to define future strategy.

Features of a Data Warehouse

It is a central data repository where data is stored from one or more heterogeneous data sources. A DW system stores both current and historical data. Normally a DW system stores 5-10 years of historical data. A DW system is always kept separate from an operational transaction system.

The data in a DW system is used for different types of analytical reporting range from Quarterly to Annual comparison.

Data Warehouse Vs Operational Database

The differences between a Data Warehouse and Operational Database are as follows –

- An **Operational System** is designed for known workloads and transactions like updating a user record, searching a record, etc. However, Data Warehouse transactions are more complex and present a general form of data.
-
- An **Operational System** contains the current data of an organization and Data warehouse normally contains the historical data.
-
- An **Operational Database** supports parallel processing of multiple transactions. Concurrency control and recovery mechanisms are required to maintain consistency of the database.
-
- An **Operational Database** query allows to read and modify operations (insert, delete and Update) while an OLAP query needs only read-only access of stored data (Select statement).

Architecture of Data Warehouse

Data Warehousing involves data cleaning, data integration, and data consolidations. A Data Warehouse has a 3-layer architecture –

Data Source Layer

It defines how the data comes to a Data Warehouse. It involves various data sources and operational transaction systems, flat files, applications, etc.

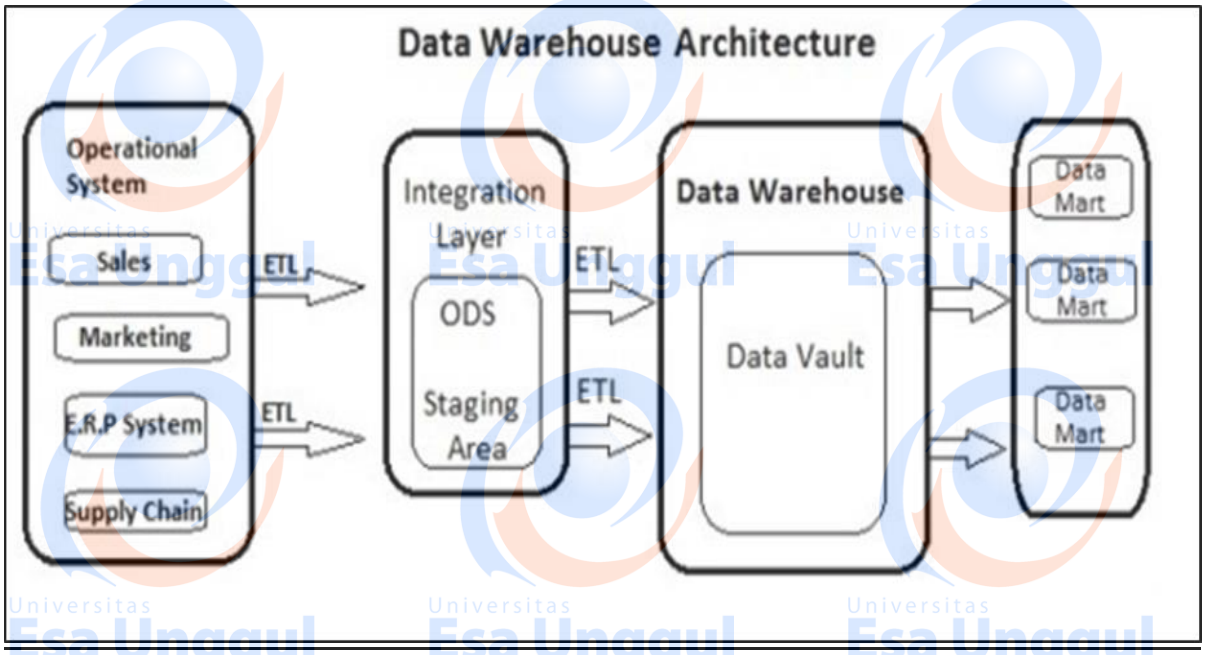
Integration Layer

It consists of Operational Data Store and Staging area. Staging area is used to perform data cleansing, data transformation and loading data from different sources to a data warehouse. As multiple data sources are available for extraction at different time zones, staging area is used to store the data and later to apply transformations on data.

Presentation Layer

This is used to perform BI reporting by end users. The data in a DW system is accessed by BI users and used for reporting and analysis.

The following illustration shows the common architecture of a Data Warehouse System.



Characteristics of a Data Warehouse

The following are the key characteristics of a Data Warehouse:

- **Subject Oriented:** In a DW system, the data is categorized and stored by a business subject rather than by application like equity plans, shares, loans, etc.
-
- **Integrated:** Data from multiple data sources are integrated in a Data Warehouse.
-
- **Non Volatile:** Data in data warehouse is non-volatile. It means when data is loaded in DW system, it is not altered.
-
- **Time Variant:** A DW system contains historical data as compared to Transactional system which contains only current data. In a Data warehouse you can see data for 3 months, 6 months, 1 year, 5 years, etc.

OLTP vs OLAP

Firstly, OLTP stands for **Online Transaction Processing,** while OLAP stands for **Online Analytical Processing**

In an OLTP system, there are a large number of short online transactions such as INSERT, UPDATE, and DELETE.

Whereas, in an OLTP system, an effective measure is the processing time of short transactions and is very less. It controls data integrity in multi-access environments. For an OLTP system, the number of transactions per second measures the effectiveness. An OLTP Data Warehouse

System contains current and detailed data and is maintained in the schemas in the entity model (3NF).

For Example:

A Day-to-Day transaction system in a retail store, where the customer records are inserted, updated and deleted on a daily basis. It provides faster query processing. OLTP databases contain detailed and current data. The schema used to store OLTP database is the Entity model.

In an OLAP system, there are lesser number of transactions as compared to a transactional system. The queries executed are complex in nature and involves data aggregations.

What is an Aggregation?

We save tables with aggregated data like yearly (1 row), quarterly (4 rows), monthly (12 rows) or so, if someone has to do a year to year comparison, only one row will be processed. However, in an un-aggregated table it will compare all the rows. This is called Aggregation.

There are various Aggregation functions that can be used in an OLAP system like Sum, Avg, Max, Min, etc.

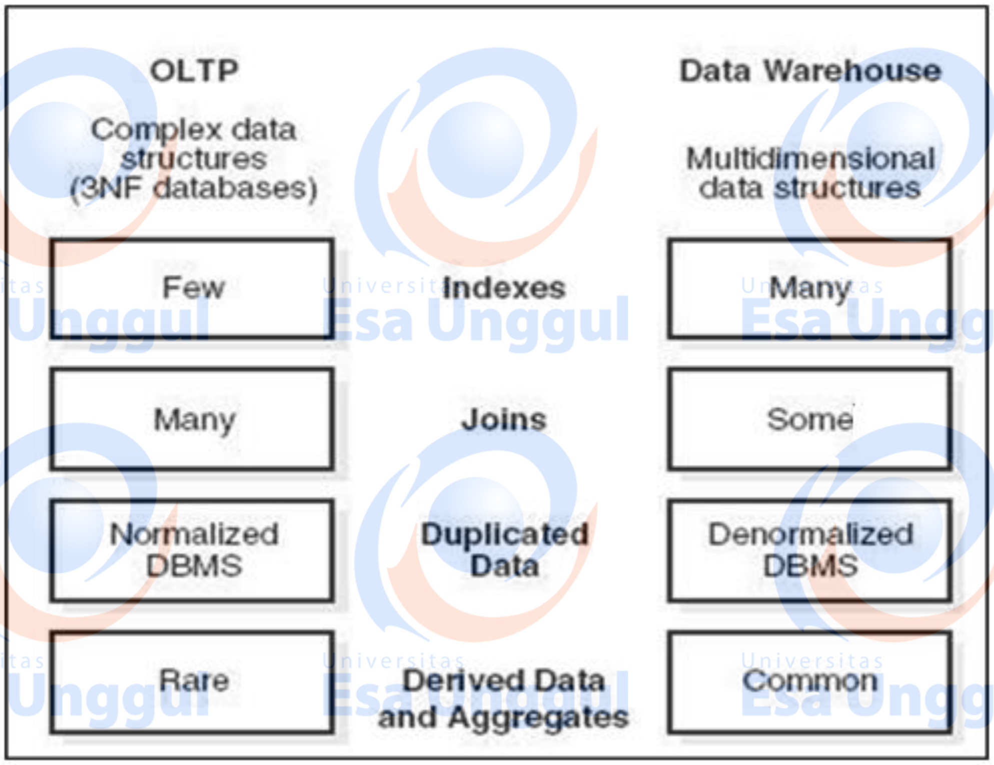
For Example:

```
SELECT Avg(salary)
FROM employee
WHERE title = 'Programmer';
```

Key Differences

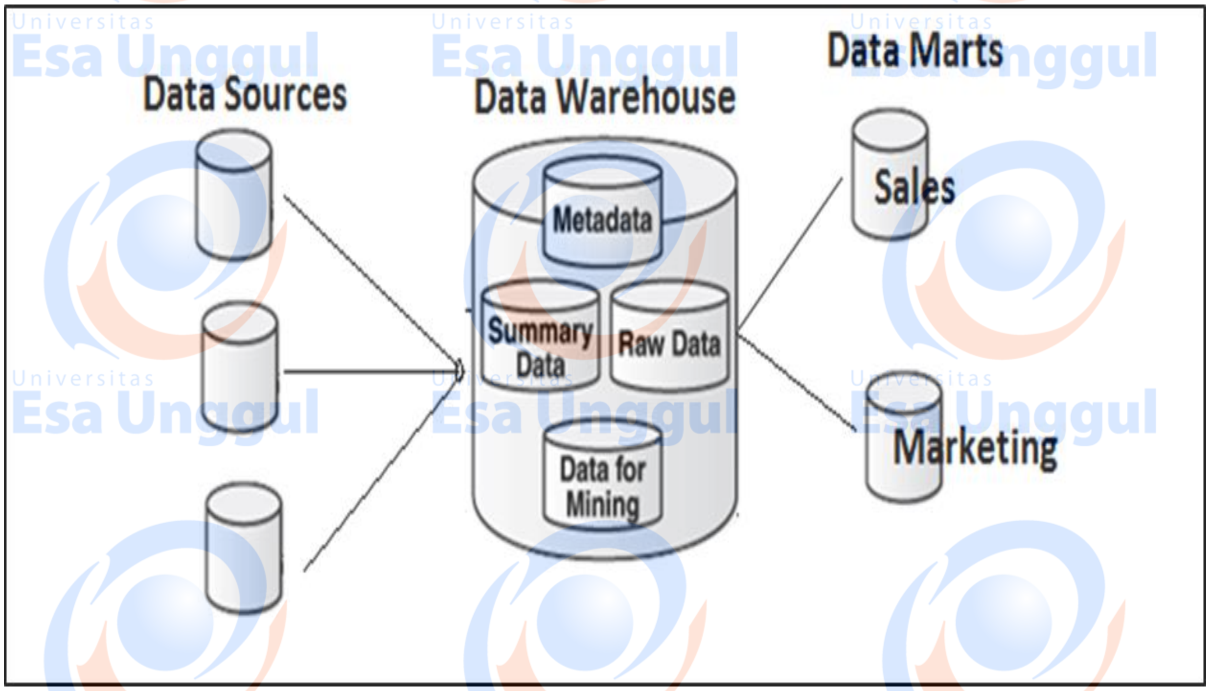
These are the major differences between an OLAP and an OLTP system.

- **Indexes:** An OLTP system has only few indexes while in an OLAP system there are many indexes for performance optimization.
-
- **Joins:** In an OLTP system, large number of joins and data are normalized. However, in an OLAP system there are less joins and are de-normalized.
-
- **Aggregation:** In an OLTP system, data is not aggregated while in an OLAP database more aggregations are used.
-
- **Normalization:** An OLTP system contains normalized data however data is not normalized in an OLAP system.



Data Mart Vs Data Warehouse

Data mart focuses on a single functional area and represents the simplest form of a Data Warehouse. Consider a Data Warehouse that contains data for Sales, Marketing, HR, and Finance. A Data mart focuses on a single functional area like Sales or Marketing.



In the above image, you can see the difference between a Data Warehouse and a data mart.

Fact vs Dimension Table

A fact table represents the measures on which analysis is performed. It also contains foreign keys for the dimension keys.

For example: Every sale is a fact.

Cust Id	Prod Id	Time Id	Qty Sold
1110	25	2	125
1210	28	4	252

The Dimension table represents the characteristics of a dimension. A Customer dimension can have Customer_Name, Phone_No, Sex, etc.

Cust_Id	Cust_Name	Phone	Sex
1110	Sally	1113334444	F
1210	Adam	2225556666	M

4. DWH – Delivery Process

A data warehouse is never static; it evolves as the business expands. As the business evolves, its requirements keep changing and therefore a data warehouse must be designed to ride with these changes. Hence a data warehouse system needs to be flexible.

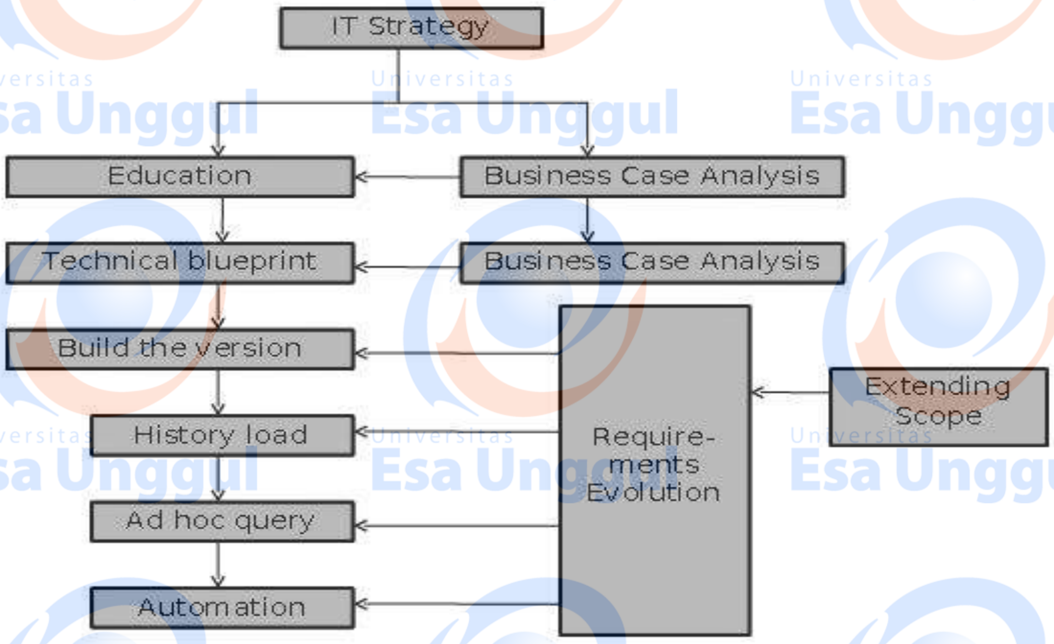
Ideally there should be a delivery process to deliver a data warehouse. However data warehouse projects normally suffer from various issues that make it difficult to complete tasks and deliverables in the strict and ordered fashion demanded by the waterfall method. Most of the times, the requirements are not understood completely. The architectures, designs, and build components can be completed only after gathering and studying all the requirements.

Delivery Method

The delivery method is a variant of the joint application development approach adopted for the delivery of a data warehouse. We have staged the data warehouse delivery process to minimize risks. The approach that we will discuss here does not reduce the overall delivery time-scales but ensures the business benefits are delivered incrementally through the development process.

Note: The delivery process is broken into phases to reduce the project and delivery risk.

The following diagram explains the stages in the delivery process:



IT Strategy

Data warehouses are strategic investments that require a business process to generate benefits. IT Strategy is required to procure and retain funding for the project.

Business Case

The objective of business case is to estimate business benefits that should be derived from using a data warehouse. These benefits may not be quantifiable but the projected benefits need to be clearly stated. If a data warehouse does not have a clear business case, then the business tends to suffer from credibility problems at some stage during the delivery process. Therefore in data warehouse projects, we need to understand the business case for investment.

Education and Prototyping

Organizations experiment with the concept of data analysis and educate themselves on the value of having a data warehouse before settling for a solution. This is addressed by prototyping. It helps in understanding the feasibility and benefits of a data warehouse. The prototyping activity on a small scale can promote educational process as long as:

- The prototype addresses a defined technical objective.
- The prototype can be thrown away after the feasibility concept has been shown.
- The activity addresses a small subset of eventual data content of the data warehouse.
- The activity timescale is non-critical.

The following points are to be kept in mind to produce an early release and deliver business benefits.

