1 : Data Management Evolution

IT3306 – Data Management

Level II - Semester 3

Overview

- This lesson on data management evolution discusses the gradual development of management of data.
- Here we look into Object, XML and NoSQL databases in detail.
- Finally, we present a comparison between relational databases concepts and non-relational databases.

Intended Learning Outcomes

At the end of this lesson, you will be able to;

- Describe how and why different data management techniques evolved.
- Identify the requirements for Object Databases.
- Explain the key concepts incorporated from OOP.
- Recognise the importance of NoSQL databases.
- Describe different data models available in NoSQL.
- Analyse the difference between NoSQL and relational Databases.

List of subtopics

1.1. Major concepts of object-oriented, XML, and NoSQL databases

1.1.1. Object Databases

- 1.1.1.1. Overview of Object Database Concepts
    - 1.1.1.1.1 Introduction to object-oriented concepts and features
    - 1.1.1.1.2 Object Identity and Literals
    - 1.1.1.1.3 Encapsulation of Operations
    - 1.1.1.1.4 Persistence of Objects
    - 1.1.1.1.5 Type Hierarchies and Inheritance
    - 1.1.1.1.6 Polymorphism and Multiple Inheritance

List of subtopics

1.1.2. XML Databases

1.1.2.1. Reason for the origination of XML

1.1.2.2. Structured, Semi structured, and Unstructured Data Structured data, Storage in relational database, Semi structured data, Directed graph model, Unstructured data

1.1.2.3. XML Hierarchical (Tree) Data Model Basic objects, Element, Attribute, Document types, Data-centric and Document-centric, Hybrid

List of subtopics

1.1.3. NoSQL Databases

1.1.3.1. Origins of NoSQL Impedance Mismatch, Problem of clusters, Common characteristics of NoSQL databases, Important rise of NoSQL with Polyglot Persistence.

1.1.3.2. Data models in NoSQL

1.1.3.2.1. Introduction to Aggregate data models, Reason for using Aggregate data models

1.1.3.2.2. Key-Value Model and suitable Use Cases

1.1.3.2.3. Document Data Model and suitable Use Cases

1.1.3.2.4. Column-Family Stores and suitable Use Cases

1.1.3.2.5. Data model for complex relationship structures (Graph database model)

List of subtopics

1.2 Contrast and compare relational databases concepts and non-relational databases.

1.2.1. Object databases and Relational databases

1.2.2. XML and Relational databases

1.2.3. NoSQL and Relational databases

- 1.2.3.1. Data modelling difference, Modeling for Data Access
- 1.2.3.2. Aggregate oriented vs aggregate ignorant
- 1.2.3.3. Schemalessness in NoSQL
- 1.2.3.4. Overview of Materialised views

Object Databases - Overview of Object Database Concepts

Overview of Object Database Concepts

- Relational database systems use relational data model, similarly Object Databases (aka. Object-Oriented Databases - OODB) are built on **object data model.**
- The major importance of object database is the flexibility to determine the ***structure*** and relevant ***operations*** of the objects***.***
- In early days business requirements were dealt with traditional data models such as network model, hierarchical model and relational model.

Object Databases - Overview of Object Database Concepts

Overview of Object Database Concepts

- However, Real time applications and information systems which require high performance and calculations such as; Telecommunication, Architectural Designing, Biological sciences and Geographical Information Systems (GIS), has shortcomings when using traditional data models due to their rigid structures.
- Object databases are developed to serve the new business requirements and are frequently used in aforementioned domains.

Object Databases - Overview of Object Database Concepts

Overview of Object Database Concepts

- popularity of Object-Oriented programming languages is the second fact that bring about object databases.
- Sometimes, applications developed with Object-Oriented Languages (e.g.: C++, Java) meet conflicts when use with traditional databases.
- However, object databases facilitate the **seamless** integration with application developed using object-oriented languages.
- There are some RDBMS include the features of object databases. Those are known as ***object-relational*** or ***RDBMSs.***
- Due to the popularity, relational and object-relational database systems are widely used when compared to the object databases.

Object Databases - Overview of Object Database Concepts

Overview of Object Database Concepts

- **Orion System** developed by Microelectronics and Computer Technology Corporation, **OpenOODB** - Texas Instruments,Iris system by Hewlett-Packard (*hp*) laboratories, the Ode system by AT&T Bell Labs, and the **ENCORE/ObServer** project by Brown University are some examples for experimental prototypes of Object DBs.

- **GemStone Object Server** of GemStone Systems, **ONTOS DB** of Ontos, **Objectivity/DB** of Objectivity Inc., **Versant Object** Database and **FastObjects** by Versant Corporation (and Poet), **ObjectStore** of Object Design, and **Ardent Database** of Ardent are commercially available Object Database Systems.

Object Databases - Overview of Object Database Concepts

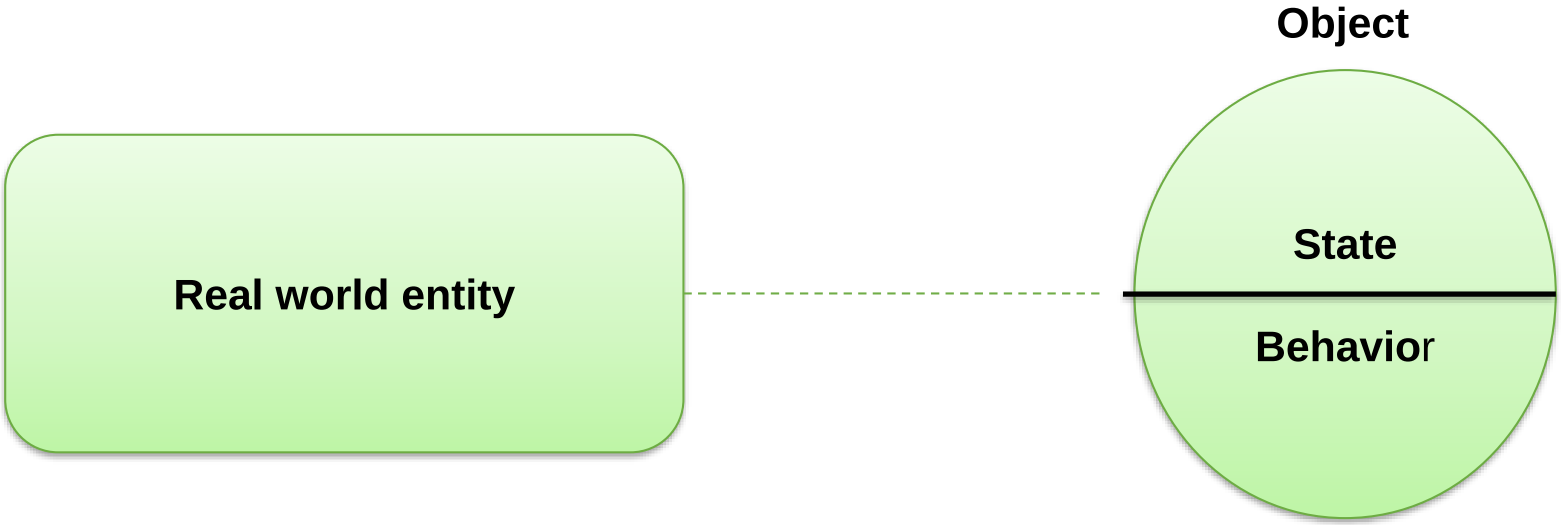
Introduction to object-oriented concepts and features

- Object- Oriented abbreviated as O-O , this term is originated from Object Oriented Programming Languages (OOPL).
- Later, the concepts used in OOPL was introduced to other areas, such as; Database, Software engineering, Computer Systems and Knowledge base etc.
- Most of the concepts originally developed for OOPL, has been adopted by Object databases.

Object Databases - Overview of Object Database Concepts

Introduction to object-oriented concepts and features

- The major components of an object are,
  - **State (value):** can have complex data structure
  - **Behavior (operations)**



Object Databases - Overview of Object Database Concepts

OOPL has two object categories based on their existence

- **Transient objects -** objects which only exist while the program is running.
- **Persistent objects -** objects which exist even after the termination of the program.

Object Databases - Overview of Object Database Concepts

Introduction to object-oriented concepts and features

- Object- Oriented(O-O) Databases expand the existence of an object by permanently storing it in a database.
- In a secondary storage, O-O databases can store persistent objects and can be retrieved later.
- Data stored inside OO databases can be shared among different applications and programs.

Object Databases - Overview of Object Database Concepts

- **Object Identity**
  - Database objects needs to have coherence with real-world objects in order to preserve their integrity and identity. It will make easy to identify objects and operate on. Therefore, a unique identity is assigned to each independent object stored in the database known as **Object Identifier (OID).**
  - OID generally is **system generated**.
  - The value of the OID might be hidden from external users. However, it is used inside the system to uniquely identify the objects and to create and manage inter-object references.

Object Databases - Overview of Object Database Concepts

- **Object Identifier (OID)**
  - Main properties of OID,
    - **Immutable** - value of OID does not change
    - **Unique - used only once**. OID of deleted object will not be assigned to a new object in the future.
- According to these two properties of OID, OID is independent from any attribute value of an object. (because attribute value may change over time)
- Object database systems must have mechanism to generate OID and preserve **immutability.** Since an object retains its identifier over its lifetime, the object remains the same despite changes in its state.
- OID is similar to the primary key attribute in relational databases, which is used to uniquely identify the tuples.

Object Databases - Overview of Object Database Concepts

Literals

- The Object Model supports different literal types, which are considered as attribute values.
- Literals are embedded inside objects and the object model facilitates to define complex structured literals within an object.
- Literals do not have identifiers (OIDs) and, therefore, cannot be individually referenced like objects from other objects.

Object Databases - Overview of Object Database Concepts

Literals

The literal types supported by the Object Model are

- **Single-valued or atomic types** where each value of the type is considered as an atomic (indivisible) single value.
- **Struct (or tuple) constructor** which is used to create standard structured types, such as the tuples (record types) in the basic relational model.
- **Collection (or multivalued) type constructors** which include the set(T), list(T), bag(T), array(T), and dictionary(K,T) type constructors.