- Identify the architecture that is capable of evolving.
- Focus on business requirements and technical blueprint phases.
- Limit the scope of the first build phase to the minimum that delivers business benefits.
- Understand the short-term and medium-term requirements of the data warehouse.

Business Requirements

To provide quality deliverables, we should make sure the overall requirements are understood. If we understand the business requirements for both short-term and medium-term, then we can design a solution to fulfil short-term requirements. The short-term solution can then be grown to a full solution.

The following aspects are determined in this stage:

- The business rule to be applied on data.
- The logical model for information within the data warehouse.
- The query profiles for the immediate requirement.
- The source systems that provide this data.

Technical Blueprint

This phase needs to deliver an overall architecture satisfying the long-term requirements. This phase also delivers the components that must be implemented on a short-term basis to derive any business benefit. A blueprint identifies the following:

- The overall system architecture.
- The data retention policy.
- The backup and recovery strategy.
- The server and data mart architecture.
- The capacity plan for hardware and infrastructure.
- The components of database design.

Building the Version

In this stage, the first production deliverable is produced. This production deliverable is the smallest component of a data warehouse. This smallest component adds business benefit.

History Load

This is the phase where the remainder of the required history is loaded into the data warehouse. In this phase, we do not add any new entities, but additional physical tables would probably be created to store increased data volumes.

Let us take an example. Suppose the build version phase has delivered a retail sales analysis data warehouse with 2 months' worth of history. This information will allow the user to analyze only the recent trends and address the short-term issues. The user in this case cannot identify annual and seasonal trends. To help him do so, last 2 years' sales history could be loaded from the archive. Now the 40GB data is extended to 400GB.

Note: The backup and recovery procedures may become complex, therefore it is recommended to perform this activity within a separate phase.

Ad hoc Query

In this phase, we configure an ad hoc query tool that is used to operate a data warehouse. These tools can generate the database query.

Note: It is recommended not to use these access tools when the database is being substantially modified.

Automation

In this phase, operational management processes are fully automated. These would include:

- Transforming the data into a form suitable for analysis.

- Monitoring query profiles and determining appropriate aggregations to maintain system performance.
- Extracting and loading data from different source systems.
- Generating aggregations from predefined definitions within the data warehouse.
- Backing up, restoring, and archiving the data.

Extending Scope

In this phase, the data warehouse is extended to address a new set of business requirements. The scope can be extended in two ways:

- By loading additional data into the data warehouse.
- By introducing new data marts using the existing information.

Note: This phase should be performed separately, since it involves substantial efforts and complexity.

Requirements Evolution

From the perspective of delivery process, the requirements are always changeable. They are not static. The delivery process must support this and allow these changes to be reflected within the system.

This issue is addressed by designing the data warehouse around the use of data within business processes, as opposed to the data requirements of existing queries.

The architecture is designed to change and grow to match the business needs, the process operates as a pseudo-application development process, where the new requirements are continually fed into the development activities and the partial deliverables are produced. These partial deliverables are fed back to the users and then reworked ensuring that the overall system is continually updated to meet the business needs.

5. DWH – System Processes

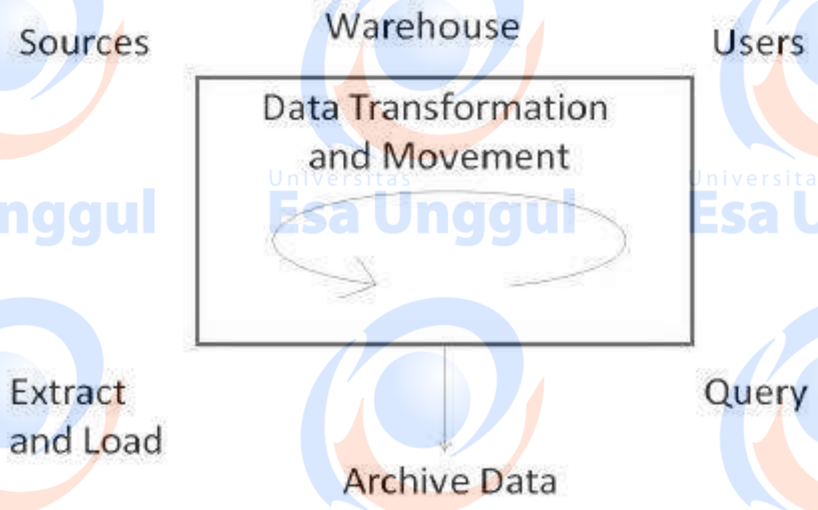
We have a fixed number of operations to be applied on the operational databases and we have well-defined techniques such as **use normalized data**, **keep table small**, etc. These techniques are suitable for delivering a solution. But in case of decision-support systems, we do not know what query and operation needs to be executed in future. Therefore techniques applied on operational databases are not suitable for data warehouses.

In this chapter, we will discuss how to build data warehousing solutions on top open-system technologies like Unix and relational databases.

Process Flow in Data Warehouse

There are four major processes that contribute to a data warehouse:

- Extract and load the data.
- Cleaning and transforming the data.
- Backup and archive the data.
- Managing queries and directing them to the appropriate data sources.



Extract and Load Process

Data extraction takes data from the source systems. Data load takes the extracted data and loads it into the data warehouse.

Note: Before loading the data into the data warehouse, the information extracted from the external sources must be reconstructed.

Controlling the Process

Controlling the process involves determining when to start data extraction and the consistency check on data. Controlling process ensures that the tools, the logic modules, and the programs are executed in correct sequence and at correct time.

When to Initiate Extract

Data needs to be in a consistent state when it is extracted, i.e., the data warehouse should represent a single, consistent version of the information to the user.

For example, in a customer profiling data warehouse in telecommunication sector, it is illogical to merge the list of customers at 8 pm on Wednesday from a customer database with the customer subscription events up to 8 pm on Tuesday. This would mean that we are finding the customers for whom there are no associated subscriptions.

Loading the Data

After extracting the data, it is loaded into a temporary data store where it is cleaned up and made consistent.

Note: Consistency checks are executed only when all the data sources have been loaded into the temporary data store.

Clean and Transform Process

Once the data is extracted and loaded into the temporary data store, it is time to perform Cleaning and Transforming. Here is the list of steps involved in Cleaning and Transforming:

- Clean and transform the loaded data into a structure
- Partition the data
- Aggregation

Clean and Transform the Loaded Data into a Structure

Cleaning and transforming the loaded data helps speed up the queries. It can be done by making the data consistent:

- within itself.
- with other data within the same data source.
- with the data in other source systems.
- with the existing data present in the warehouse.

Transforming involves converting the source data into a structure. Structuring the data increases the query performance and decreases the operational cost. The data contained in a data warehouse must be transformed to support performance requirements and control the ongoing operational costs.

Partition the Data

It will optimize the hardware performance and simplify the management of data warehouse. Here we partition each fact table into multiple separate partitions.

Aggregation

Aggregation is required to speed up common queries. Aggregation relies on the fact that most common queries will analyze a subset or an aggregation of the detailed data.

Backup and Archive the Data

In order to recover the data in the event of data loss, software failure, or hardware failure, it is necessary to keep regular backups. Archiving involves removing the old data from the system in a format that allows it to be quickly restored whenever required.

For example, in a retail sales analysis data warehouse, it may be required to keep data for 3 years with the latest 6 months data being kept online. In such as scenario, there is often a requirement to be able to do month-on-month comparisons for this year and last year. In this case, we require some data to be restored from the archive.

Query Management Process

This process performs the following functions:

- manages the queries.
- helps speed up the execution time of queris.
- directs the queries to their most effective data sources.
- ensures that all the system sources are used in the most effective way.
- monitors actual query profiles.

The information generated in this process is used by the warehouse management process to determine which aggregations to generate. This process does not generally operate during the regular load of information into data warehouse.

Modul 5

Process Database

7. DWH – OLAP

Online Analytical Processing Server (OLAP) is based on the multidimensional data model. It allows managers and analysts to get an insight of the information through fast, consistent, and interactive access to information. This chapter covers the types of OLAP, operations on OLAP, difference between OLAP, and statistical databases and OLTP.

Types of OLAP Servers

We have four types of OLAP servers:

- Relational OLAP (ROLAP)
- Multidimensional OLAP (MOLAP)
- Hybrid OLAP (HOLAP)
- Specialized SQL Servers

Relational OLAP

ROLAP servers are placed between relational back-end server and client front-end tools. To store and manage warehouse data, ROLAP uses relational or extended-relational DBMS.

ROLAP includes the following:

- Implementation of aggregation navigation logic.
- Optimization for each DBMS back-end.
- Additional tools and services.

Multidimensional OLAP

MOLAP uses array-based multidimensional storage engines for multidimensional views of data. With multidimensional data stores, the storage utilization may be low if the dataset is sparse. Therefore, many MOLAP servers use two levels of data storage representation to handle dense and sparse datasets.

Hybrid OLAP

Hybrid OLAP is a combination of both ROLAP and MOLAP. It offers higher scalability of ROLAP and faster computation of MOLAP. HOLAP servers allow to store large data volumes of detailed information. The aggregations are stored separately in MOLAP store.

Specialized SQL Servers

Specialized SQL servers provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

OLAP Operations

Since OLAP servers are based on multidimensional view of data, we will discuss OLAP operations in multidimensional data.

Here is the list of OLAP operations:

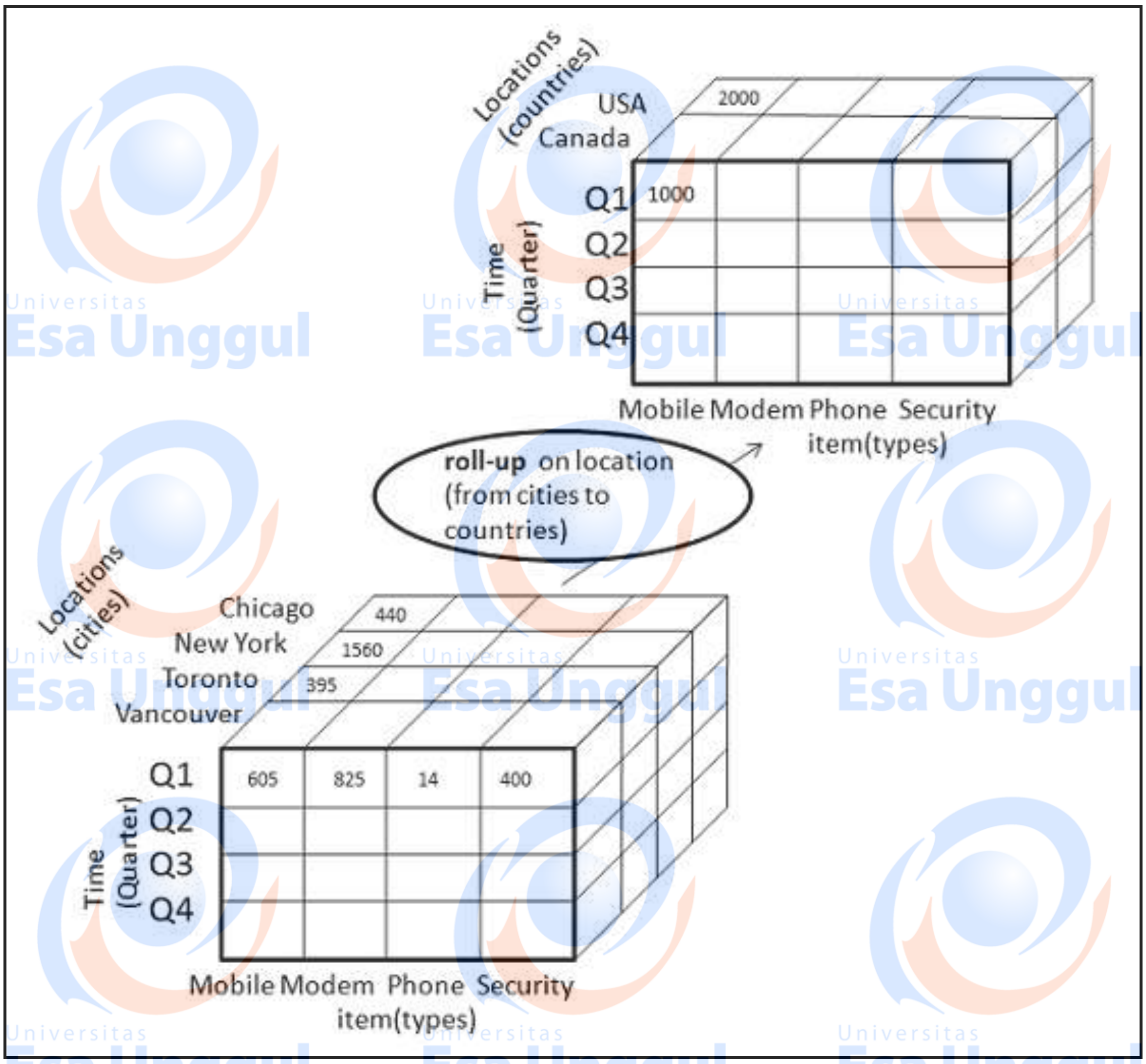
- Roll-up
- Drill-down
- Slice and dice
- Pivot (rotate)

Roll-up

Roll-up performs aggregation on a data cube in any of the following ways:

- By climbing up a concept hierarchy for a dimension
- By dimension reduction

The following diagram illustrates how roll-up works.



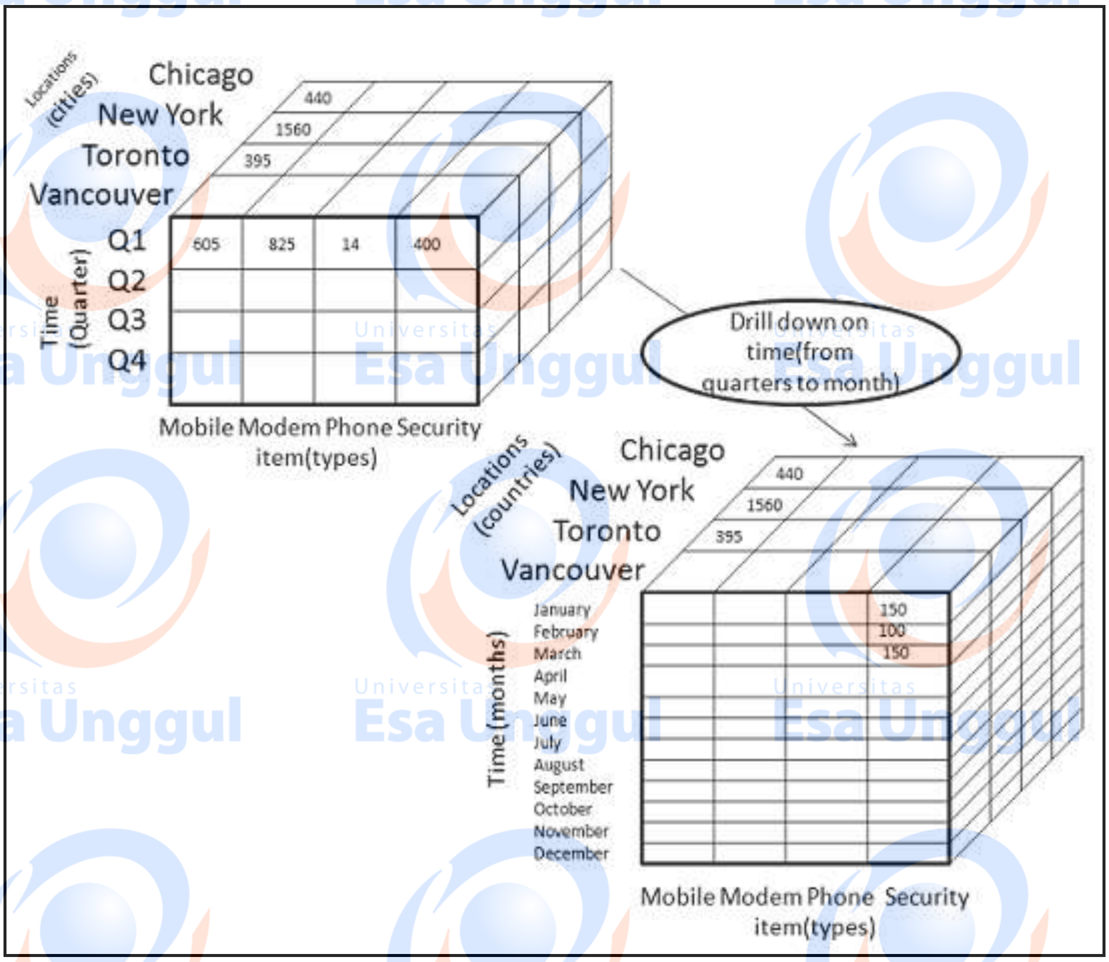
- Roll-up is performed by climbing up a concept hierarchy for the dimension location.
- Initially the concept hierarchy was "street < city < province < country".
- On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
- The data is grouped into cities rather than countries.
- When roll-up is performed, one or more dimensions from the data cube are removed.

Drill-down

Drill-down is the reverse operation of roll-up. It is performed by either of the following ways:

- By stepping down a concept hierarchy for a dimension
- By introducing a new dimension

The following diagram illustrates how drill-down works.

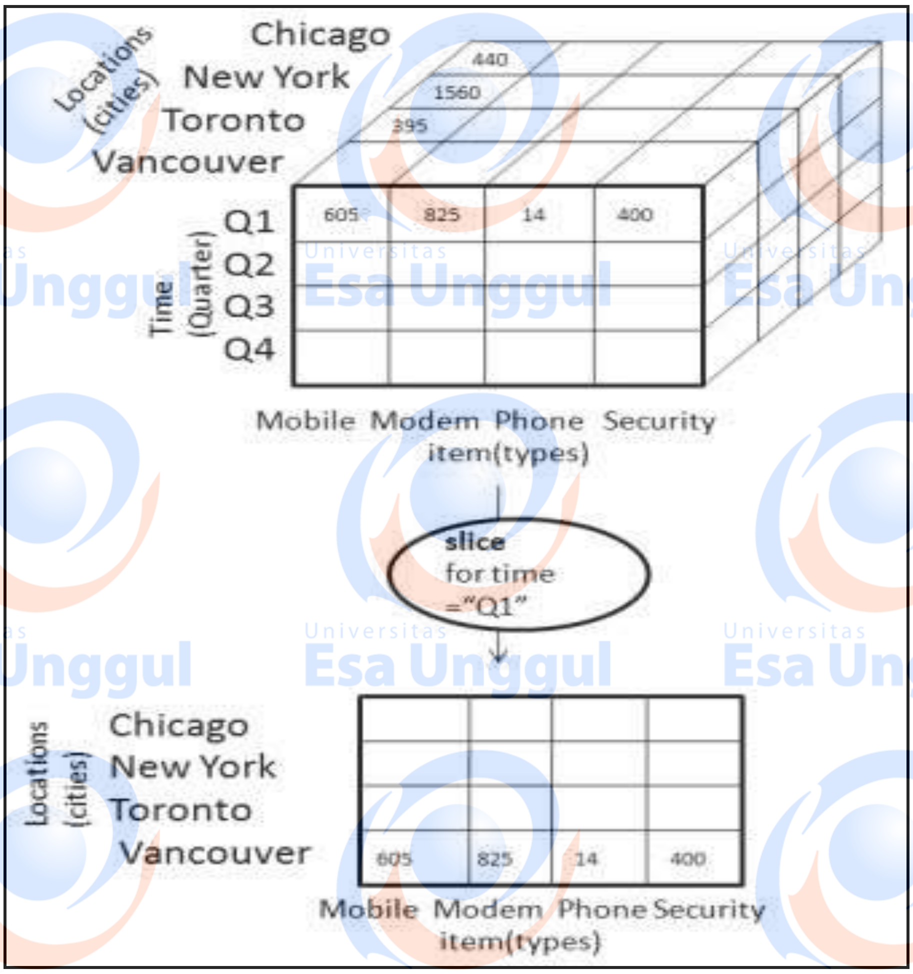


- Drill-down is performed by stepping down a concept hierarchy for the dimension time.
- Initially the concept hierarchy was "day < month < quarter < year."

- On drilling down, the time dimension is descended from the level of quarter to the level of month.
- When drill-down is performed, one or more dimensions from the data cube are added.
- It navigates the data from less detailed data to highly detailed data.

Slice

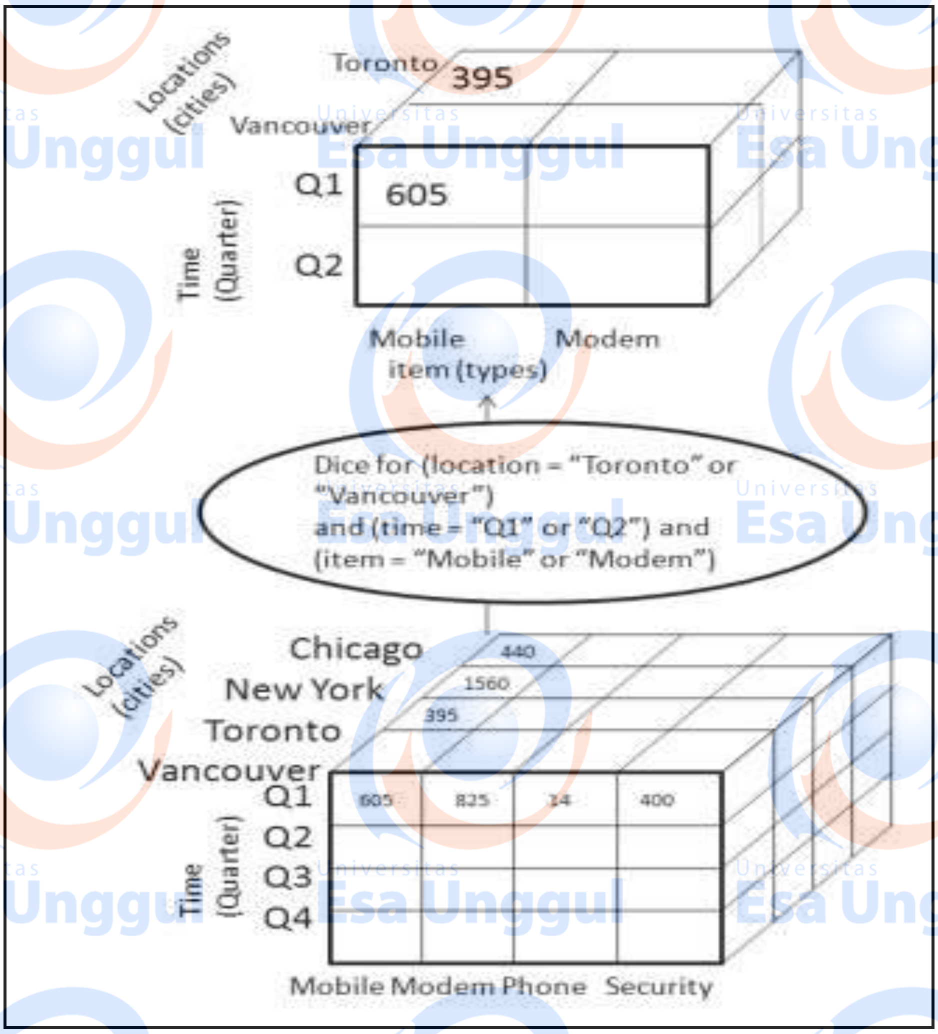
The slice operation selects one particular dimension from a given cube and provides a new sub-cube. Consider the following diagram that shows how slice works.



- Here Slice is performed for the dimension "time" using the criterion time = "Q1".
- It will form a new sub-cube by selecting one or more dimensions.

Dice

Dice selects two or more dimensions from a given cube and provides a new sub-cube. Consider the following diagram that shows the dice operation.

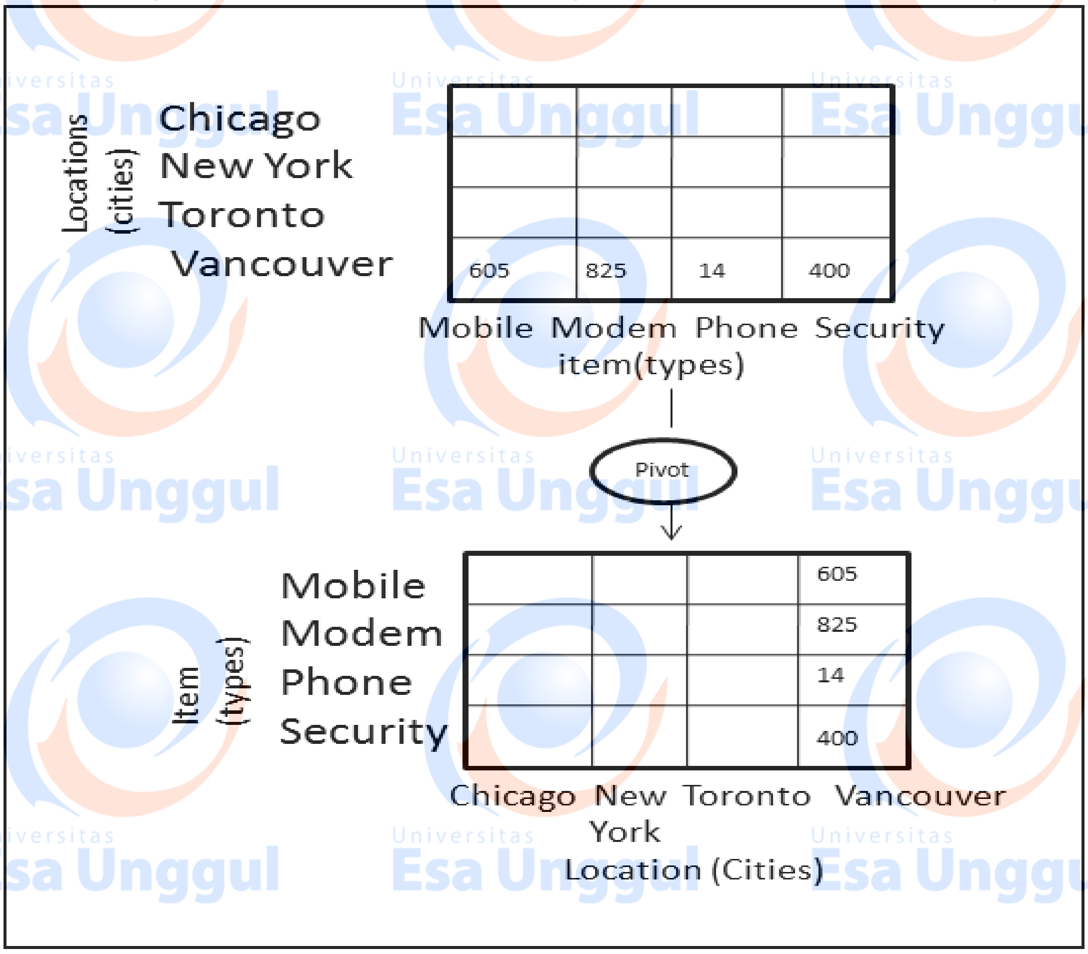


The dice operation on the cube based on the following selection criteria involves three dimensions.

- (location = "Toronto" or "Vancouver")
- (time = "Q1" or "Q2")
- (item =" Mobile" or "Modem")

Pivot

The pivot operation is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data. Consider the following diagram that shows the pivot operation.



OLAP vs OLTP

Data Warehouse (OLAP)	Operational Database (OLTP)
Involves historical processing of information.	Involves day-to-day processing.
OLAP systems are used by knowledge workers such as executives, managers, and analysts.	OLTP systems are used by clerks, DBAs, or database professionals.
Useful in analyzing the business.	Useful in running the business.
It focuses on Information out.	It focuses on Data in.

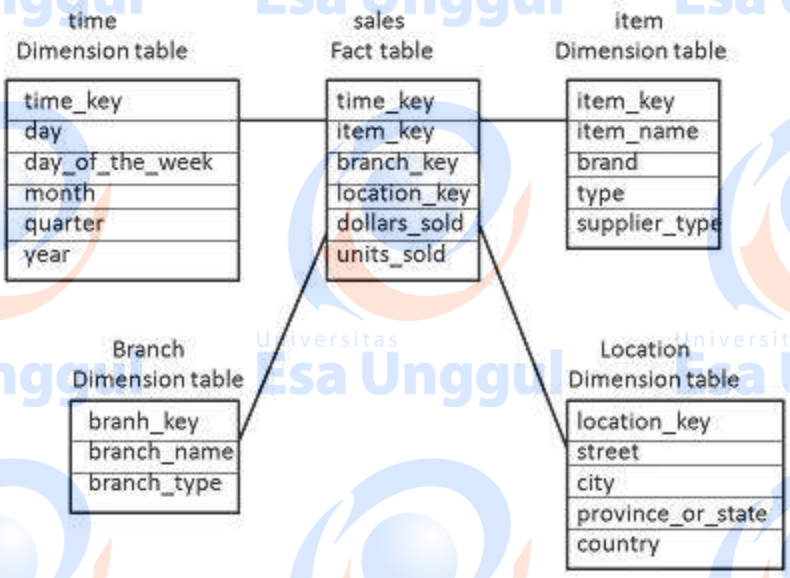
Based on Star Schema, Snowflake Schema, and Fact Constellation Schema.	Based on Entity Relationship Model.
Contains historical data.	Contains current data.
Provides summarized and consolidated data.	Provides primitive and highly detailed data.
Provides summarized and multidimensional view of data.	Provides detailed and flat relational view of data.
Number or users is in hundreds.	Number of users is in thousands.
Number of records accessed is in millions.	Number of records accessed is in tens.
Database size is from 100 GB to 1 TB.	Database size is from 100 MB to 1 GB.
Highly flexible.	Provides high performance.

10. DWH – Schemas

Schema is a logical description of the entire database. It includes the name and description of records of all record types including all associated data-items and aggregates. Much like a database, a data warehouse also requires to maintain a schema. A database uses relational model, while a data warehouse uses Star, Snowflake, and Fact Constellation schema. In this chapter, we will discuss the schemas used in a data warehouse.

Star Schema

- Each dimension in a star schema is represented with only one-dimension table.
- This dimension table contains the set of attributes.
- The following diagram shows the sales data of a company with respect to the four dimensions, namely time, item, branch, and location.

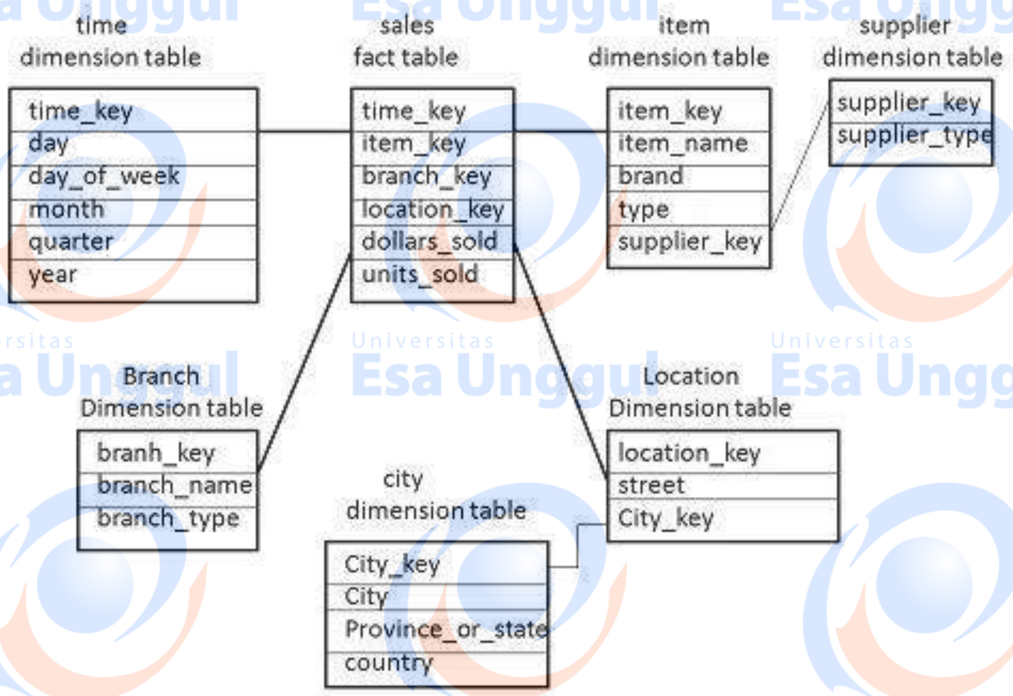


- There is a fact table at the center. It contains the keys to each of four dimensions.
- The fact table also contains the attributes, namely dollars sold and units sold.

Note: Each dimension has only one dimension table and each table holds a set of attributes. For example, the location dimension table contains the attribute set {location_key, street, city, province_or_state,country}. This constraint may cause data redundancy. For example, "Vancouver" and "Victoria" both the cities are in the Canadian province of British Columbia. The entries for such cities may cause data redundancy along the attributes province_or_state and country.