Object Databases - Overview of Object Database Concepts

Single-valued or atomic types

- This includes the basic built-in data types such as integer, string, char, floating-point number, date, enumerated type and Boolean.
- For examples some atomic types relevant to the Employee could be defined as given below:

  Fname: **string**;
  Lname: **string**;
  Empid: **char(05)**;
  Birth_date: DATE;
  Address: **string**;
  Gender: **char**;
  Salary: **float**;

Object Databases - Overview of Object Database Concepts

Struct (or tuple) constructor

- A structured type is made up of several components and is similar to tuples/record types in the basic relational mode. So sometimes structured type is referred to as a compound or composite type.
- The struct constructor is a type generator, because many different structured types can be created.
- For example, the following structured type can be created for the composite attribute EmpName with three components FirstName, MiddleName and LasName or the Date attribute with its components such as Year, Month and Day.

struct EmpName<FirstName: string, MiddleName: string,
LastName: string>

Object Databases - Overview of Object Database Concepts

Collection (or multivalued) type constructors

This enables to represent a collection of elements, which themselves could be of any literal or object type. The collection literal types supported by the Object Model include set, bag, list, array, and dictionary.

A set is an unordered collection of elements $\{i_1, i_2, \dots, i_n\}$ of the same type without any duplicates.

A bag (also called a multiset) is an unordered collection of elements similar to a set except that the elements in a bag may contain duplicates.

Object Databases - Overview of Object Database Concepts

Collection (or multivalued) type constructors (contd..)

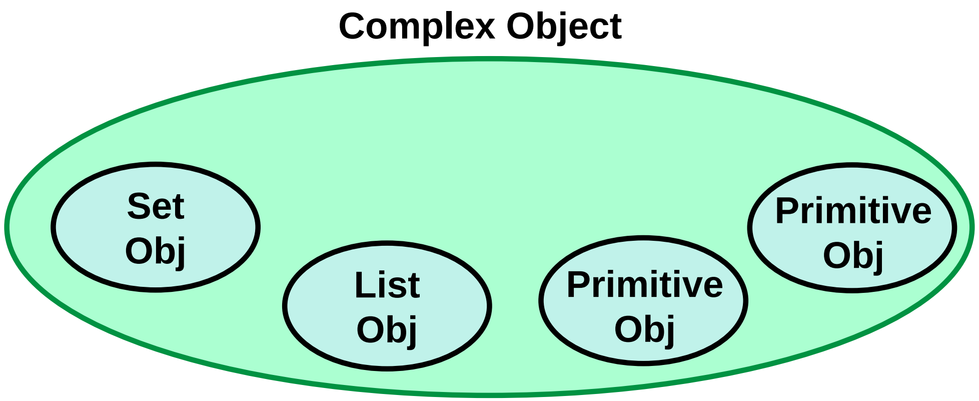
In contrast to sets and bags, a list is an ordered collection of elements of the same type. A list is similar to a bag except that the elements in a list are ordered.

An array is a dynamically sized ordered collection of elements that can be located by position. The difference between array and list is that a list can have an arbitrary number of elements whereas an array typically has a maximum size.

A dictionary is an unordered sequence of key-value pairs (K, V), without any duplicates. The value of a key K can be used to retrieve the corresponding value V.

Object Databases - Overview of Object Database Concepts

The literal types supported by the object model enables to define complex objects which may consist of other objects as illustrated by the following Department example..



```
define type DEPARTMENT
tuple (  Dname: string;
         Dnumber: integer;
         Mgr: tuple ( Manager: EMPLOYEE;
                      Start_date: DATE; );
         Locations: set(string);
         Employees: set(EMPLOYEE);
         Projects: set(PROJECT); );
```

Atomic Type

Tuple Type

Set Type

Activity

State the correct answer by filling the provided spaces.

1. The major components of an object are _____, _______ .
2. OOPL has two object categories based on their existence, namely ________ and ________.
3. In ____________ each value of the type is considered as an atomic (indivisible) single value.
4. _______ constructor is used to create standard structured types.
5. Collection or ______ type constructors which include the set(T), list(T), bag(T), array(T), and dictionary(K,T) type constructors.

Object Databases - Overview of Object Database Concepts

Encapsulation

- The concept of encapsulation is applied to database objects and it is the mechanism that binds together code and the data it manipulates.
- Thus, encapsulation defines the behavior of a type of object based on the operations that can be externally applied to objects of that type.
- Encapsulation can bring a form of **data and operation independence**. Therefore, encapsulation is encouraged by defining the operation in two parts namely **signature** and **method**.

Object Databases - Overview of Object Database Concepts

Encapsulation

- **Signature/Interface** of the operation - Specifies the name of the operations and its arguments (parameters).
- **Method/Body -** Specifies the implementation of the operation.

- External programs pass messages to the objects, to invoke operations which includes the operation name and the parameters.
- Thus, encapsulation restricts the direct access to the values of an object as the values are required to be accessed through the pre-defined methods.

Object Databases - Overview of Object Database Concepts

Encapsulation

- For database applications, the requirement that all objects be completely encapsulated is too strict.
- This requirement is relaxed by dividing the structure of an object into **visible** and **hidden** attributes (instance variables).
- Visible attributes can be seen by and are directly accessible to the database users and programmers via the query language.
- The hidden attributes of an object are completely encapsulated and can be accessed only through predefined operations.

Object Databases - Overview of Object Database Concepts

Class

- The term **class** is often used to refer to a type definition, along with the definitions of the operations for that type.
- The relevant operations are declared for each class, and the signature (interface) of each operation is included in the class definition.
- A method (implementation) for each operation must be defined elsewhere using a programming language.

Object Databases - Overview of Object Database Concepts

Class

- The operations which would be defined in a class may include the
  - **object constructor** operation (often called *new*), which is used to create a new object
  - **destructor** operation, which is used to destroy (delete) an object.
  - A number of **object modifier** operations can also be declared to modify the states (values) of various attributes of an object.
  - Additional operations can **retrieve** information about the object.

Object Databases - Overview of Object Database Concepts

The following example shows how the type definitions can be extended with operations to define classes.

```
               define class DEPARTMENT
                 type tuple ( Dname: string;
                              Dnumber: integer;
                              Mgr: tuple ( Manager: EMPLOYEE;
                                           Start_date: DATE; );
                              Locations: set (string);
                              Employees: set (EMPLOYEE);
                              Projects set(PROJECT); );
                 operations  no_of_emps: integer;
                             create_dept: DEPARTMENT;
                             delete_dept: boolean;
                             assign_emp(e: EMPLOYEE): boolean;
                         (* adds an employee to the department *)
                             remove_emp(e: EMPLOYEE): boolean;
                         (* removes an employee from the
               department *)
                 end DEPARTMENT;
```

An operation is applied to an object by using the **dot notation**. For example**, if d is a reference to a DEPARTMENT** object, an operation such as **no_of_emps** can be invoked by writing **d.no_of_emps**.

Object Databases - Overview of Object Database Concepts

- **Encapsulation of Operations**

```
define class Student
     type tuple ( Firstname: string;
                  Lname: string;
                  NIC: string;
                  Birth_date: DATE;
                  Address: string;
                  Gender: char;
                  Dept: DEPARTMENT; );
operations        age: integer;
                  create_stue: Student;
                  destroy_stu: Boolean;
end Student;
```

Activity

Define a class to store following details about a vehicle.

Registration_No: String, Color: String, Engine_capacity: integer, No_of_seats: integer, No_of_wheels

Operations - Start: boolean, Speed: float, Stop: Boolean

Object Databases - Overview of Object Database Concepts

- **Persistence of Objects**
  - Transient objects - in OOPL, objects only exist while the program is running. Hence, these objects are known as transient objects.
  - Persistent objects - objects that are exist even after the termination of the program.
  - There are two mechanism to make an object persistence.

    i) Naming Mechanism

    ii) Reachability Mechanism

Object Databases - Overview of Object Database Concepts

Persistence of Objects

i) Naming Mechanism

- This mechanism assigns a name to an object which is unique within database. An operation or a statement can be used to specify the name.
- Users and applications perform database access through the named persistent objects which are used as entry points to the database.
- However, it is not practical to name all objects in a large database that includes thousands of objects. Therefore, most objects are made persistent by using the second mechanism, called reachability.

Object Databases - Overview of Object Database Concepts

Persistence of Objects

ii) Reachability Mechanism

- The reachability mechanism works by making the object reachable from some other persistent object.

- An object B is said to be reachable from an object A if a sequence of references in the database lead from object A to object B.

Object Databases - Overview of Object Database Concepts

For example, to make the DEPARTMENT object persistent the following steps are to be followed:

- create a class DEPARTMENT_SET whose objects are of type set(DEPARTMENT)
- create an object of type DEPARTMENT_SET and give it a persistent name ALL_DEPARTMENTS for example. ALL_DEPARTMENTS is a named object that defines a persistent collection of objects of class DEPARTMENT. In the object model standard, ALL_DEPARTMENTS is called the **extent**.
- Any DEPARTMENT object that is added to the set of ALL_DEPARTMENTS by using the *add_dept* operation becomes persistent by virtue of its being reachable from ALL_DEPARTMENTS

```
define class DEPARTMENT_SET
                  type set (DEPARTMENT);
operations add_dept(d: DEPARTMENT): boolean;
             (* adds a department to the
            DEPARTMENT_SET object *)
            remove_dept(d: DEPARTMENT): boolean;
            (* removes a department from the
            DEPARTMENT_SET object *)

            create_dept_set:  DEPARTMENT_SET;
            destroy_dept_set: boolean;
  end Department_Set;
…
```

Class DEPARTMENT_SET is defined as a collection of DEPARTMENT objects.

persistent collection of objects of class DEPARTMENT known as Extent

```
persistent name ALL_DEPARTMENTS:
                                 DEPARTMENT_SET;
(* ALL_DEPARTMENTS is a persistent named object of
type DEPARTMENT_SET *)
…
d:= create_dept;
(* create a new DEPARTMENT object in the variable d *)

…
b:= ALL_DEPARTMENTS.add_dept(d);
```

DEPARTMENT object d is added to the set of ALL_DEPARTMENTS by using the add_dept operation.

```
define class DEPARTMENT
type tuple ( Dname: string;
                Dnumber: integer;
                Mgr: tuple ( Manager: EMPLOYEE;
                Start_date: DATE; );
                Locations: set (string);
                Employees: set (EMPLOYEE);
                Projects set(PROJECT); );
Operations
        no_of_emps: integer;
        create_dept: DEPARTMENT;
        destroy_dept: boolean;
        assign_emp(e: EMPLOYEE): boolean;
      (* adds an employee to the department *)
        remove_emp(e: EMPLOYEE): boolean;
(* removes an employee from the department *)
end DEPARTMENT;
```

Object Databases - Overview of Object Database Concepts

- It is necessary to understand the difference between relational databases and ODBs in persistence aspect.
- In relational databases, *all* objects are assumed to be persistent. Hence, when a table such as DEPARTMENT is created in a relational database, it represents both the *type declaration* for DEPARTMENT and a *persistent set* of *all* DEPARTMENT records (tuples).
- In the OO approach, a class declaration of DEPARTMENT specifies only the type and operations for a class of objects. Object persistency is required to be defined separately through type set(DEPARTMENT) whose value is the *collection of references* (OIDs) to all persistent DEPARTMENT objects.

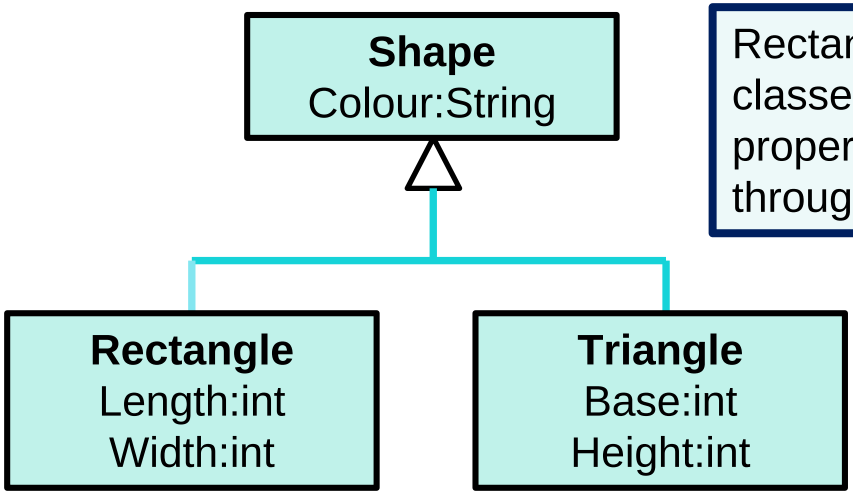
Object Databases - Overview of Object Database Concepts

- Thus, object model allows transient and persistent objects to follow the same type and class declarations.
- Moreover, it is possible to define several persistent collections for the same class definition, if needed.

Object Databases - Overview of Object Database Concepts

Type Hierarchies and Inheritance

- Type/class Hierarchies and Inheritance are two key features in Object database. Inheritance direct to type/class hierarchies.
- **Inheritance** is a key concept in Object model which allows to inherit structure and/or operations of previously defined classes.
- Inheritance promotes the reuse of existing type definitions and ease the incremental development of data types.



Rectangle and Triangle classes acquire the property of Shape class through Inheritance.

Object Databases - Overview of Object Database Concepts

Type Hierarchies and Inheritance

- **Functions -** in the basic level of inheritance the term “functions” is used to refer both attributes and operations together since attributes resemble functions with zero arguments.
- Since attributes and operations are handled in a similar manner in the basic level of inheritance, an attribute value or a value returned from an operation would be referred using a function name.

Object Databases - Overview of Object Database Concepts

Type Hierarchies and Inheritance

- With the function concept a type is specified as given below:

  Type_Name: Function1, Function2, Function3,..

  Student: Name, NIC, Birth_date, Age, Gender

- Implementation of the functions *Name, NIC, DoB, Gender* can be implemented as **stored attributes** while function *Age* can be implemented as an operation that calculates the Age from the value of the DOB attribute and the current date.

Object Databases - Overview of Object Database Concepts

Type Hierarchies and Inheritance

- **Subtypes & Supertypes:** The concept of subtype is useful when it is necessary to create a new type/class that is similar but not identical to an already defined type/class.
- Suppose that the type PERSON is already defined as follows:

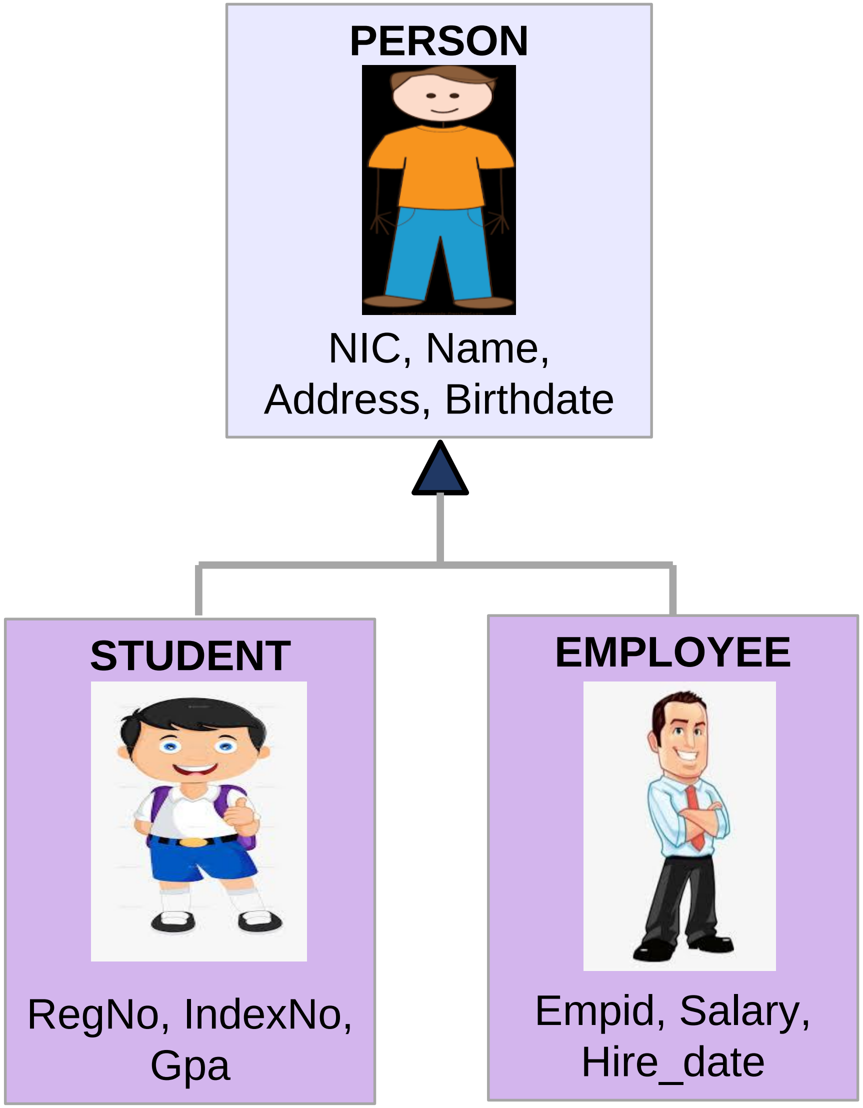
  PERSON: NIC, Name, Address, Birthdate

- Consider that two new types EMPLOYEE and STUDENT are to be defined as given below:

  EMPLOYEE: NIC, Name, Address, Birthdate, **Empid, Salary, Hire_date**

  STUDENT: NIC, Name, Address, Birthdate, **RegNo, IndexNo, Gpa**

Object Databases - Overview of Object Database Concepts



PERSON

NIC, Name, Address, Birthdate

STUDENT

RegNo, IndexNo, Gpa

EMPLOYEE

Empid, Salary, Hire_date

Attributes of Student

(NIC, Name, Address, Birthdate, RegNo, IndexNo, Gpa)

Attributes of Employee

(NIC, Name, Address, Birthdate, Empid, Salary, Hire_date)

Object Databases - Overview of Object Database Concepts

Type Hierarchies and Inheritance

- It is possible to define EMPLOYEE and STUDENT as subtypes of PERSON and the type PERSON is referred to as the supertype.
- The subtype then inherits all the functions of the super type and it may have some additional functions as well.
- Thus, EMPLOYEE and STUDENT can be declared as given below:

  EMPLOYEE subtype-of PERSON: Empid, Salary, Hire_date

  STUDENT subtype-of PERSON: RegNo, IndexNo, Gpa

- Hence, it is possible to create a **type hierarchy** to show the supertype/subtype relationships among all the types declared in the system.

Object Databases - Overview of Object Database Concepts

Type Hierarchies and Inheritance

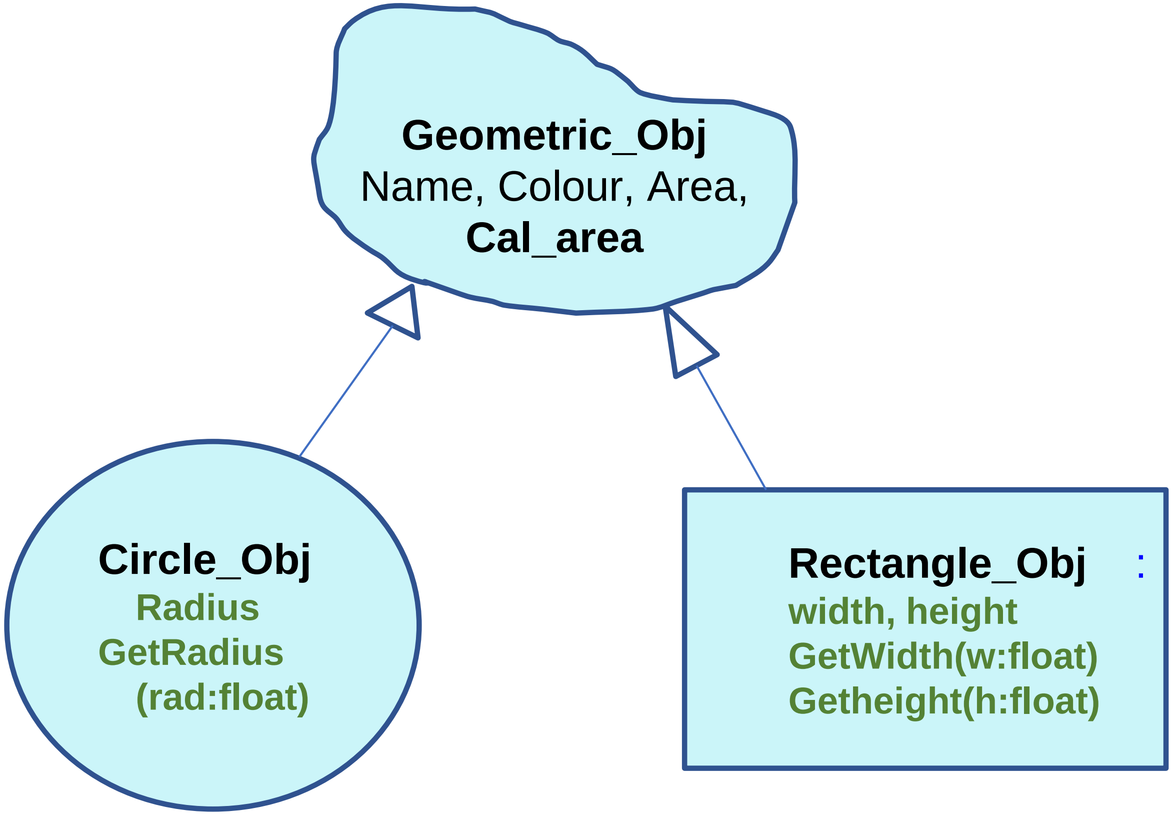
Geometric_Obj : Name, Colour, Area, Cal_area

Circle_Obj subtype-of **Geometric_Obj** : **radius**

Rectangle_Obj subtype-of **Geometric_Obj** : **width, height**

- In the above example Geometric_Obj is the supertype. Circle_Obj and Rectangle_Obj are subtypes that inherits all the functions of Geometric_Obj and in addition to that their additional functions.