Snowflake Schema

- Some dimension tables in the Snowflake schema are normalized.
- The normalization splits up the data into additional tables.
- Unlike Star schema, the dimensions table in a snowflake schema are normalized. For example, the item dimension table in star schema is normalized and split into two dimension tables, namely item and supplier table.

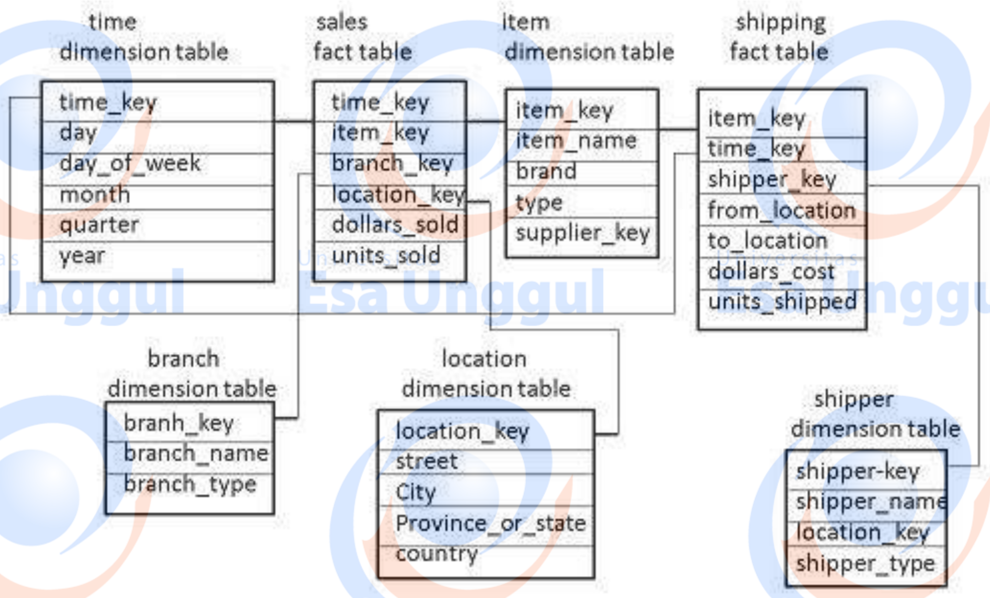


- Now the item dimension table contains the attributes item_key, item_name, type, brand, and supplier-key.
- The supplier key is linked to the supplier dimension table. The supplier dimension table contains the attributes supplier_key and supplier_type.

Note: Due to normalization in the Snowflake schema, the redundancy is reduced and therefore, it becomes easy to maintain and save the storage space.

Fact Constellation Schema

- A fact constellation has multiple fact tables. It is also known as galaxy schema.
- The following diagram shows two fact tables, namely sales and shipping.



- The sales fact table is same as that in the star schema.
- The shipping fact table has the five dimensions, namely item_key, time_key, shipper_key, from_location, to_location.
- The shipping fact table also contains two measures, namely dollars sold and units sold.
- It is also possible to share dimension tables between fact tables. For example, time, item, and location dimension tables are shared between the sales and shipping fact table.

Schema Definition

Multidimensional schema is defined using Data Mining Query Language (DMQL). The two primitives, cube definition and dimension definition, can be used for defining the data warehouses and data marts.

Syntax for Cube Definition

```
define cube < cube_name > [ < dimension-list > }: < measure_list >
```

Syntax for Dimension Definition

```
define dimension < dimension_name > as ( < attribute_or_dimension_list > )
```

Star Schema Definition

The star schema that we have discussed can be defined using Data Mining Query Language (DMQL) as follows:

```
define cube sales star [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)
define dimension item as (item key, item name, brand, type, supplier type)
define dimension branch as (branch key, branch name, branch type)
define dimension location as (location key, street, city, province or state,
country)
```

Snowflake Schema Definition

Snowflake schema can be defined using DMQL as follows:

```
define cube sales snowflake [time, item, branch, location]:

dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)
define dimension item as (item key, item name, brand, type, supplier
(supplier key, supplier type))
define dimension branch as (branch key, branch name, branch type)
define dimension location as (location key, street, city
(city key, city, province or state, country))
```

Fact Constellation Schema Definition

Fact constellation schema can be defined using DMQL as follows:

```
define cube sales [time, item, branch, location]:
dollars sold = sum(sales in dollars), units sold = count(*)

define dimension time as (time key, day, day of week, month, quarter, year)
define dimension item as (item key, item name, brand, type, supplier type)
define dimension branch as (branch key, branch name, branch type)
define dimension location as (location key, street, city, province or state,
country)
```

```
define cube shipping [time, item, shipper, from location, to location]:

dollars cost = sum(cost in dollars), units shipped = count(*)

define dimension time as time in cube sales
define dimension item as item in cube sales
define dimension shipper as (shipper key, shipper name, location as
location in cube sales, shipper type)
define dimension from location as location in cube sales
define dimension to location as location in cube sales
```

11. DWH – Partitioning Strategy

Partitioning is done to enhance performance and facilitate easy management of data. Partitioning also helps in balancing the various requirements of the system. It optimizes the hardware performance and simplifies the management of data warehouse by partitioning each fact table into multiple separate partitions. In this chapter, we will discuss different partitioning strategies.

Why is it Necessary to Partition?

Partitioning is important for the following reasons:

- For easy management,
- To assist backup/recovery,
- To enhance performance.

For Easy Management

The fact table in a data warehouse can grow up to hundreds of gigabytes in size. This huge size of fact table is very hard to manage as a single entity. Therefore it needs partitioning.

To Assist Backup / Recovery

If we do not partition the fact table, then we have to load the complete fact table with all the data. Partitioning allows us to load only as much data as is required on a regular basis. It reduces the time to load and also enhances the performance of the system.

Note: To cut down on the backup size, all partitions other than the current partition can be marked as read-only. We can then put these partitions into a state where they cannot be modified. Then they can be backed up. It means only the current partition is to be backed up.

To Enhance Performance

By partitioning the fact table into sets of data, the query procedures can be enhanced. Query performance is enhanced because now the query scans only those partitions that are relevant. It does not have to scan the whole data.

Horizontal Partitioning

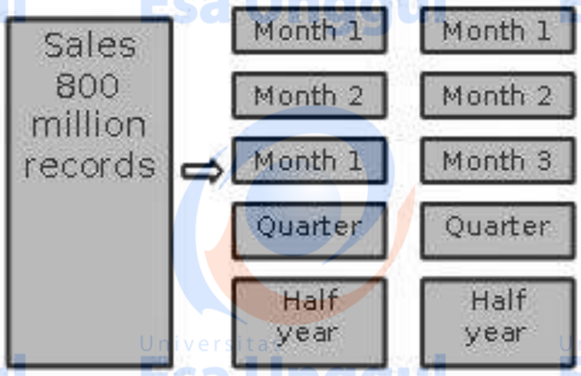
There are various ways in which a fact table can be partitioned. In horizontal partitioning, we have to keep in mind the requirements for manageability of the data warehouse.

Partition by Time into Equal Segments

In this partitioning strategy, the fact table is partitioned on the basis of time period. Here each time period represents a significant retention period within the business. For example, if the user queries for **month to date data** then it is appropriate to partition the data into monthly segments. We can reuse the partitioned tables by removing the data in them.

Partition by Time into Different-sized Segments

This kind of partition is done where the aged data is accessed infrequently. It is implemented as a set of small partitions for relatively current data, larger partition for inactive data.



Points to Note

- The detailed information remains available online.
- The number of physical tables is kept relatively small, which reduces the operating cost.
- This technique is suitable where a mix of data dipping recent history and data mining through entire history is required.
- This technique is not useful where the partitioning profile changes on a regular basis, because repartitioning will increase the operation cost of data warehouse.

Partition on a Different Dimension

The fact table can also be partitioned on the basis of dimensions other than time such as product group, region, supplier, or any other dimension. Let's have an example.

Suppose a market function has been structured into distinct regional departments like on a **state by state** basis. If each region wants to query on information captured within its region, it would prove to be more effective to partition the fact table into regional partitions. This will cause the queries to speed up because it does not require to scan information that is not relevant.

Points to Note

- The query does not have to scan irrelevant data which speeds up the query process.
- This technique is not appropriate where the dimensions are unlikely to change in future. So, it is worth determining that the dimension does not change in future.
- If the dimension changes, then the entire fact table would have to be repartitioned.

Note: We recommend to perform the partition only on the basis of time dimension, unless you are certain that the suggested dimension grouping will not change within the life of the data warehouse.

Partition by Size of Table

When there are no clear basis for partitioning the fact table on any dimension, then we should **partition the fact table on the basis of their size.** We can set the predetermined size as a critical point. When the table exceeds the predetermined size, a new table partition is created.

Points to Note

- This partitioning is complex to manage.
- It requires metadata to identify what data is stored in each partition.

Partitioning Dimensions

If a dimension contains large number of entries, then it is required to partition the dimension. Here we have to check the size of a dimension.

Consider a large design that changes over time. If we need to store all the variations in order to apply comparisons, that dimension may be very large. This would definitely affect the response time.

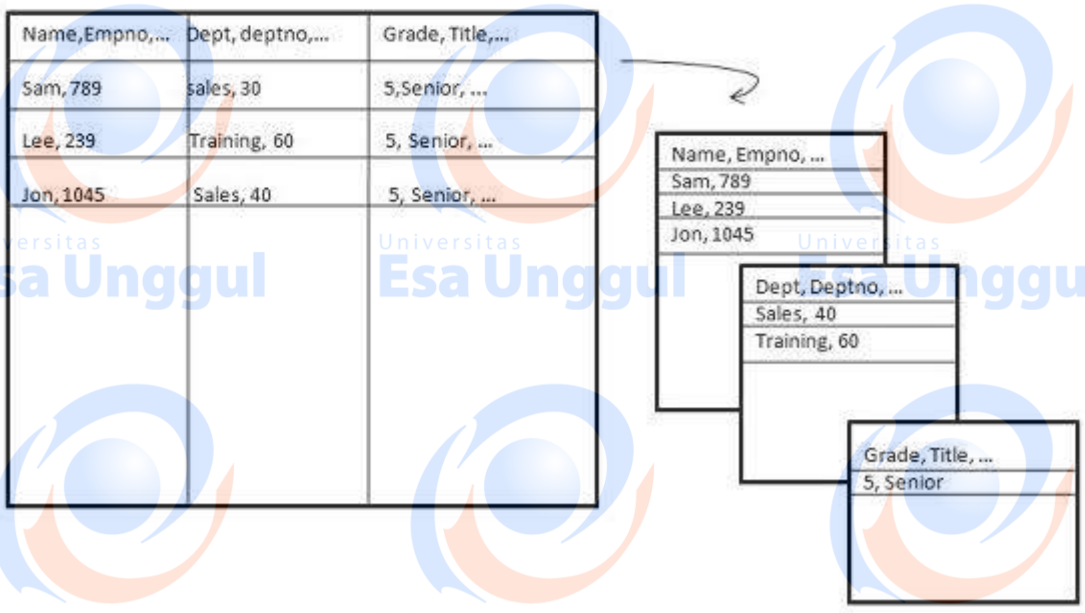
Round Robin Partitions

In the round robin technique, when a new partition is needed, the old one is archived. It uses metadata to allow user access tool to refer to the correct table partition.

This technique makes it easy to automate table management facilities within the data warehouse.

Vertical Partition

Vertical partitioning splits the data vertically. The following image depicts how vertical partitioning is done.



Vertical partitioning can be performed in the following two ways:

- Normalization
- Row Splitting

Normalization

Normalization is the standard relational method of database organization. In this method, the rows are collapsed into a single row, hence it reduces space. Take a look at the following tables that show how normalization is performed.

Table before Normalization

Product _id	**Qnty**	**Value**	**sales_date**	**Store_id**	**Store _name**	**Location**	**Region**
30	5	3.67	3-Aug-13	16	sunny	Bangalore	S
35	4	5.33	3-Sep-13	16	sunny	Bangalore	S
40	5	2.50	3-Sep-13	64	san	Mumbai	W
45	7	5.66	3-Sep-13	16	sunny	Bangalore	S

Table after Normalization

Store_id	**Store_name**	**Location**	**Region**
16	sunny	Bangalore	W
64	san	Mumbai	S

Product_id	**Quantity**	**Value**	**sales_date**	**Store_id**
30	5	3.67	3-Aug-13	16
35	4	5.33	3-Sep-13	16
40	5	2.50	3-Sep-13	64
45	7	5.66	3-Sep-13	16

Row Splitting

Row splitting tends to leave a one-to-one map between partitions. The motive of row splitting is to speed up the access to a large table by reducing its size.

Note: While using vertical partitioning, make sure that there is no requirement to perform a major join operation between two partitions.

Identify Key to Partition

It is crucial to choose the right partition key. Choosing a wrong partition key will lead to reorganizing the fact table. Let's have an example. Suppose we want to partition the following table.

```
Account_Txn_Table
transaction_id
account_id
transaction_type
value
transaction_date
region
branch_name
```

We can choose to partition on any key. The two possible keys could be

- region
- transaction_date

Suppose the business is organized in 30 geographical regions and each region has different number of branches. That will give us 30 partitions, which is reasonable. This partitioning is good enough because our requirements capture has shown that a vast majority of queries are restricted to the user's own business region.

If we partition by transaction_date instead of region, then the latest transaction from every region will be in one partition. Now the user who wants to look at data within his own region has to query across multiple partitions.

Hence it is worth determining the right partitioning key.

Modul 6

Naming

Distributed Systems

6. Name Services

Werner Nutt

Naming Concepts

Names = strings used to identify objects (files, computers, people, processes, objects)

- Textual names (human readable)
  - used to identify individual services, people
    - email address: Hans.Mair@inf.unibz.it
    - URL: www.google.com
  - or groups of people or objects
    - mailing lists: professors@unibz.it
    - mail domains (if there are several mail exchangers)

Naming Concepts (cntd)

- Numeric addresses (identify the location of an object)
  - locate individual resources, e.g.

    193.206.186.100 (IP host address)
  - special case: group addresses, e.g.

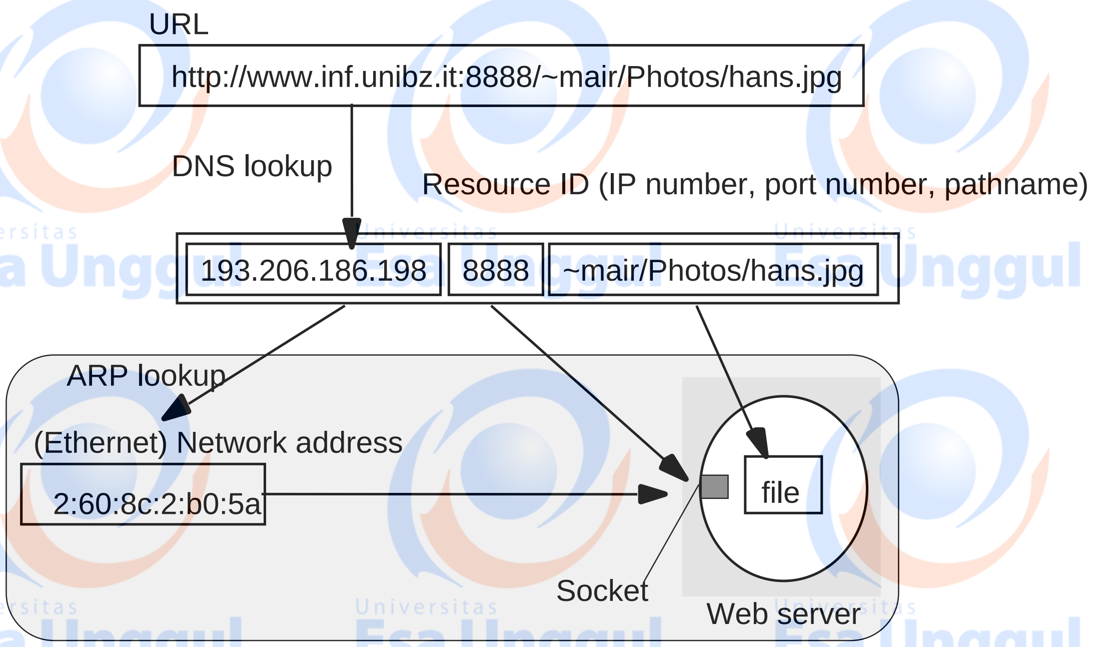
    multicast and broadcast addresses: IP Multicast, Ethernet
- Object identifiers
  - "pure" names (=bit patterns), usually numeric and large
    - never reused (include timestamp)
    - used for identification purposes

No real distinction between names and addresses.
Both must be looked up to obtain lower-level data (= name resolution).