Object Databases - Overview of Object Database Concepts



Activity

Toy: product_ID, colour, age_limit, battery, toy_type, price.

Create two subtypes Bear and Car of the Toy super type. Type Bear has greeted, sing and size as additional functions and type Car has controller function.

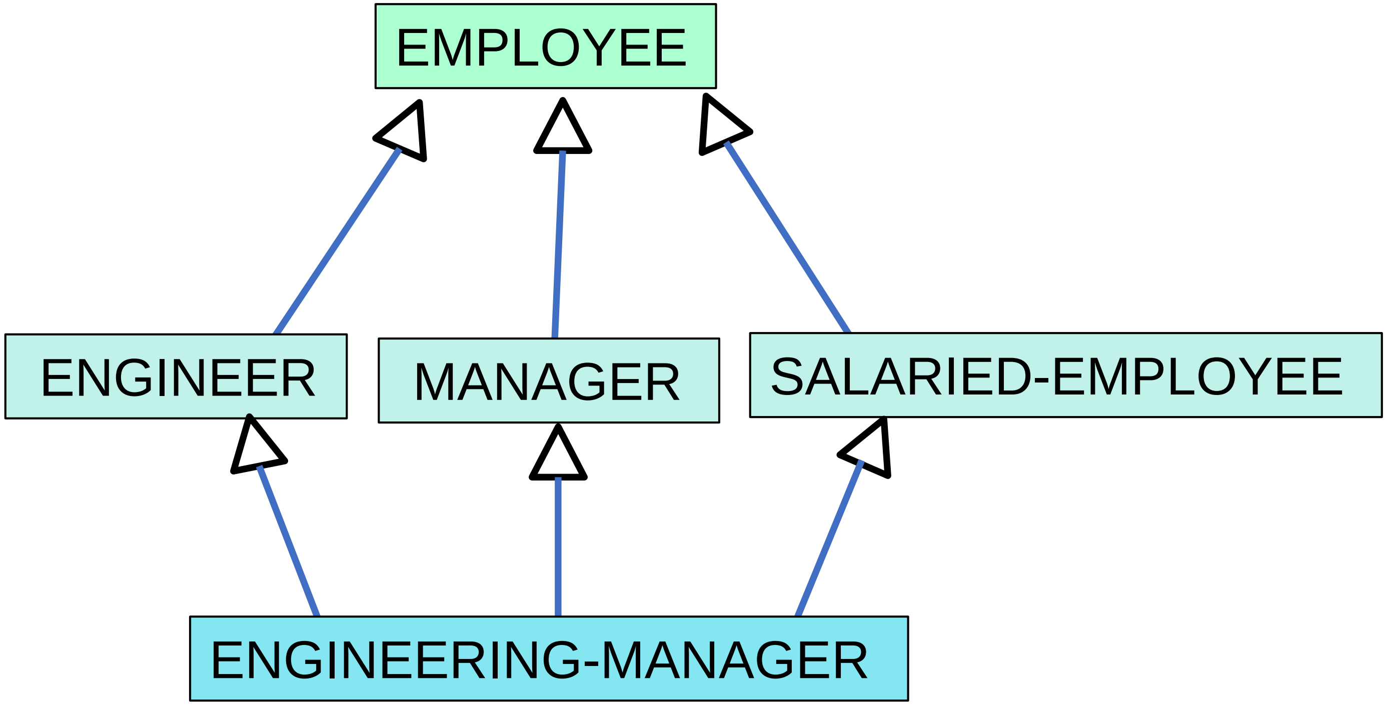
Bear	Car

Object Databases - Overview of Object Database Concepts

Multiple Inheritance

- Multiple inheritance takes place when a subtype inherits from two or more supertypes. In such cases, the subtype may inherit all the functions of all it’s supertypes.
- A subtype ENGINEERING_MANAGER that is a subtype of both MANAGER and ENGINEER.
- A **type lattice** is created as a result of multiple inheritance rather than a type hierarchy.

Object Databases - Overview of Object Database Concepts


EMPLOYEE

ENGINEER

MANAGER

SALARIED-EMPLOYEE

ENGINEERING-MANAGER

© e-Learning Centre, UCSC

Object Databases - Overview of Object Database Concepts

Problems with multiple inheritance

- **Ambiguity -** A subtype is inherited from two supertypes and both these supertypes have a function with similar name.
- Such cases present an ambiguity (doubtfulness) when implementing a particular function with similar name.
- For example, both MANAGER and ENGINEER may have a function called Salary. If MANAGER and ENGINEER supertypes have different implementations for salary function an ambiguity exists as to which of the two is inherited by the subtype ENGINEERING_MANAGER.

Object Databases - Overview of Object Database Concepts

Problems with multiple inheritance

Overcome Ambiguity

Given below are techniques for dealing with ambiguity in multiple inheritance.

- Check for ambiguity during the creation of subtype and let user/developer to specify the function which should be inherited.
- Use a system default.
- Check for name ambiguity and deny multiple inheritance if there is an ambiguity present. It is possible to request the user to change function names in supertypes to eliminate ambiguity.

Object Databases - Overview of Object Database Concepts

Selective inheritance

- Selective inheritance enables subtype to inherit only some of the functions of a supertype. Other functions are not inherited.
- The functions in a supertype that are *not* to be inherited by the subtype are defined through EXCEPT clause .
- The mechanism of selective inheritance is not facilitated in ODBs, but it is used more frequently in artificial intelligence applications.

Object Databases - Overview of Object Database Concepts

Polymorphism of operations / operator overloading

- The OO concept of representing one operation in multiple forms is known as Polymorphism. This concept is also known as operator overloading
- Polymorphism is derived from two greek words: poly and morphs. The word "poly" means many and "morphs" means forms. Thus, polymorphism means many forms.
- The implementation varies based on the object type that operation is applied to.
    - E.g: an operator to calculate area of a geometric object can be applied to different object types such as triangles, rectangles and circles. Implementation differs based on the object type.

Object Databases - Overview of Object Database Concepts

Polymorphism of operations / operator overloading

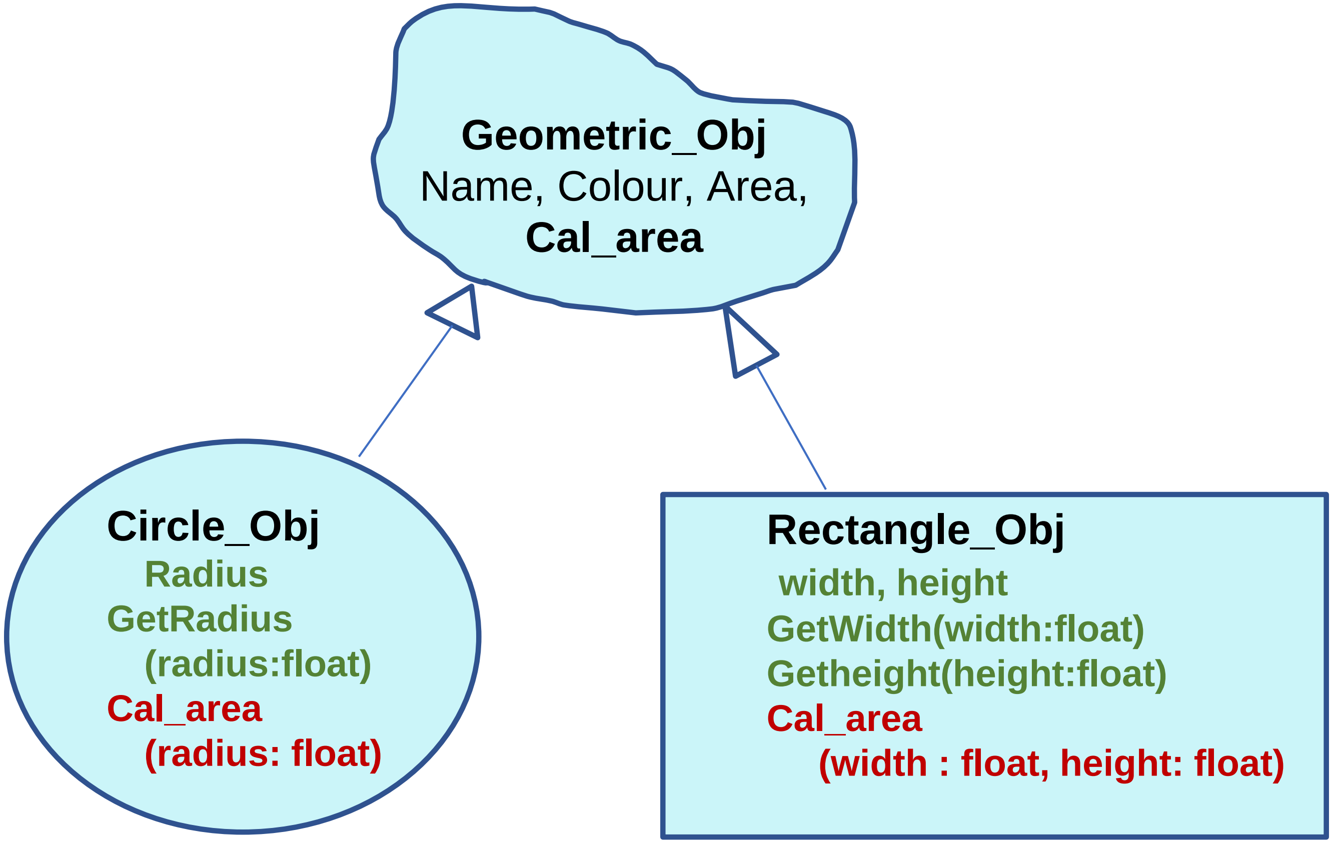
- The cal_area function to calculate the area of a circle or a rectangle has two different implementations. This is due to the fact that, area calculation of a circle is different to area calculation of a rectangle. Circle and Rectangle are two different types.

For Circle_Obj : **Cal_area (radius: float)**

For Rectangle_Obj: **Cal_area (width : float, height: float)**

In this example Cal_area operation is **overloaded by different implementations** .

Object Databases - Overview of Object Database Concepts



Geometric_Obj
Name, Colour, Area,
Cal_area

Circle_Obj
Radius
GetRadius
(radius:float)
Cal_area
(radius: float)

Rectangle_Obj
width, height
GetWidth(width:float)
Getheight(height:float)
Cal_area
(width : float, height: float)

Object Databases - Overview of Object Database Concepts

Polymorphism of operations / operator overloading

- **Static (Early) binding:** Defines the appropriate method at compile time. Example: for area calculation, the type of the object is known before compiling the program.
- **Dynamic (Late) binding:** Checks the object type during the program execution and then call the appropriate method. Example: for area calculation the appropriate Cal_area function is called based on the object type.

Activity

Match the correct phrase from the given list.

(Ambiguity, Naming mechanism, overloading, Type lattice, hidden attributes, late binding)

1. A problem with multiple inheritance
2. A result of multiple inheritance
3. Call the appropriate method during the runtime
4. Attributes are encapsulated and access through predefined operations
5. A mechanism to make an object persistence

XML Databases - Reason for the Origination of XML

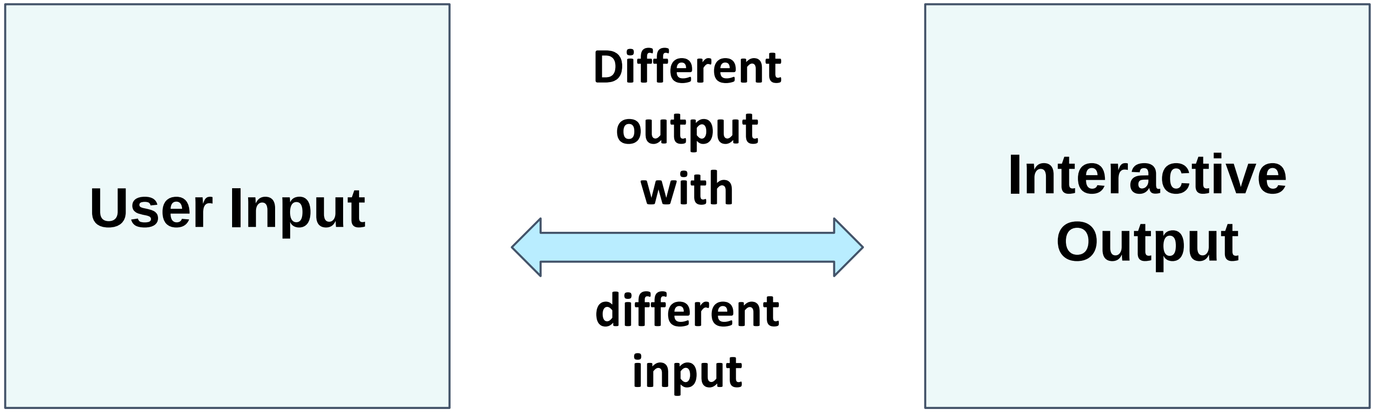
- XML - Extensible Markup Language
- Same as desktop applications are connected to local databases for running them, web applications have interfaces that connect them with data sources.
- Data sources can be used to access information in web applications.
- Formatting web pages is done using Hypertext Documents
- Eg: HTML

XML Databases - Reason for the Origination of XML

- HTML - Not a good method to represent structured data
- Languages that can structure and exchange data on web applications;
  - XML
  - JSON
- XML - Describes the meaning of the data and how they are structured in the web pages on a text document itself. **(Self Describing Documents)**
- Eg: Attribute Names, Attribute Values

XML Databases - Reason for the Origination of XML

- Basic HTML - Static Web Pages
- Dynamic Web Pages



XML - Transfer self describing text documents in dynamic web pages

Activity

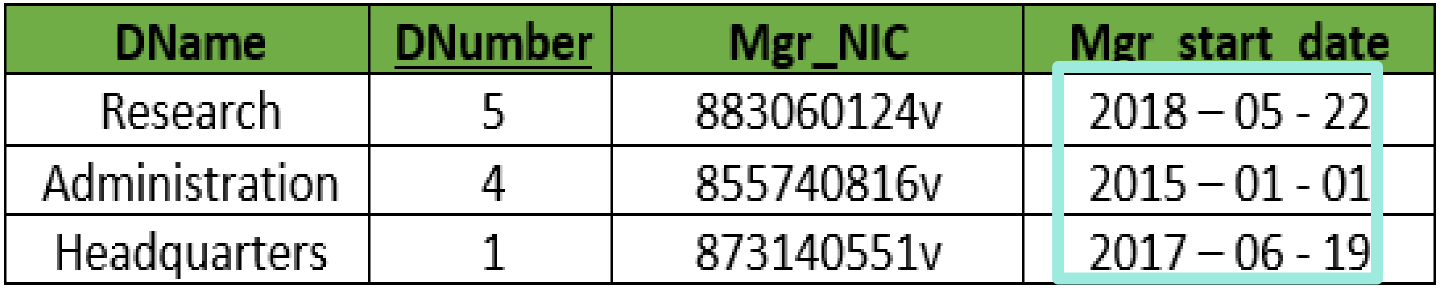
State whether the following statements are true or false.

1. HTML is a good method to represent structured data.	
2. A self describing document mentions both the meaning of data and how they are structured.	
3. XML supports only the functioning of static web pages.	
4. Generating different outputs that match the different inputs is done with static data.	
5. Generating the specific order values for different customers in e commerce is an instant of dynamic data.	

XML Databases - Structured, Semi - Structured and Unstructured Data

Structured Data

- Information in Relational Databases - **Structured Data**
- Each Data Record in the database follows the same structure



DName	DNumber	Mgr_NIC	Mgr_start_date
Research	5	883060124v	2018 – 05 - 22
Administration	4	855740816v	2015 – 01 - 01
Headquarters	1	873140551v	2017 – 06 - 19

- Structure is decided by the database schema

XML Databases - Structured, Semi - Structured and Unstructured Data

Semi - Structured Data

- Some data is collected in unplanned situations. Therefore, all the data may not have the same format.
- Data may have a certain structure. However, not all the data has the same structure

  E.g: Some have additional attributes; Some attributes maybe present only in some data.

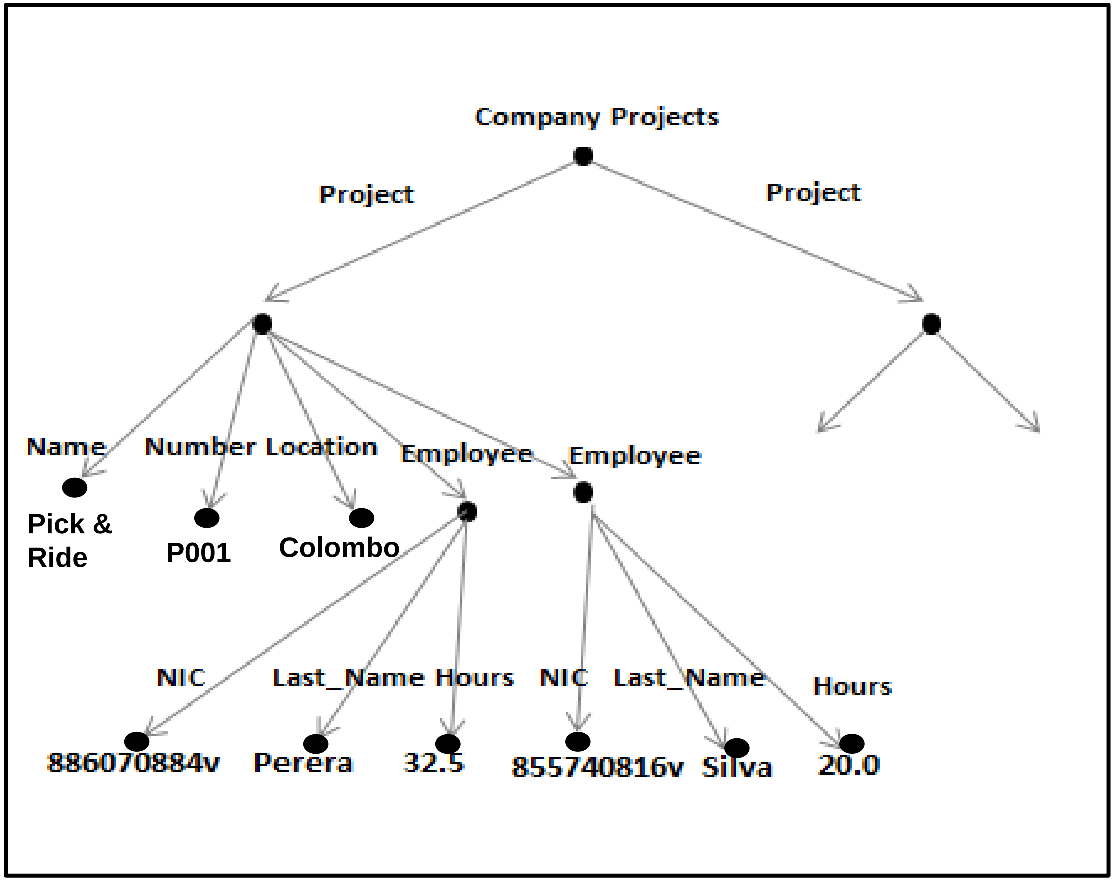
- No predefined schema
- Data model to represent semi - structured data