Examples of Name Services

- File system
  - maps file name to file
- RMI registry
  - binds remote objects to symbolic names
- DNS (=Domain Name Service)
  - maps domain names to IP addresses
  - scalable, can handle change
- X.500/LDAP directory service
  - maps person's name to email address, phone number

Name Resolution on the WWW


URL
http://www.inf.unibz.it:8888/~mair/Photos/hans.jpg
DNS lookup
Resource ID (IP number, port number, pathname)
193.206.186.198
8888
~mair/Photos/hans.jpg
ARP lookup
(Ethernet) Network address
2:60:8c:2:b0:5a
file
Socket
Web server

Names and Resources

Currently, different name systems are used for each type of resource:

resource	*name*	*identifies*
file	pathname	file within a given file system
process	process id	process on a given computer
port	port number	IP port on a given computer

Uniform Resource Identifiers (URI) identify arbitrary resources:

- Uniform Resource Locator (URL): locates resource
  - typed by the scheme field (http, ftp, nfs, etc.)
  - part of the name is service-specific
  - resources cannot be moved between domains
- Uniform Resource Name (URN): names resource

Name Spaces

- Name space = collection of all valid names recognised by a service with
  - a syntax for specifying names, and
  - rules for resolving names (e.g., left to right)
- Naming context = maps a name to primitive attributes directly, or to another context and derived name (usually by prefixing)
  - telephone no: country, area, number
  - Internet host names: contexts = domains
  - Unix file system: contexts = directories

Name Spaces (cntd)

- Binding
  - associating a name to an object
  - binding names to attributes, one of which may be address
- Naming domain
  - has an authority that assigns names to objects within a name space or context
    - sysadmin assigns login names
    - Host names are assigned in a domain
  - object may be registered more than once within context
- Multiple names
  - alias (alternative name for an object, e.g. www, ftp, etc.)
  - symbolic name (alternative name which maps to a path name in the name space, e.g., symbolic link for file)

Hierarchic Name Spaces

- Sequence of name tokens resolved in different context
  - syntax: name token (text string) + delimiter
  - DNS: inf.unibz.it
  - Unix: /usr/bin
- Name structure reflects organisational structure
  - name changes if object migrates
  - names can be used relative to context or absolute
  - local contexts managed in a distributed fashion
- Examples
  - domain names, Unix file system

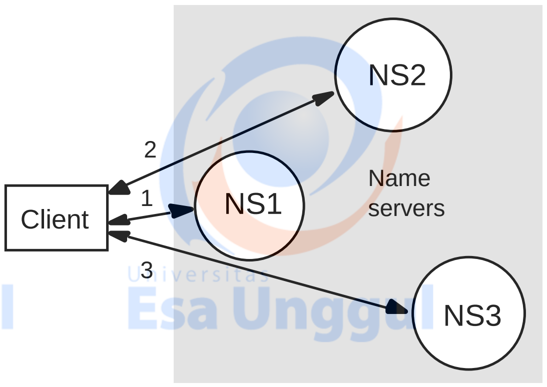
Flat Name Spaces

- Single global context and naming authority for all names
  - computer serial number
  - Ethernet address
  - remote object reference
    (IP address, port, time, object number, interface id)
- Names not meaningful
  - difficult to resolve (no tree hierarchy)
  - easy to create
  - easy to ensure uniqueness (timestamps)

Name Resolution

- Iteratively present name to a naming context
  - start with initial naming context
  - repeat as long as contexts + derived names are returned
  - aliases can introduce cycles
    (abandon after threshold no of resolutions or ensure no cycles)
- Replication/Caching
  - used for improved fault-tolerance on large services
    (more than one server, cf. DNS)
- Navigation
  - organising the access to several servers

Iterative Navigation

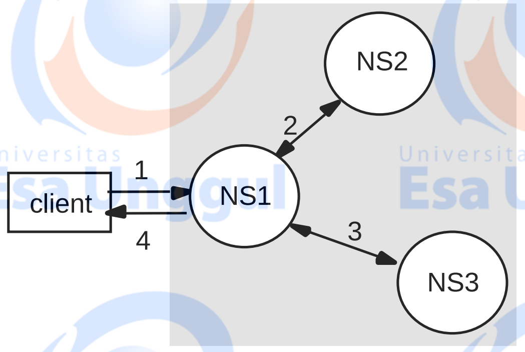


e.g., in DNS

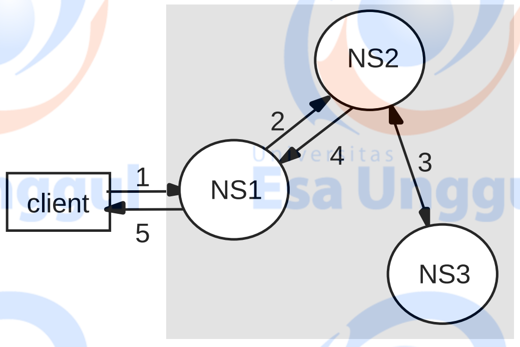
- The database is distributed over servers for different domains
- A client contacts servers NS1–NS3 one after the other in order to resolve a name
- Server returns attributes if it knows the name, otherwise suggests another server

Server-controlled Navigation

Name server communicates with other name servers on the client's behalf



Non-recursive server-controlled



Recursive server-controlled

In DNS, iterative navigation is the standard.
Recursive navigation is an option that is necessary in domains that limit client access to their DNS information for security reasons

Replication and Caching

Replicate some directories for performance and availability.

- Updates
  - Approach 1: write to single master, master propagates updates
  - Approach 2: write to any replica, later merge updates (timestamps)
  - Result: weak consistency (some entries out of date)
- Look-ups
  - try any local server, then go to root and down the tree
- Caching
  - names and addresses of recently used objects

Internet Domain Name System (DNS)

Maps host names to IP addresses (basically)

Design dates back to 1987 (Mockapetris)

Before

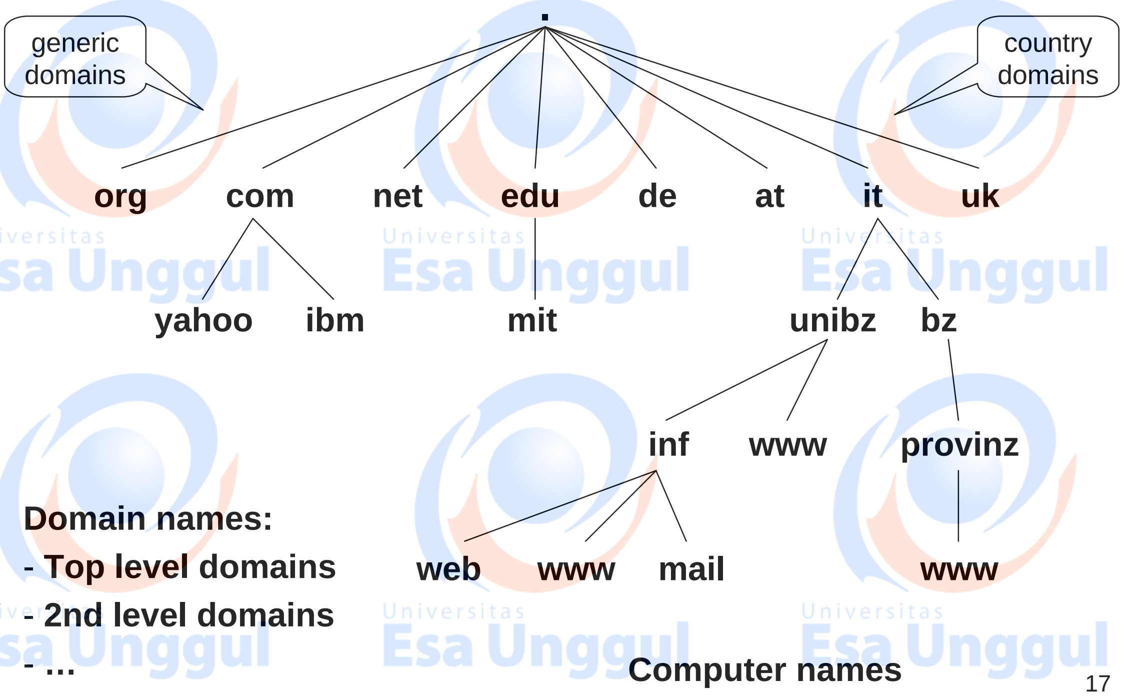
- all host names and addresses in one large master file
- stored on one central host
- downloaded by computers that needed to resolve names

What were the drawbacks of that approach?

Internet Domain Name System (cntd)

- Distributed naming database
- Hierarchical name structure reflects administrative structure of the Internet
- Rapidly resolves domain names to IP addresses
  - exploits caching heavily
  - typical query time ~100 milliseconds
- Scales to millions of computers
  - partitioned database
  - caching
- Resilient to failure of a server
  - replication (e.g., 13 root servers, 6 servers for .it, etc.)

The DNS Name Space



Domain names:
- **Top level domains**
- **2nd level domains**
- **…**

17

DNS Server Functions

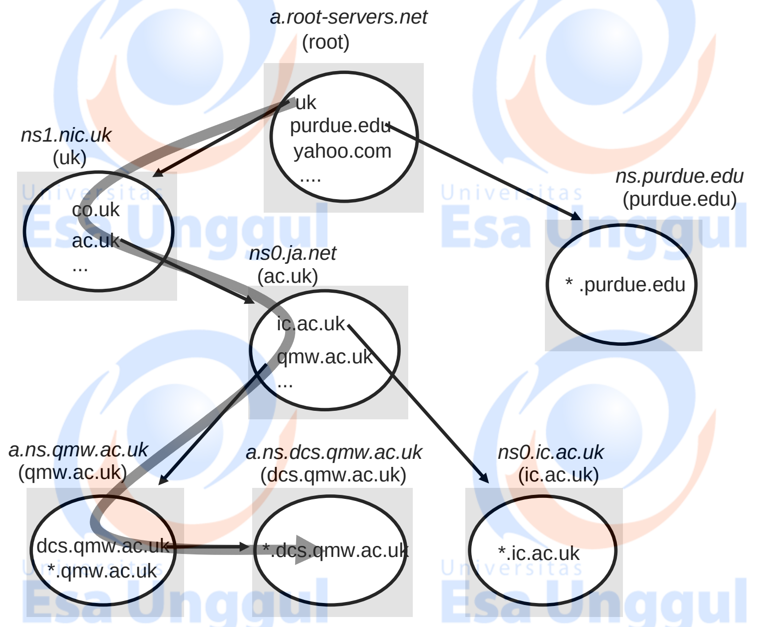
- Main function:
  - resolves domain names for computers, i.e. gets their IP addresses
  - caches the results of previous searches until they pass their “time to live”
- Info offered:
  - host IP addresses and canonical names
  - name servers for a domain
  - mail exchangers for a domain
  - host information - type of hardware and OS
  - well-known services - a list of well-known services offered by a host
- Other functions:
  - reverse resolution - get domain name from IP address

18

Example: DNS Servers

Look up IP-address of
`www.dcs.qmw.ac.uk`

- Name server names are in *italics*
- (Corresponding domains are in parentheses)
- → denotes a name server entry



DNS Servers and Zones

- The DNS namespace consists of zones:
  - zone = domain minus sub-domains, administered independently
- Every zone must have at least two name servers
  - exactly one master (= primary) server: contains the only writable copy of the “zone file”
  - one or more secondary (= slave) servers: copies its zone file from the master
  - both, master and slaves, are “authoritative” for the zone
  - set up should guarantee that slaves never hold information that is out of date

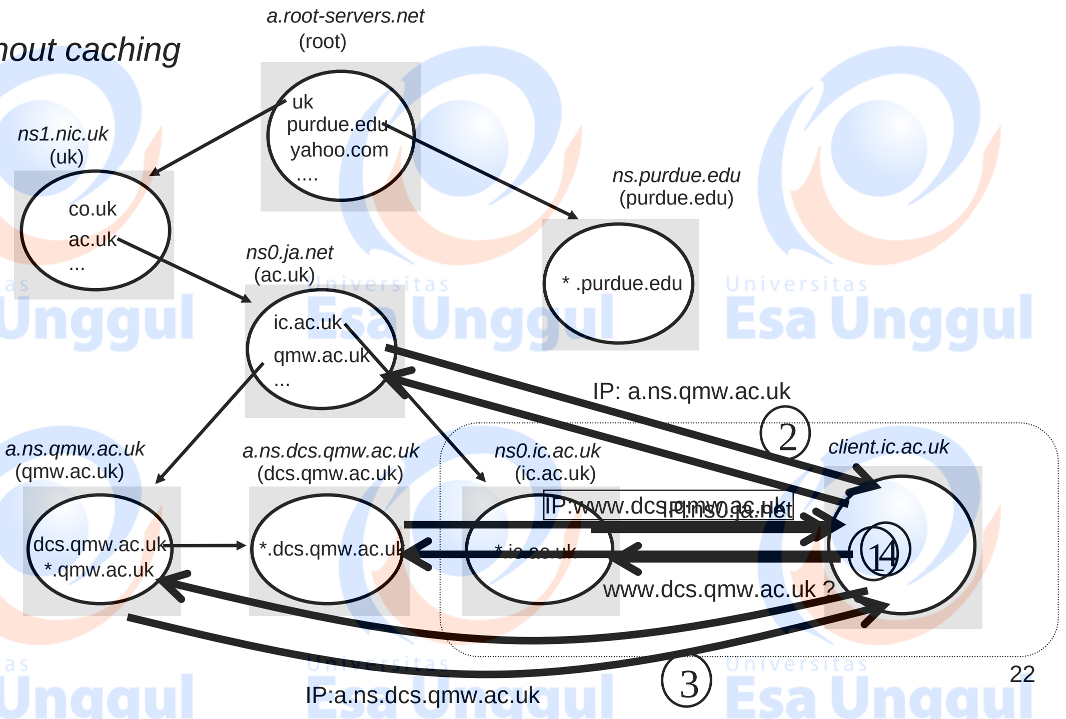
DNS Name Resolution

Basic algorithm

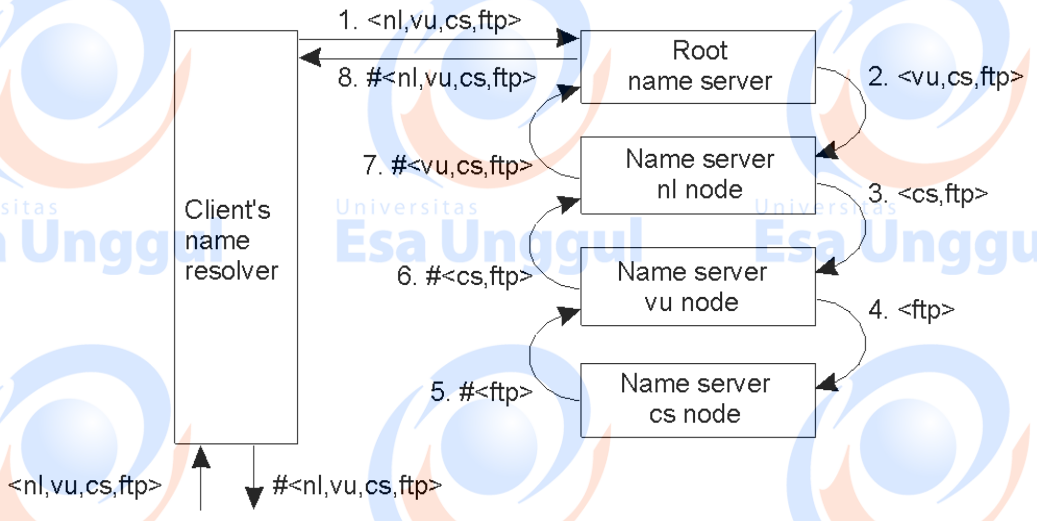
- Look for the name in the local cache
- Try a superior DNS server, which responds with:
  - another recommended DNS server
  - the IP address (which may not be entirely up to date)

DNS Iteration

Without caching



Recursive Name Resolution in DNS



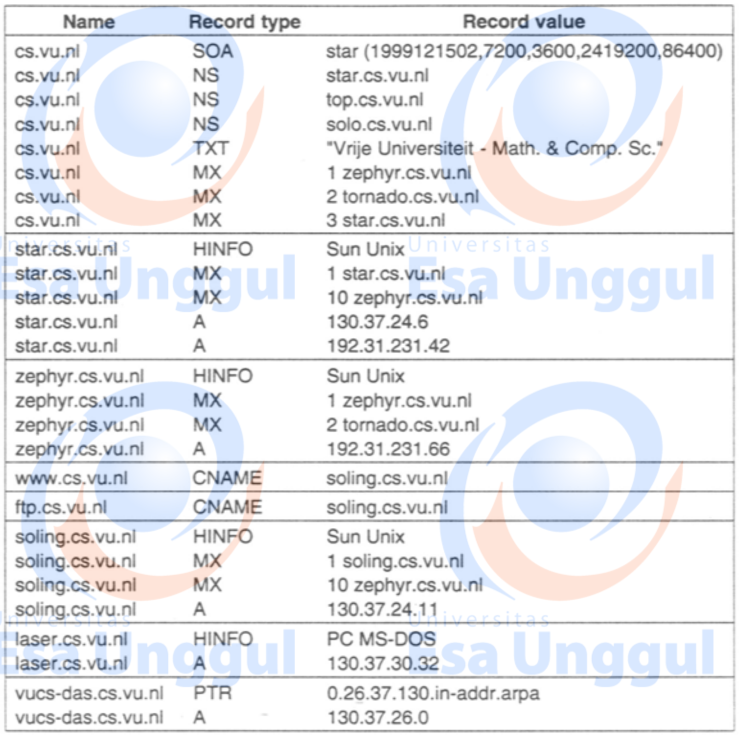
Types of DNS Resource Records

Record type	*Meaning*	*Main contents*
A	A computer address	IP number
NS	An authoritative name server	Domain name for server
CNAME	The canonical name for an alias	Domain name for alias
SOA	Marks the start of data for a zone	Parameters governing the zone
WKS	A well-known service description	List of service names and protocols
PTR	Domain name pointer (reverse lookups)	Domain name
HINFO	Host information	Machine architecture and operating system
MX	Mail exchange	List of <*preference, host*> pairs
TXT	Text string	Arbitrary text

DNS Serves organize their info in “resource records”

Name Server Content

An excerpt from the DNS database for the zone *cs.vu.nl*.

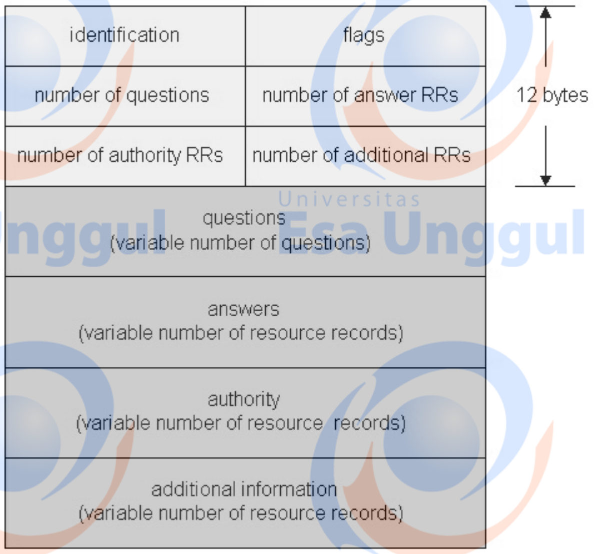


Name	Record type	Record value
cs.vu.nl	SOA	star (1999121502,7200,3600,2419200,86400)
cs.vu.nl	NS	star.cs.vu.nl
cs.vu.nl	NS	top.cs.vu.nl
cs.vu.nl	NS	solo.cs.vu.nl
cs.vu.nl	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl	MX	1 zephyr.cs.vu.nl
cs.vu.nl	MX	2 tornado.cs.vu.nl
cs.vu.nl	MX	3 star.cs.vu.nl
star.cs.vu.nl	HINFO	Sun Unix
star.cs.vu.nl	MX	1 star.cs.vu.nl
star.cs.vu.nl	MX	10 zephyr.cs.vu.nl
star.cs.vu.nl	A	130.37.24.6
star.cs.vu.nl	A	192.31.231.42
zephyr.cs.vu.nl	HINFO	Sun Unix
zephyr.cs.vu.nl	MX	1 zephyr.cs.vu.nl
zephyr.cs.vu.nl	MX	2 tornado.cs.vu.nl
zephyr.cs.vu.nl	A	192.31.231.66
www.cs.vu.nl	CNAME	soling.cs.vu.nl
ftp.cs.vu.nl	CNAME	soling.cs.vu.nl
soling.cs.vu.nl	HINFO	Sun Unix
soling.cs.vu.nl	MX	1 soling.cs.vu.nl
soling.cs.vu.nl	MX	10 zephyr.cs.vu.nl
soling.cs.vu.nl	A	130.37.24.11
laser.cs.vu.nl	HINFO	PC MS-DOS
laser.cs.vu.nl	A	130.37.30.32
vucs-das.cs.vu.nl	PTR	0.26.37.130.in-addr.arpa
vucs-das.cs.vu.nl	A	130.37.26.0

DNS Message Format

Queries and replies have the same format (using UDP)

Header
- identification: 16 bit number set in query, matching reply with same number
- flags: 1 bit each, e.g.,
  - query or reply
  - authoritative answer
  - recursion desired
  - recursion available


identification
flags
number of questions
number of answer RRs
number of authority RRs
number of additional RRs
12 bytes
questions (variable number of questions)
answers (variable number of resource records)
authority (variable number of resource records)
additional information (variable number of resource records)

DNS Message Format (cntd)

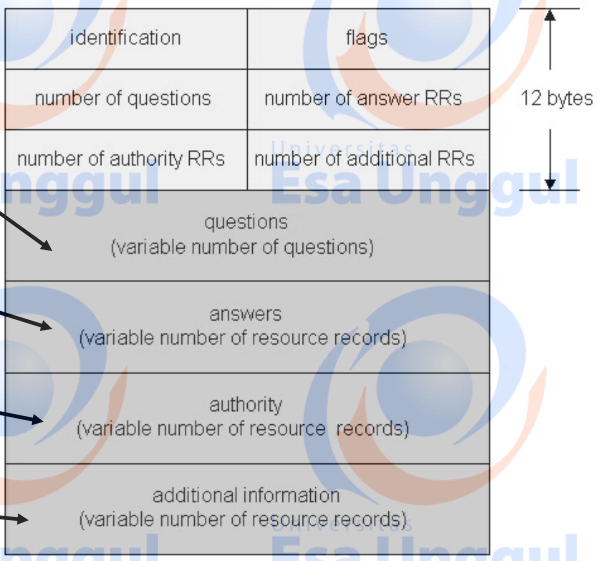
The message body consists of resource records

Domain names (or IP adds), type of records requested *(incomplete records)*

Resource records answering the query

Records pointing to authoritative servers

Typically, address records of the authoritative servers


identification
flags
number of questions
number of answer RRs
number of authority RRs
number of additional RRs
12 bytes
questions (variable number of questions)
answers (variable number of resource records)
authority (variable number of resource records)
additional information (variable number of resource records)

Implementations of DNS

- De facto standard for UNIX is BIND (= Berkeley Internet Name Domain)
  - Client programs acting as resolver link in library software (i.e., no process on client)
  - Server machines run a daemon (“named”)
  - Server can be configured as one of three categories:
    - primary, secondary, caching-only
- Microsoft’s Active Directory supports DNS

Access to DNS

- **`host`**
  - command for name resolution and reverse resolution
- **`nslookup`**
  - command/tool to query DNS servers for arbitrary info
- **`dig`**
  - similar to nslookup, without some of the deficiencies of the former

- /etc/resolv.conf
  - file containing IP address of default name server

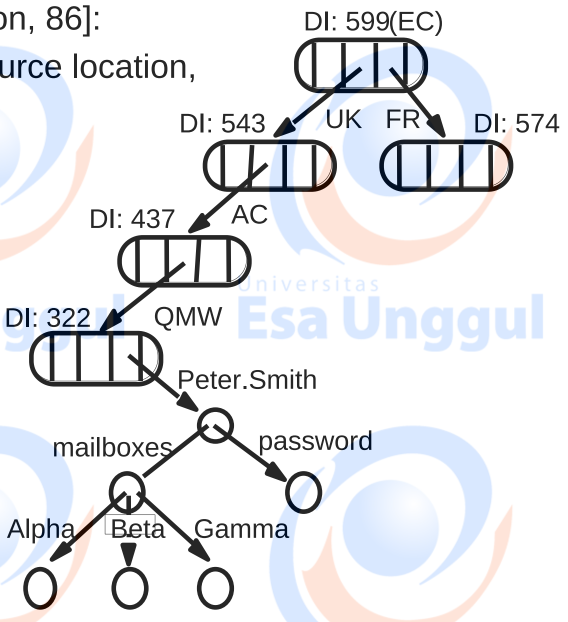
- Java JNDI (= Java Naming and Directory Interface)
  - provides interface for querying DNS

Global Name Service (GNS)

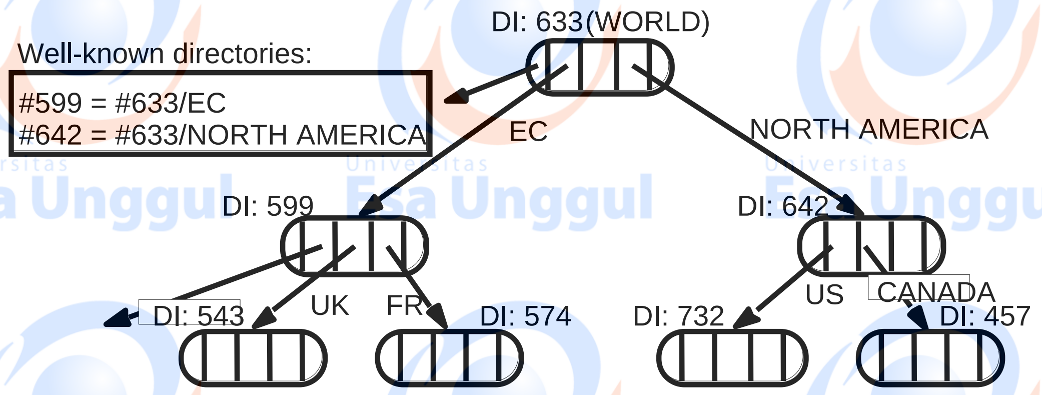
A proposal from research [B. Lampson, 86]:
GNS is more flexible system for resource location, mail addressing and authentication

- Structured leafs:
  "Value trees"
- Directory nodes have a unique directory identifier ID
- Names in GNS have two parts
  <directory name, value name>

GNS accommodates change:
use directory identifiers
to identify old roots



Merging Trees Under a New Root

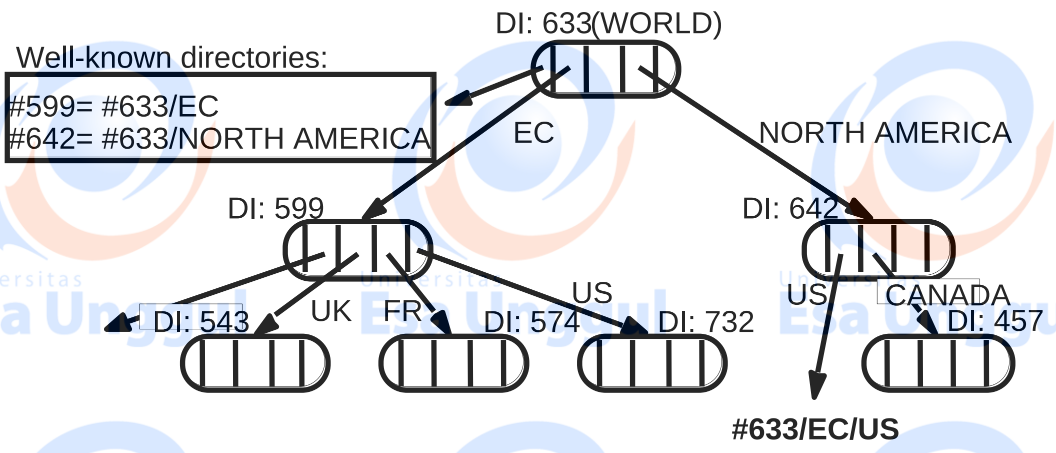


Well-known directories:

#599 = #633/EC
#642 = #633/NORTH AMERICA

Old "working roots" (like #599 (EC)) can be found in the new tree, using the "well-known" directories table of #633 (WORLD)

Restructuring the Directory



Well-known directories:

#599= #633/EC
#642= #633/NORTH AMERICA

- The US becomes part of the EU:
  a symbolic link (#633/EC/US) points to the new location

GNS gains flexibility at the cost of accumulating additional data after reconfiguration

Naming in Distributed Systems

Unique identifiers **UIDs** e.g. 128 bits
- are never reused
- refer to the same thing at all times, or to nothing at all

UIDs should be location-independent! Can the named object be moved?

Pure and impure names (Needham)

pure names
- the name itself yields no information, and commits the system to nothing
- it can only be used to compare with other similar names e.g. in table look-up

impure names
- the name yields information,
- and commits the system to maintaining the context in which it can be resolved

Examples of impure names

jmb25@cl.cam.ac.uk name of a person, registered in a DNS domain
jeanb@lcs.mit.edu name of a person, registered in another DNS domain
another or the same person?

puccini.cl.cam.ac.uk name of a machine, registered in a DNS domain

(disc-pack-ID, object-ID) Bad idea from history of naming files and directories.
Seemed efficient until the objects had to be moved

(host-ID, object-ID) OK, it's impure .... but how are pure names generated in a DS?
We must not have centralised name allocation.
(***host-ID, object-ID)*** has been used (badly) in middleware,
and has made the objects unmoveable.
It could be used to generate pure names,
if we do not make use of the separate fields. Typical example:

32-bit host-ID	96 bit object-ID

Unique names

Both pure and impure names can be unique.
Uniqueness can be achieved by using:

for impure names:

- hierarchical names: scope of uniqueness is level in hierarchy
  (uniqueness is within the names in the directory in which the name is recorded)

for pure names

- a bit pattern: flat, system-wide uniqueness,

Problems with pure names:

- where to look them up to find out information about them?
- how do you know that an object does not exist? How can a global search be avoided?
- how to engineer uniqueness reliably in a distributed system?
  centralised creation of names? As discussed above, ***(host-ID, object-ID)***?

Problems with impure names:

- how to restructure the namespace
  e.g. when objects move about or when companies restructure

Examples of (pure/impure) names - unique identification

UK national insurance - allocated on employment
US Social Security - allocated on employment
Passport
Driving licence
Services: RAC, AA, AAA(US), AAA(Aus)
Credit cards
Bank accounts
Utilities’ customer numbers: gas/electricity/water/phone
Charity members
Loyalty card members

For the above examples:

- Is the structure explicit or implicit?
- Is allocation centralised or distributed?
- What is the resolution context?

Name resolution - binding

Name resolution or binding

obtaining a value for an attribute of the named object that allows the object to be used

Late binding is considered good practice

Programs should contain names, not addresses

$$\begin{tikzcd}
\text{file-service?} \arrow[r] & \boxed{\text{name service}} \\
\text{IP-address, port\#, timestamp} & \arrow[l]
\end{tikzcd}$$

A machine may fail and the service moved to another machine.

Your local agent may cache resolved names for subsequent use,

and may expire values based on timestamps (TTL time-to-live)

Cached values are always used “at your own risk”.

They should not be embedded in programs.

If cached values don’t work, the lookup has to be repeated.

Lookup may be iterative for large-scale systems – see later.