  - Tree data structure

  - Graph data structure

XML Databases - Structured, Semi - Structured and Unstructured Data

- Semi - Structured Data as a Directed Graph



XML Databases - Structured, Semi - Structured and Unstructured Data

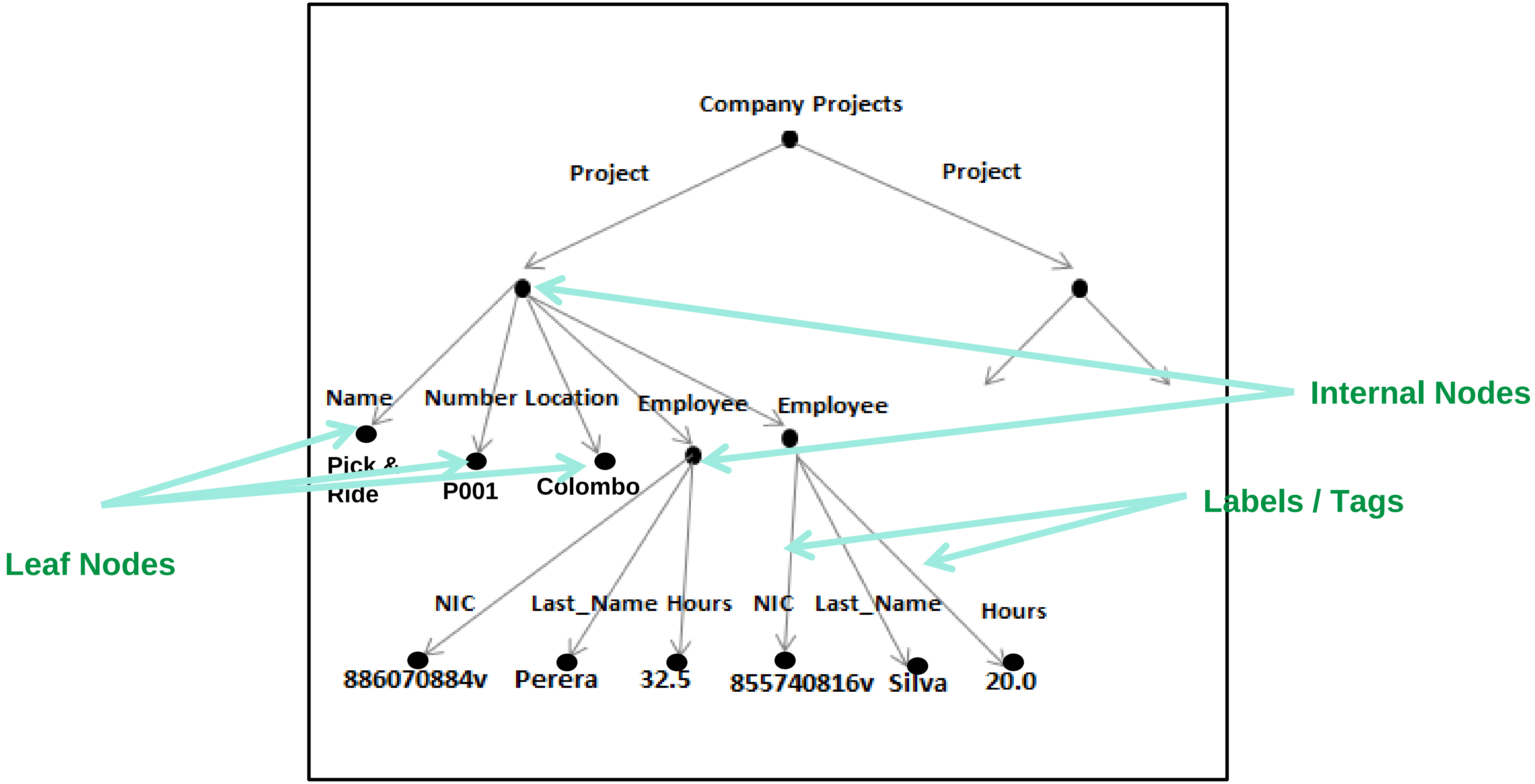
Semi - Structured Data

- Schema information and data values are represented as mixed.

  i.e.: Names of attributes, Relationships etc. are mixed with data values because the different entities will have different attributes.

XML Databases - Structured, Semi - Structured and Unstructured Data

Semi - Structured Data as a Directed Graph



Internal Nodes

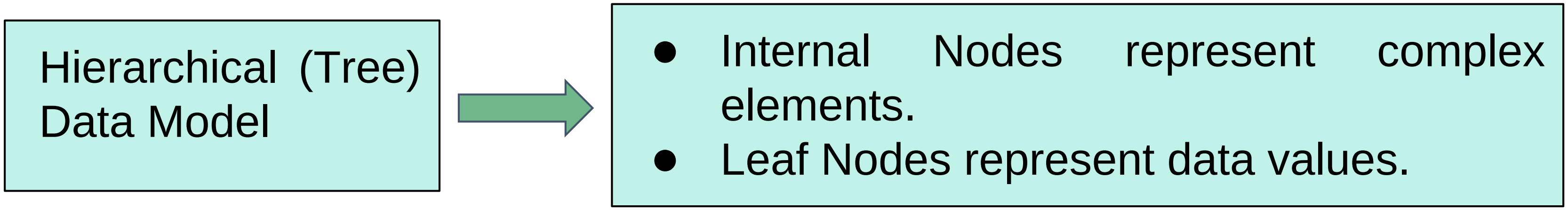
Labels / Tags

Leaf Nodes

XML Databases - Structured, Semi - Structured and Unstructured Data

Semi - Structured Data as a Directed Graph

- Semi structured data is represented as a directed graph as given below:
  - Labels/Tags represent Schema Names (names of attributes, object types, relationships).
  - Internal nodes represent Objects or Composite attributes
  - Leaf Nodes represent actual data values



Hierarchical (Tree) Data Model

- Internal Nodes represent complex elements.
- Leaf Nodes represent data values.

XML Databases - Structured, Semi - Structured and Unstructured Data

Unstructured Data

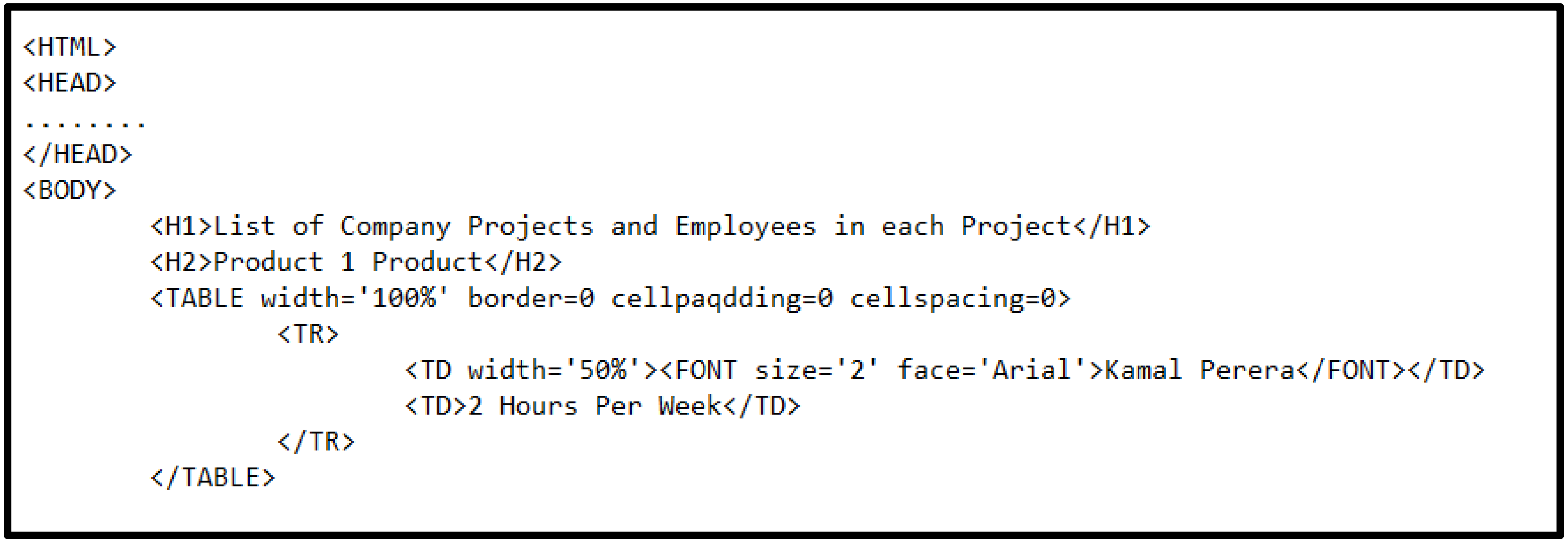
- Almost has no specification of the type of data

  eg: Web pages designed using HTML

- <p></p> - Outputs whatever data inside the two tags regardless of the meaning of them.
- <p><b></b></p> - Data formatting is also mixed with data values.

XML Databases - Structured, Semi - Structured and Unstructured Data

Unstructured Data: eg: - HTML document with unstructured data



```
<HTML>
<HEAD>
........
</HEAD>
<BODY>
        <H1>List of Company Projects and Employees in each Project</H1>
        <H2>Product 1 Product</H2>
        <TABLE width='100%' border=0 cellpaqdding=0 cellspacing=0>
                <TR>
                        <TD width='50%'><FONT size='2' face='Arial'>Kamal Perera</FONT></TD>
                        <TD>2 Hours Per Week</TD>
                </TR>
        </TABLE>
```

XML Databases - Structured, Semi - Structured and Unstructured Data

- **Unstructured Data**
  - HTML documents are harder to analyze and interpret using software since they do not have schema information on the type of data.
  - However, with the shifting of the human needs to an online environment, it is required to have correct interpretations on data presented online and exchange these data.
  - This need gave rise to the use of XML in online data manipulation environments.

Activity

Select the correct type of data for the following.

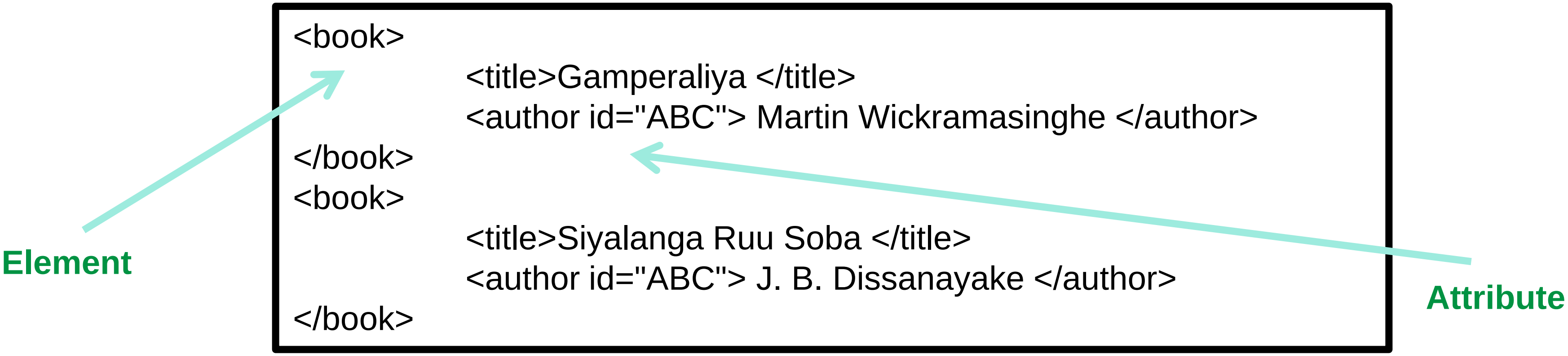
(Structured data, Semi structured data, Unstructured data)

1. Geo Spatial and Weather data at different conditions
2. Customer transaction database of a retail outlet
3. Email data
4. XML Data
5. Data generated by robots at intense environments
6. Student details database of a school management system

XML Databases - XML Hierarchical (Tree) Data Model

Constructing an XML Document

- XML comprises of XML objects
- Structure of an XML document includes:
  - Elements
  - Attributes



```
<book>
                <title>Gamperaliya </title>
                <author id="ABC"> Martin Wickramasinghe </author>
</book>
<book>
                <title>Siyalanga Ruu Soba </title>
                <author id="ABC"> J. B. Dissanayake </author>
</book>
```

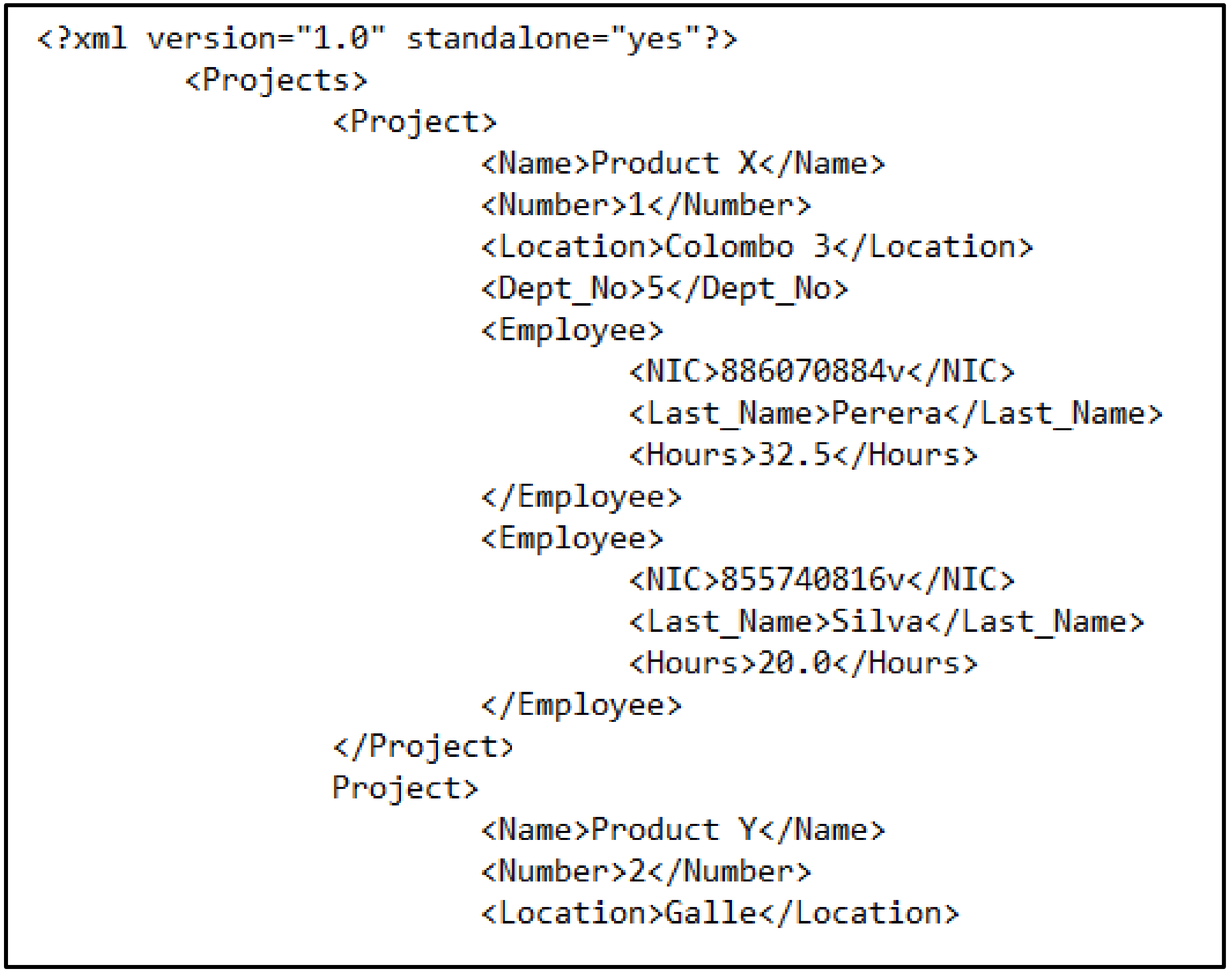
- Same as in HTML, in XML elements and attributes have a starting tag and an ending tag
- Tag name is defined inside the angle's brackets used.

XML Databases - XML Hierarchical (Tree) Data Model

- A major difference between XML and HTML is that HTML tags define how the text is to be displayed whereas XML tag names describe the meaning of the data elements in the document.
- This makes it possible to process the data elements in the XML document automatically by computer programs.
- Also, the XML tag (element) names can be defined in another document, known as the *schema document*, to give a semantic meaning to the tag names that can be exchanged among multiple programs and users.
- In HTML, all tag names are predefined and fixed; that is why they are not extendible.

XML Databases - XML Hierarchical (Tree) Data Model

- **Constructing an XML Document**



```
<?xml version="1.0" standalone="yes"?>
        <Projects>
                <Project>
                        <Name>Product X</Name>
                        <Number>1</Number>
                        <Location>Colombo 3</Location>
                        <Dept_No>5</Dept_No>
                        <Employee>
                                <NIC>886070884v</NIC>
                                <Last_Name>Perera</Last_Name>
                                <Hours>32.5</Hours>
                        </Employee>
                        <Employee>
                                <NIC>855740816v</NIC>
                                <Last_Name>Silva</Last_Name>
                                <Hours>20.0</Hours>
                        </Employee>
                </Project>
                Project>
                        <Name>Product Y</Name>
                        <Number>2</Number>
                        <Location>Galle</Location>
```

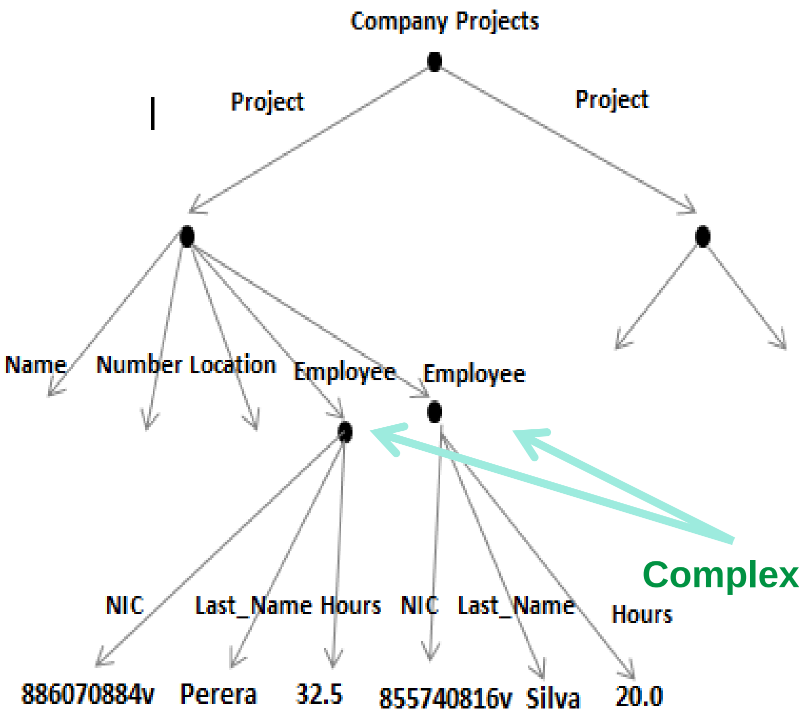
XML Databases - XML Hierarchical (Tree) Data Model

Constructing an XML Document

- **Simple Elements** include data values
- **Complex elements** are created from other elements by the hierarchy they are presenting
- Tag names are created such that the meaning of the data inside the tag is conveyed (In HTML the formatting is presented by the tag name)
- A document named **Schema document** is prepared separately to convey the meanings of the created tags.
- **Schemaless XML Documents -** XML documents that do not have a predefined element names are called Schemaless XML documents. They do not have a hierarchical tree structure.