Names, attributes and values stored in a name service

example:

object type	attribute list
user	login-name, mailbox-hosts(s)
computer	architecture, OS, network-address, owner
service	network-address, version#, protocol
group	list of names of members
alias	canonical name
directory?	list of hosts holding the directory (may be held in a separate structure rather than as a type of name, as here)

Directories are likely to be replicated for scalability, fault-tolerance, efficiency, availability

Directory names resolve to a list of hosts plus their addresses to avoid an extra lookup per host

Attribute-based (inverse) lookup may be offered –
A YELLOW PAGES style of service for object discovery e.g. X.500, LDAP

Too much information can be dangerous – useful for hackers

Names, attributes and values - examples

object type -> list of attribute names

name service holds:
object-type, object-name -> list of attribute values

You can acquire a standard directory service e.g. LDAP and use it to store whatever your service/application needs

querying:
object-type, object-name, attribute-name -> attribute value

computer, puccini.cl.cam.ac.uk, location -> IP-address
user, some-user-name, public-key -> PK bit pattern

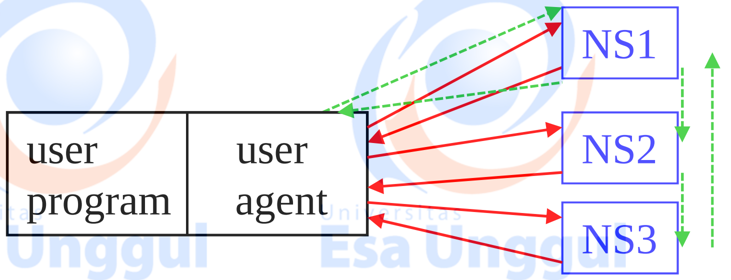
checking:
object-type, object-name, attribute-name, attribute value -> yes/no

ACL, filename, some-user-name, write-access -> yes/no

Attribute-based (inverse) lookup:
object-type, attribute-name, attribute value -> list of object-names

computer, OS version#, OS version# value -> list of computers

Iterative name resolution



user agent (UA) starts from the root address of the name service, or tries some well-known sub-tree root, e.g. the location of the ***uk*** directory may be used by agents in the UK

To resolve ***cl.cam.ac.uk*** the UA, as in

looks up ***ac***'s address in uk, then looks up ***cl***'s address in ***ac***

The UA will cache resolved names as hints for future use, see slide 6

Alternatively, any name server will take a name, resolve it and return the resolved value, as in

The client may be able to choose, e.g. select "recursive" in DNS

Engineering optimisations:

- use of caching at UA and at directories
- try cached values first

Name services - examples

DNS Internet Domain Name Service, see below

Grapevine: Xerox PARC early 1980's, *Birrell, Levin, Needham, Schroeder CACM 25(1) 1982*
two-level naming hierarchy e.g. ***name@registry*** ***birrell@pa***
primarily for email, but also gave primitive authentication and access control
(check password as attribute of user, check ACL)
any Grapevine server would take any request from a GV user agent

Clearinghouse, Xeroc PARC, Oppen and Dalal 1983
ISO standard based on an extension of Grapevine
three-level hierarchy

GNS Global Name Service, DEC SRC, *Lampson et al. 1986* - see below
full hierarchical naming
support for namespace restructuring

X.500 and LDAP, see below

Name services in Middleware
CORBA: naming and (interface) trading services, Java JNDI, Web W3C: UDDI
Allow registration of names/interfaces of externally invocable components with interface references and attributes such as location
May offer separate services for: ***name –> object-reference,*** ***object-reference –> location***

DNS

Computers using DNS are grouped into zones e.g. ***uk, cam***
Within a zone, management of nested sub-domains can be delegated
e.g. ***cl*** is managed locally by the domain manager who adds names to a local file
Each zone has a primary name server that holds the master list for the zone. Secondary name servers hold replicas for the zone.

Queries can relate to individual hosts or zones/domains, examples:

query			response
A	computer name	->	IPv4 address
AAAA	computer names	->	IPv6 address
MX	mail host for domain	->	list < host, preference, IP address >
	includes mail hosts for detached computers		
NS	DNS servers for a domain	->	list < host, authority?Y/N, IP address >

Name service design: Replication and Consistency

Directories are replicated for scalability, availability, reliability , ...

How should propagation of updates between replicas be managed?
lookup (arguments) < - > is the most recent value known, system wide, guaranteed to be returned?

If system-wide (strong) consistency were guaranteed this would imply:

- delay on update
- delay on lookup

It is essential to have fast access to naming data
– so we relax the consistency requirement

Is this justified?

Name services – assumptions to justify weak consistency

Design assumptions were as below, but new issues have arisen

- naming data change rarely,
- changes propagate quickly,
- inconsistencies will be rare

YES – information on users and machines
NO – distribution lists (see analysis of how Grapevine outgrew its specification)
NEW – mobile users, computers and small devices e.g. Internet-enabled phones
NEW – huge numbers of devices to be named – does the design rely on low update traffic?

- we detect obsolete naming data when it doesn't work
  YES – users
  NO – distribution lists

- If it works it doesn't matter that it's out of date
  - you might have made the request a little earlier
  - recall uncertainties over time in DS

Consistency – vs - Availability

We have argued that availability must be chosen for name services, so weak consistency
When only weak consistency is supported:

lookup (arguments) - > returns either: value, version# / timestamp
or: not known at time of last update

Examples:
Service on failed machine, restart at new IP address – update directory(s) – rare event
User changes company – coarse time grain
Companies merge – coarse time grain
Change of password – takes time to propagate – insecurity during propagation
Changes to ACLs and DLs – insecurity during propagation
Revocation of users' credentials – may have been used for authentication/authorisation
at session start – can the effect be made instantaneous?
Hot lists e.g. stolen credit cards – must propagate fast – push rather than pull model

Lessons:
Note design assumptions
Take care what data the name service is being used for
Does the service offer notification of change, on registration of interest, as in active databases?

Long-term Consistency

Requirement:
If updates stopped there would be consistency when all updates had propagated
Note that failure and restart behaviour must be specified for updates to propagate reliably

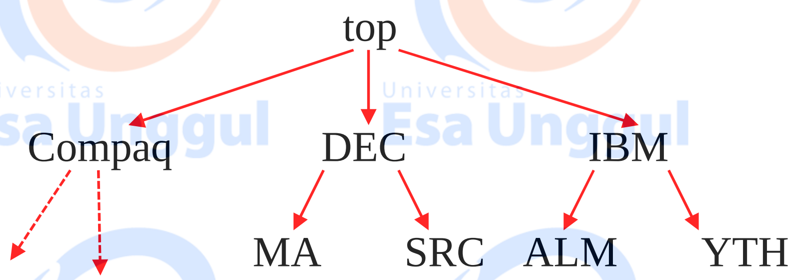
This requirement cannot be tested in a distributed system
- no guarantee that there will be periods of quiescence (no activity)

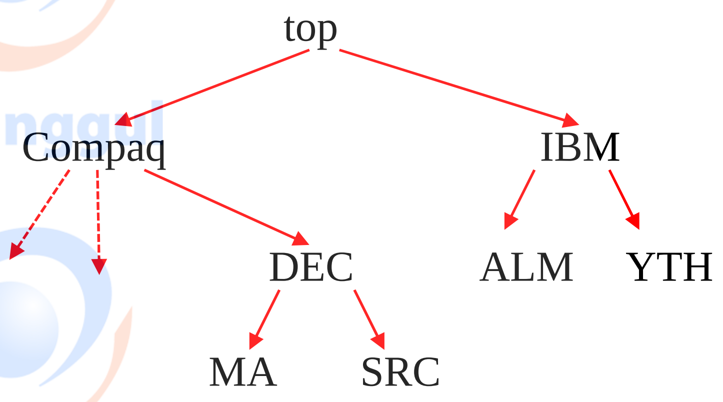
Updates are propagated by the message transport system
- conflicting updates might arrive out of order, from different sources
- need an arbitration policy based on timestamps
- but recall unreliability of source timestamps, so outcomes of an agreement protocol may not meet the *external* requirements.

Name services typically exchange whole directories periodically and compare them.
The directory is tagged with a new version# after this consistency check
e.g. GNS declares a new "epoch"

GNS namespace reconfiguration

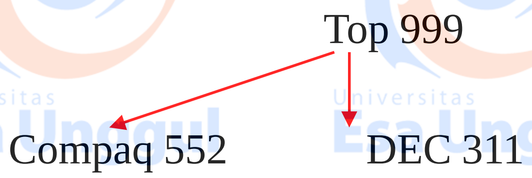
If the directory hierarchy is reconfigured, a directory may still be found via its DI
Names starting with that DI do not change if the reconfiguration is above that DI

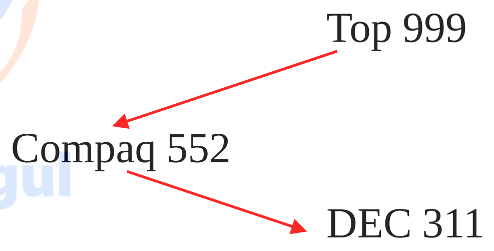




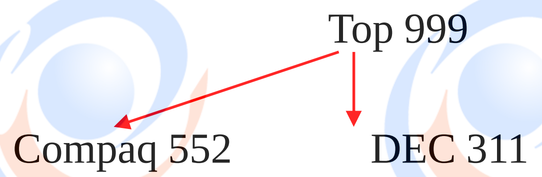
old names still work below the DEC directory

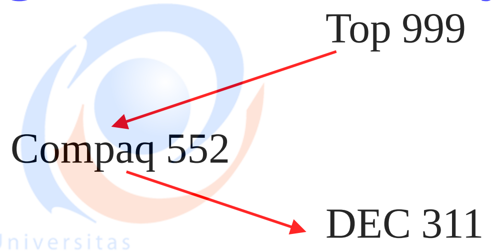
Support is needed by the directory service to locate a directory from its DI
a DI is a pure name – where do we look it up?
Directories usually map directory pathnames to IP addresses
In addition, top level GNS directories store DIs with directory names, e.g.





GNS namespace reconfiguration – directory updates





Names starting from DEC: ***311/SRC, birrell*** do not change. DEC is always 311
Names starting above DEC: ***999/DEC/SRC, birrell -> 999/Compaq/DEC/SRC, birrell***

Directory entries include – DIs with directory names
– pathnames from root

552 = 999/Compaq 311 = 999/DEC n = 999/DEC/SRC	IP addresses → → →

552 = 999/Compaq 311 = 999/Compaq/DEC n = 999/Compaq/DEC/SRC	IP addresses → → →

Modul 7

Synchronization

16. DWH – Security

The objective of a data warehouse is to make large amounts of data easily accessible to the users, hence allowing the users to extract information about the business as a whole. But we know that there could be some security restrictions applied on the data that can be an obstacle for accessing the information. If the analyst has a restricted view of data, then it is impossible to capture a complete picture of the trends within the business.

The data from each analyst can be summarized and passed on to management where the different summaries can be aggregated. As the aggregations of summaries cannot be the same as that of the aggregation as a whole, it is possible to miss some information trends in the data unless someone is analyzing the data as a whole.

Security Requirements

Adding security features affect the performance of the data warehouse, therefore it is important to determine the security requirements as early as possible. It is difficult to add security features after the data warehouse has gone live.

During the design phase of the data warehouse, we should keep in mind what data sources may be added later and what would be the impact of adding those data sources. We should consider the following possibilities during the design phase.

- Whether the new data sources will require new security and/or audit restrictions to be implemented?
- Whether the new users added who have restricted access to data that is already generally available?

This situation arises when the future users and the data sources are not well known. In such a situation, we need to use the knowledge of business and the objective of data warehouse to know likely requirements.

The following activities get affected by security measures:

- User access
- Data load
- Data movement
- Query generation

User Access

We need to first classify the data and then classify the users on the basis of the data they can access. In other words, the users are classified according to the data they can access.

Data Classification

The following two approaches can be used to classify the data:

- Data can be classified according to its sensitivity. Highly-sensitive data is classified as highly restricted and less-sensitive data is classified as less restrictive.

- Data can also be classified according to the job function. This restriction allows only specific users to view particular data. Here we restrict the users to view only that part of the data in which they are interested and are responsible for.

There are some issues in the second approach. To understand, let's have an example. Suppose you are building the data warehouse for a bank. Consider that the data being stored in the data warehouse is the transaction data for all the accounts. The question here is, who is allowed to see the transaction data. The solution lies in classifying the data according to the function.

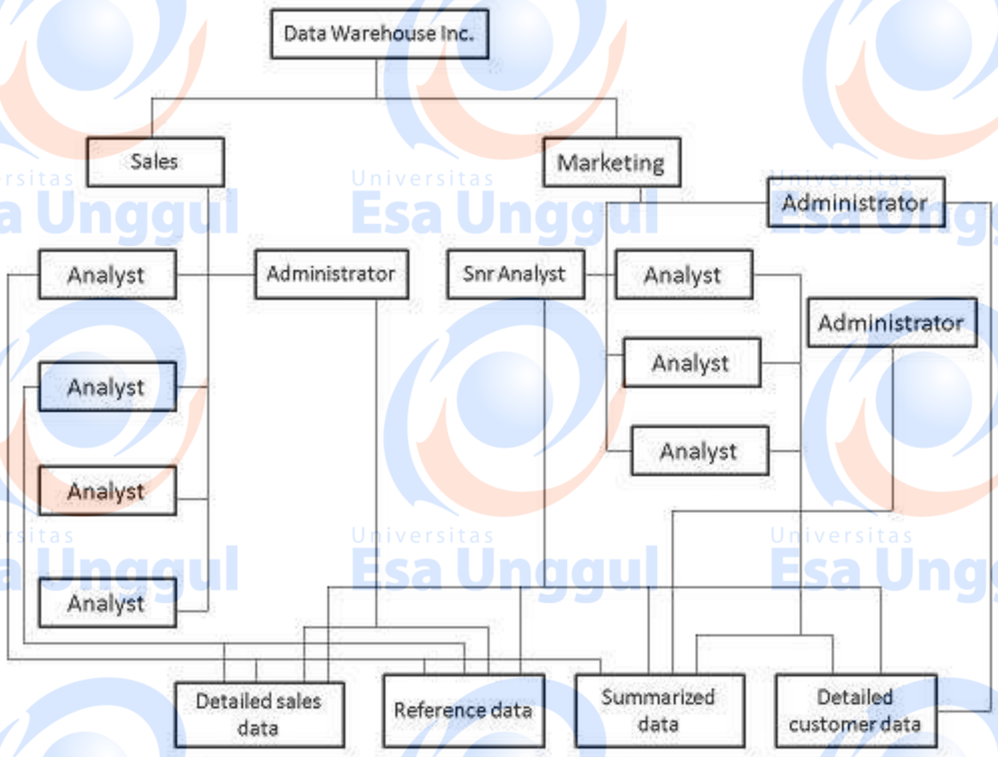
User Classification

The following approaches can be used to classify the users:

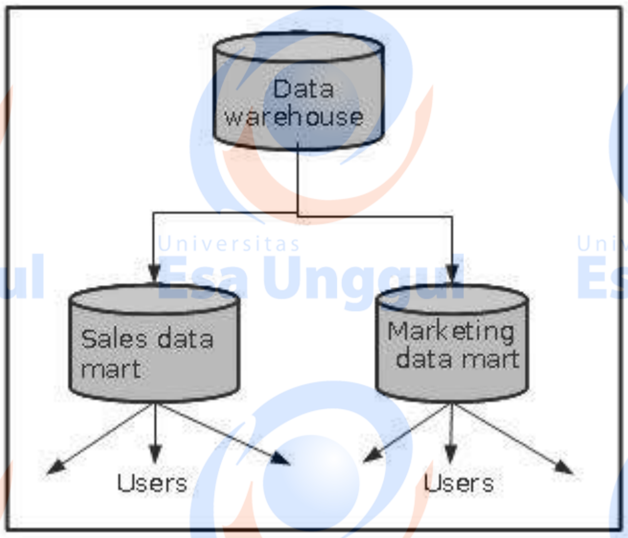
- Users can be classified as per the hierarchy of users in an organization, i.e., users can be classified by departments, sections, groups, and so on.
- Users can also be classified according to their role, with people grouped across departments based on their role.

Classification Based on Department

Let's have an example of a data warehouse where the users are from sales and marketing department. We can have security by top-to-down company view, with access centered on the different departments. But there could be some restrictions on users at different levels. This structure is shown in the following diagram.

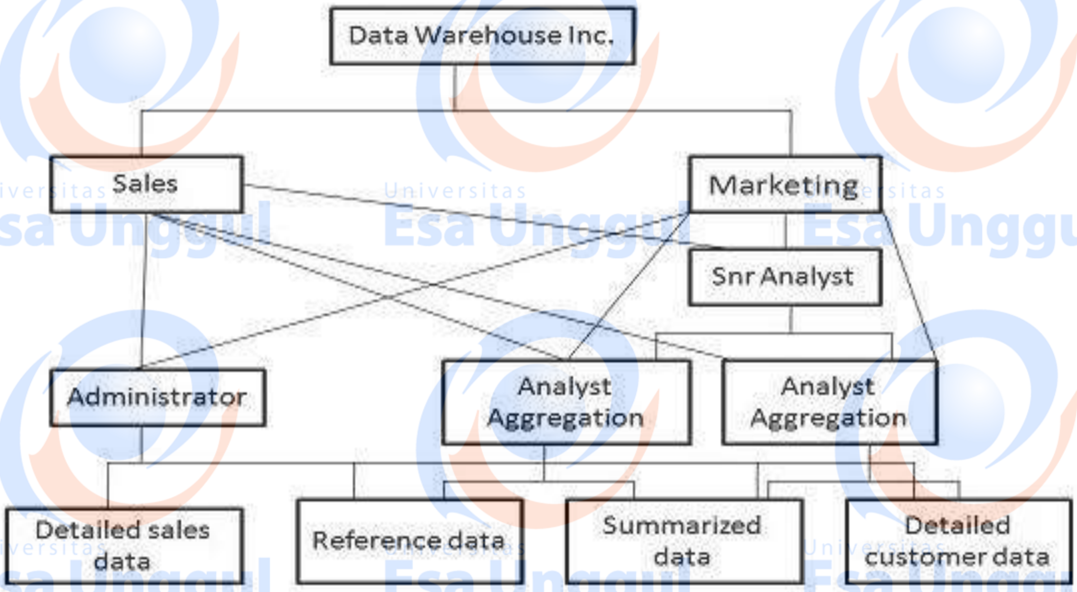


But if each department accesses different data, then we should design the security access for each department separately. This can be achieved by departmental data marts. Since these data marts are separated from the data warehouse, we can enforce separate security restrictions on each data mart. This approach is shown in the following figure.



Classification Based on Role

If the data is generally available to all the departments, then it is useful to follow the role access hierarchy. In other words, if the data is generally accessed by all the departments, then apply security restrictions as per the role of the user. The role access hierarchy is shown in the following figure.



Audit Requirements

Auditing is a subset of security, a costly activity. Auditing can cause heavy overheads on the system. To complete an audit in time, we require more hardware and therefore, it is recommended that wherever possible, auditing should be switched off. Audit requirements can be categorized as follows:

- Connections
- Disconnections

- Data access
- Data change

Note: For each of the above-mentioned categories, it is necessary to audit success, failure, or both. From the perspective of security reasons, the auditing of failures are very important. Auditing of failure is important because they can highlight unauthorized or fraudulent access.

Network Requirements

Network security is as important as other securities. We cannot ignore the network security requirement. We need to consider the following issues:

- Is it necessary to encrypt data before transferring it to the data warehouse?
- Are there restrictions on which network routes the data can take?

These restrictions need to be considered carefully. Following are the points to remember:

- The process of encryption and decryption will increase overheads. It would require more processing power and processing time.
- The cost of encryption can be high if the system is already a loaded system because the encryption is borne by the source system.

Data Movement

There exist potential security implications while moving the data. Suppose we need to transfer some restricted data as a flat file to be loaded. When the data is loaded into the data warehouse, the following questions are raised:

- Where is the flat file stored?
- Who has access to that disk space?

If we talk about the backup of these flat files, the following questions are raised:

- Do you backup encrypted or decrypted versions?
- Do these backups need to be made to special tapes that are stored separately?
- Who has access to these tapes?

Some other forms of data movement like query result sets also need to be considered. The questions raised while creating the temporary table are as follows:

- Where is that temporary table to be held?
- How do you make such table visible?

We should avoid the accidental flouting of security restrictions. If a user with access to the restricted data can generate accessible temporary tables, data can be visible to non-authorized users. We can overcome this problem by having a separate temporary area for users with access to restricted data.

Documentation

The audit and security requirements need to be properly documented. This will be treated as a part of justification. This document can contain all the information gathered from:

- Data classification
- User classification
- Network requirements
- Data movement and storage requirements
- All auditable actions

Impact of Security on Design

Security affects the application code and the development timescales. Security affects the following area:

- Application development
- Database design
- Testing

Application Development

Security affects the overall application development and it also affects the design of the important components of the data warehouse such as load manager, warehouse manager, and query manager. The load manager may require checking code to filter record and place them in different locations. More transformation rules may also be required to hide certain data. Also there may be requirements of extra metadata to handle any extra objects.

To create and maintain extra views, the warehouse manager may require extra codes to enforce security. Extra checks may have to be coded into the data warehouse to prevent it from being fooled into moving data into a location where it should not be available. The query manager requires the changes to handle any access restrictions. The query manager will need to be aware of all extra views and aggregations.

Database Design

The database layout is also affected because when security measures are implemented, there is an increase in the number of views and tables. Adding security increases the size of the database and hence increases the complexity of the database design and management. It will also add complexity to the backup management and recovery plan.

Testing

Testing the data warehouse is a complex and lengthy process. Adding security to the data warehouse also affects the testing time complexity. It affects the testing in the following two ways:

- It will increase the time required for integration and system testing.
- There is added functionality to be tested which will increase the size of the testing suite.

17. DWH – Backup

A data warehouse is a complex system and it contains a huge volume of data. Therefore it is important to back up all the data so that it becomes available for recovery in future as per requirement. In this chapter, we will discuss the issues in designing the backup strategy.

Backup Terminologies

Before proceeding further, you should know some of the backup terminologies discussed below.

- **Complete backup** – It backs up the entire database at the same time. This backup includes all the database files, control files, and journal files.
- **Partial backup** – As the name suggests, it does not create a complete backup of the database. Partial backup is very useful in large databases because they allow a strategy whereby various parts of the database are backed up in a round-robin fashion on a day-to-day basis, so that the whole database is backed up effectively once a week.
- **Cold backup** - Cold backup is taken while the database is completely shut down. In multi-instance environment, all the instances should be shut down.
- **Hot backup** - Hot backup is taken when the database engine is up and running. The requirements of hot backup varies from RDBMS to RDBMS.
- **Online backup** - It is quite similar to hot backup.

Hardware Backup

It is important to decide which hardware to use for the backup. The speed of processing the backup and restore depends on the hardware being used, how the hardware is connected, bandwidth of the network, backup software, and the speed of server's I/O system. Here we will discuss some of the hardware choices that are available and their pros and cons. These choices are as follows:

- Tape Technology
- Disk Backups

Tape Technology

The tape choice can be categorized as follows:

- Tape media
- Standalone tape drives
- Tape stackers
- Tape silos

Tape Media

There exists several varieties of tape media. Some tape media standards are listed in the table below:

Tape Media	Capacity	I/O rates
DLT	40 GB	3 MB/s
3490e	1.6 GB	3 MB/s
8 mm	14 GB	1 MB/s

Other factors that need to be considered are as follows:

- Reliability of the tape medium
- Cost of tape medium per unit
- Scalability
- Cost of upgrades to tape system
- Cost of tape medium per unit
- Shelf life of tape medium

Standalone Tape Drives

The tape drives can be connected in the following ways:

- Direct to the server
- As network available devices
- Remotely to other machine

There could be issues in connecting the tape drives to a data warehouse.

- Consider the server is a 48node MPP machine. We do not know the node to connect the tape drive and we do not know how to spread them over the server nodes to get the optimal performance with least disruption of the server and least internal I/O latency.
- Connecting the tape drive as a network available device requires the network to be up to the job of the huge data transfer rates. Make sure that sufficient bandwidth is available during the time you require it.
- Connecting the tape drives remotely also require high bandwidth.

Tape Stackers

The method of loading multiple tapes into a single tape drive is known as tape stackers. The stacker dismounts the current tape when it has finished with it and loads the next tape, hence only one tape is available at a time to be accessed. The price and the capabilities may vary, but the common ability is that they can perform unattended backups.

Tape Silos

Tape silos provide large store capacities. Tape silos can store and manage thousands of tapes. They can integrate multiple tape drives. They have the software and hardware to label and store the tapes they store. It is very common for the silo to be connected remotely over a network or a dedicated link. We should ensure that the bandwidth of the connection is up to the job.

Disk Backups

Methods of disk backups are:

- Disk-to-disk backups
- Mirror breaking

These methods are used in the OLTP system. These methods minimize the database downtime and maximize the availability.

Disk-to-Disk Backups

Here backup is taken on the disk rather on the tape. Disk-to-disk backups are done for the following reasons:

- Speed of initial backups
- Speed of restore

Backing up the data from disk to disk is much faster than to the tape. However it is the intermediate step of backup. Later the data is backed up on the tape. The other advantage of disk-to-disk backups is that it gives you an online copy of the latest backup.

Mirror Breaking

The idea is to have disks mirrored for resilience during the working day. When backup is required, one of the mirror sets can be broken out. This technique is a variant of disk-to-disk backups.

Note: The database may need to be shutdown to guarantee consistency of the backup.

Optical Jukeboxes

Optical jukeboxes allow the data to be stored near line. This technique allows a large number of optical disks to be managed in the same way as a tape stacker or a tape silo. The drawback of this technique is that it has slow write speed than disks. But the optical media provides long-life and reliability that makes them a good choice of medium for archiving.

Software Backups

There are software tools available that help in the backup process. These software tools come as a package. These tools not only take backup, they can effectively manage and control the backup strategies. There are many software packages available in the market. Some of them are listed in the following table:

Package Name	Vendor
Networker	Legato
ADSM	IBM
Epoch	Epoch Systems
Omniback II	HP
Alexandria	Sequent

Criteria for Choosing Software Packages

The criteria for choosing the best software package are listed below:

- How scalable is the product as tape drives are added?
- Does the package have client-server option, or must it run on the database server itself?
- Will it work in cluster and MPP environments?
- What degree of parallelism is required?
- What platforms are supported by the package?
- Does the package support easy access to information about tape contents?
- Is the package database aware?
- What tape drive and tape media are supported by the package?

18. DWH – Tuning

A data warehouse keeps evolving and it is unpredictable what query the user is going to post in the future. Therefore it becomes more difficult to tune a data warehouse system. In this chapter, we will discuss how to tune the different aspects of a data warehouse such as performance, data load, queries, etc.

Difficulties in Data Warehouse Tuning

Tuning a data warehouse is a difficult procedure due to following reasons:

- Data warehouse is dynamic; it never remains constant.
- It is very difficult to predict what query the user is going to post in the future.
- Business requirements change with time.
- Users and their profiles keep changing.
- The user can switch from one group to another.
- The data load on the warehouse also changes with time.

Note: It is very important to have a complete knowledge of data warehouse.

Performance Assessment

Here is a list of objective measures of performance:

- Average query response time
- Scan rates
- Time used per day query
- Memory usage per process
- I/O throughput rates

Following are the points to remember.

- It is necessary to specify the measures in service level agreement (SLA).
- It is of no use trying to tune response time, if they are already better than those required.
- It is essential to have realistic expectations while making performance assessment.
- It is also essential that the users have feasible expectations.
- To hide the complexity of the system from the user, aggregations and views should be used.
- It is also possible that the user can write a query you had not tuned for.

Data Load Tuning

Data load is a critical part of overnight processing. Nothing else can run until data load is complete. This is the entry point into the system.

Note: If there is a delay in transferring the data or in arrival of data, then the entire system is affected badly. Therefore it is very important to tune the data load first.

There are various approaches of tuning data load that are discussed below:

- The very common approach is to insert data using the **SQL Layer**. In this approach, normal checks and constraints need to be performed. When the data is inserted into the table, the code will run to check for enough space to insert the data. If sufficient space is not available, then more space may have to be allocated to these tables. These checks take time to perform and are costly to CPU.

- The second approach is to bypass all these checks and constraints and place the data directly into the preformatted blocks. These blocks are later written to the database. It is faster than the first approach, but it can work only with whole blocks of data. This can lead to some space wastage.

- The third approach is that while loading the data into the table that already contains the table, we can maintain indexes.

- The fourth approach says that to load the data in tables that already contain data, **drop the indexes & recreate them** when the data load is complete. The choice between the third and the fourth approach depends on how much data is already loaded and how many indexes need to be rebuilt.

Integrity Checks

Integrity checking highly affects the performance of the load. Following are the points to remember:

- Integrity checks need to be limited because they require heavy processing power.

- Integrity checks should be applied on the source system to avoid performance degrade of data load.

Tuning Queries

We have two kinds of queries in a data warehouse:

- Fixed queries
- Ad hoc queries

Fixed Queries

Fixed queries are well defined. Following are the examples of fixed queries:

- Regular reports
- Canned queries
- Common aggregations

Tuning the fixed queries in a data warehouse is same as in a relational database system. The only difference is that the amount of data to be queried may be different. It is good to store the most successful execution plan while testing fixed queries. Storing these executing plan will allow us to spot changing data size and data skew, as it will cause the execution plan to change.

Note: We cannot do more on fact table but while dealing with dimension tables or the aggregations, the usual collection of SQL tweaking, storage mechanism, and access methods can be used to tune these queries.

Ad hoc Queries

To understand ad hoc queries, it is important to know the ad hoc users of the data warehouse. For each user or group of users, you need to know the following:

- The number of users in the group
- Whether they use ad hoc queries at regular intervals of time
- Whether they use ad hoc queries frequently
- Whether they use ad hoc queries occasionally at unknown intervals.
- The maximum size of query they tend to run
- The average size of query they tend to run
- Whether they require drill-down access to the base data
- The elapsed login time per day
- The peak time of daily usage
- The number of queries they run per peak hour

Points to Note

- It is important to track the user's profiles and identify the queries that are run on a regular basis.
- It is also important that the tuning performed does not affect the performance.
- Identify similar and ad hoc queries that are frequently run.
- If these queries are identified, then the database will change and new indexes can be added for those queries.
- If these queries are identified, then new aggregations can be created specifically for those queries that would result in their efficient execution.

19. DWH – Testing

Testing is very important for data warehouse systems to make them work correctly and efficiently. There are three basic levels of testing performed on a data warehouse:

- Unit testing
- Integration testing
- System testing

Unit Testing

- In unit testing, each component is separately tested.
- Each module, i.e., procedure, program, SQL Script, Unix shell is tested.
- This test is performed by the developer.

Integration Testing

- In integration testing, the various modules of the application are brought together and then tested against the number of inputs.
- It is performed to test whether the various components do well after integration.

System Testing

- In system testing, the whole data warehouse application is tested together.
- The purpose of system testing is to check whether the entire system works correctly together or not.
- System testing is performed by the testing team.
- Since the size of the whole data warehouse is very large, it is usually possible to perform minimal system testing before the test plan can be enacted.

Test Schedule

First of all, the test schedule is created in the process of developing the test plan. In this schedule, we predict the estimated time required for the testing of the entire data warehouse system.

There are different methodologies available to create a test schedule, but none of them are perfect because the data warehouse is very complex and large. Also the data warehouse system is evolving in nature. One may face the following issues while creating a test schedule:

- A simple problem may have a large size of query that can take a day or more to complete, i.e., the query does not complete in a desired time scale.

- There may be hardware failures such as losing a disk or human errors such as accidentally deleting a table or overwriting a large table.

Note: Due to the above-mentioned difficulties, it is recommended to always double the amount of time you would normally allow for testing.

Testing Backup Recovery

Testing the backup recovery strategy is extremely important. Here is the list of scenarios for which this testing is needed:

- Media failure
- Loss or damage of table space or data file
- Loss or damage of redo log file
- Loss or damage of control file
- Instance failure
- Loss or damage of archive file
- Loss or damage of table
- Failure during data failure

Testing Operational Environment

There are a number of aspects that need to be tested. These aspects are listed below.

- **Security** - A separate security document is required for security testing. This document contains a list of disallowed operations and devising tests for each.
- **Scheduler** - Scheduling software is required to control the daily operations of a data warehouse. It needs to be tested during system testing. The scheduling software requires an interface with the data warehouse, which will need the scheduler to control overnight processing and the management of aggregations.
- **Disk Configuration** - Disk configuration also needs to be tested to identify I/O bottlenecks. The test should be performed with multiple times with different settings.
- **Management Tools** - It is required to test all the management tools during system testing. Here is the list of tools that need to be tested.
  - Event manager
  - System manager
  - Database manager
  - Configuration manager
  - Backup recovery manager

Testing the Database

The database is tested in the following three ways:

- **Testing the database manager and monitoring tools** - To test the database manager and the monitoring tools, they should be used in the creation, running, and management of test database.
- **Testing database features** - Here is the list of features that we have to test:
  - Querying in parallel
  - Create index in parallel
  - Data load in parallel
- **Testing database performance** - Query execution plays a very important role in data warehouse performance measures. There are sets of fixed queries that need to be run regularly and they should be tested. To test ad hoc queries, one should go through the user requirement document and understand the business completely. Take time to test the most awkward queries that the business is likely to ask against different index and aggregation strategies.

Testing the Application

- All the managers should be integrated correctly and work in order to ensure that the end-to-end load, index, aggregate and queries work as per the expectations.
- Each function of each manager should work correctly.
- It is also necessary to test the application over a period of time.
- Weekend and month-end tasks should also be tested.

Logistic of the Test

The aim of system test is to test all of the following areas:

- Scheduling software
- Day-to-day operational procedures
- Backup recovery strategy
- Management and scheduling tools
- Overnight processing
- Query performance

Note: The most important point is to test the scalability. Failure to do so will leave us a system design that does not work when the system grows.