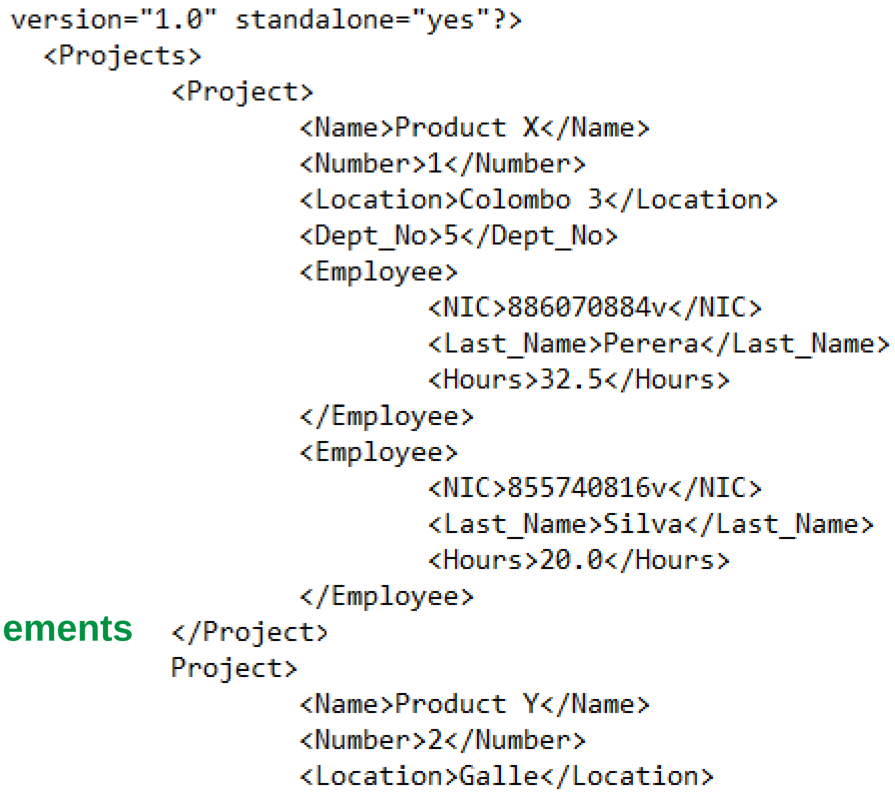
XML Databases - XML Hierarchical (Tree) Data Model

- **Constructing an XML Document**



Complex Elements

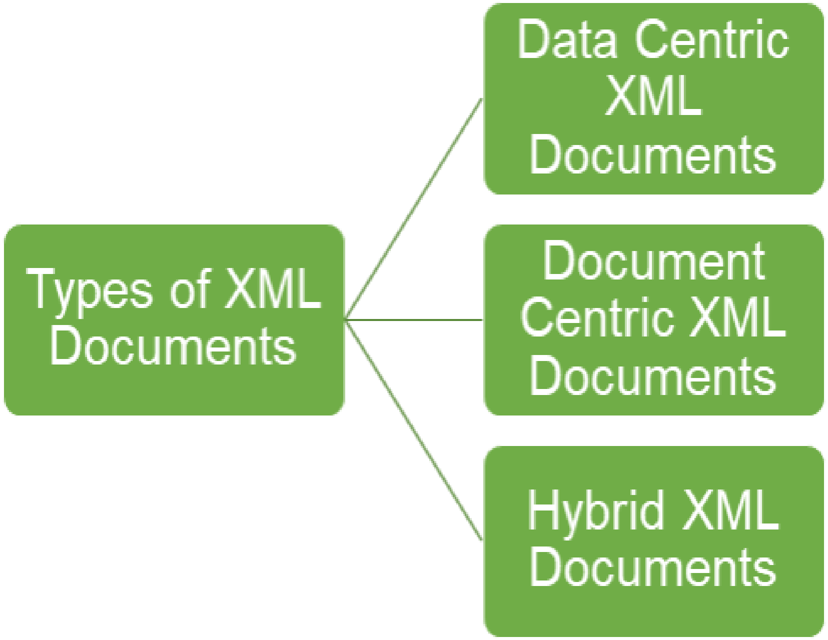
Simple Elements



```
version="1.0" standalone="yes"?>
  <Projects>
          <Project>
                  <Name>Product X</Name>
                  <Number>1</Number>
                  <Location>Colombo 3</Location>
                  <Dept_No>5</Dept_No>
                  <Employee>
                          <NIC>886070884v</NIC>
                          <Last_Name>Perera</Last_Name>
                          <Hours>32.5</Hours>
                  </Employee>
                  <Employee>
                          <NIC>855740816v</NIC>
                          <Last_Name>Silva</Last_Name>
                          <Hours>20.0</Hours>
                  </Employee>
          </Project>
          Project>
                  <Name>Product Y</Name>
                  <Number>2</Number>
                  <Location>Galle</Location>
```

XML Databases - XML Hierarchical (Tree) Data Model

There are three main types of XML documents:



Types of XML Documents

Data Centric XML Documents

Document Centric XML Documents

Hybrid XML Documents

Includes small data items with the purpose of transferring over the web with a predefined structure.

Includes large amounts of text data with less structure.

Includes both text and structured data.

XML Databases - XML Hierarchical (Tree) Data Model

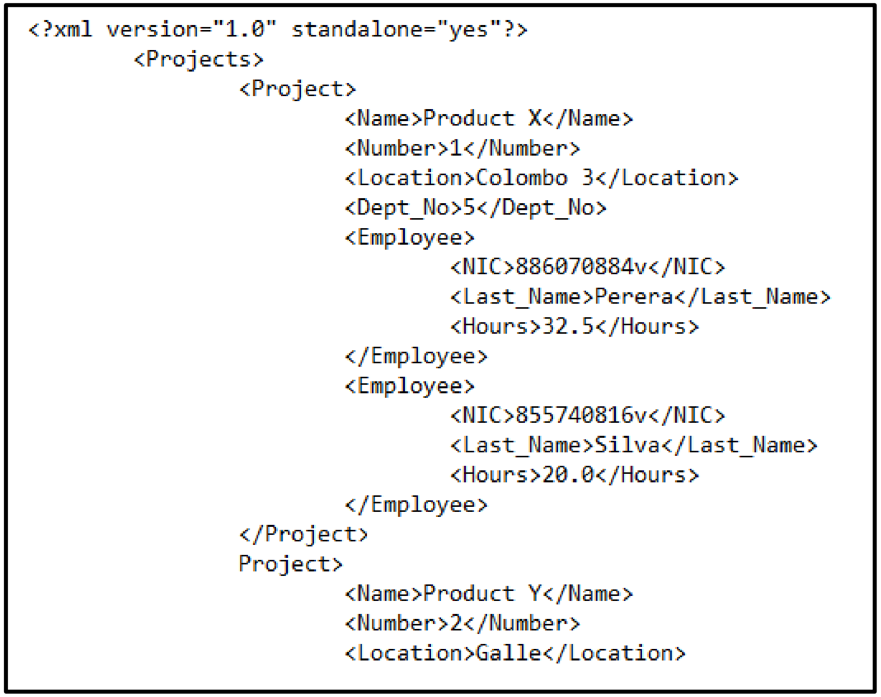
Data-centric XML documents

- Data-centric XML documents usually have a *predefined schema* that defines the tag names.
- These documents contain many small data items that follow a specific structure and hence may be extracted from a structured database. Thus, data-centric XML is used to mark up highly structured information such as the textual representation of relational data from databases.
- Since data-centric XML documents represent structured information in its textual representation, these documents are used to exchange data over the Web. Therefore, web services are about data-centric uses of XML.

XML Databases - XML Hierarchical (Tree) Data Model

Data-centric XML documents

- The XML includes many different types of tags and there is no long-running text.
- The tags are organized in a highly structured manner and the order and positioning of tags matter, relative to other tags. For example, <Hours> is defined under <Employee>, which should be defined under <Project>. The <Projects> tag is used only once in the document.



```
<?xml version="1.0" standalone="yes"?>
        <Projects>
                <Project>
                        <Name>Product X</Name>
                        <Number>1</Number>
                        <Location>Colombo 3</Location>
                        <Dept_No>5</Dept_No>
                        <Employee>
                                <NIC>886070884v</NIC>
                                <Last_Name>Perera</Last_Name>
                                <Hours>32.5</Hours>
                        </Employee>
                        <Employee>
                                <NIC>855740816v</NIC>
                                <Last_Name>Silva</Last_Name>
                                <Hours>20.0</Hours>
                        </Employee>
                </Project>
                Project>
                        <Name>Product Y</Name>
                        <Number>2</Number>
                        <Location>Galle</Location>
```

XML Databases - XML Hierarchical (Tree) Data Model

Document-centric XML documents

- These documents have large amounts of text, such as news articles or books.
- There are few or no structured data elements in these documents.
- The usage rules for tags are very loosely defined and they could appear pretty much anywhere in the document:

XML Databases - XML Hierarchical (Tree) Data Model

Document-centric XML documents

<H1>Types of XML documents</H1>

<P>It is possible to characterize <B>three main types </B> of XML documents</P>

<LIST>

<ITEM>Data-centric XML documents.</ITEM>

<ITEM>Document-centric XML documents</ITEM>

<ITEM>Hybrid XML documents</ITEM>

</LIST>

<P>If an XML document conforms to a predefined XML schema or DTD then the document can be considered as <LINK HREF=“structuredXML.xml"> structured data </LINK> </P>

XML Databases - XML Hierarchical (Tree) Data Model

Hybrid XML documents

- These documents may consist of both structured data and textual or unstructured data.
- They may or may not have a predefined schema.

Activity

Select the correct term for the description.

(Composite elements, Leaf nodes, Document centric XML documents, Schema document, Schemaless XML documents)

Represents data values in the hierarchical (tree) model	
Document prepared for XML tag definition	
Includes large volumes of text XML data	
Elements created from other elements	
Does not have a data definition for XML tags	

NoSQL Databases - Origins of NoSQL

- Relational database management systems (RDBMS) provide many advantages and thus are widely being used.
- However, there are number of issues as given below that RDBMS is not capable of handling. This gave rise to find an alternative ways of managing data.

  i) Impedance Mismatch (**Data Modeling issues**)

  ii) Drawbacks of shared database integration (**Data Modeling issues)**

  iii) Difficulties in scaling up to manage the growth in data volume (**Scalability & Availability issues**)

NoSQL Databases - Origins of NoSQL

Impedance Mismatch

- Impedance mismatch is the term used to refer to the dissimilarity between the relational database model and the programming language model (data structures in-memory).
- Although relational data model represents data as relations in tabular format which is simple, it introduces certain limitations. In particular, tabular representation cannot contain any structure, such as a nested record or a list.

NoSQL Databases - Origins of NoSQL

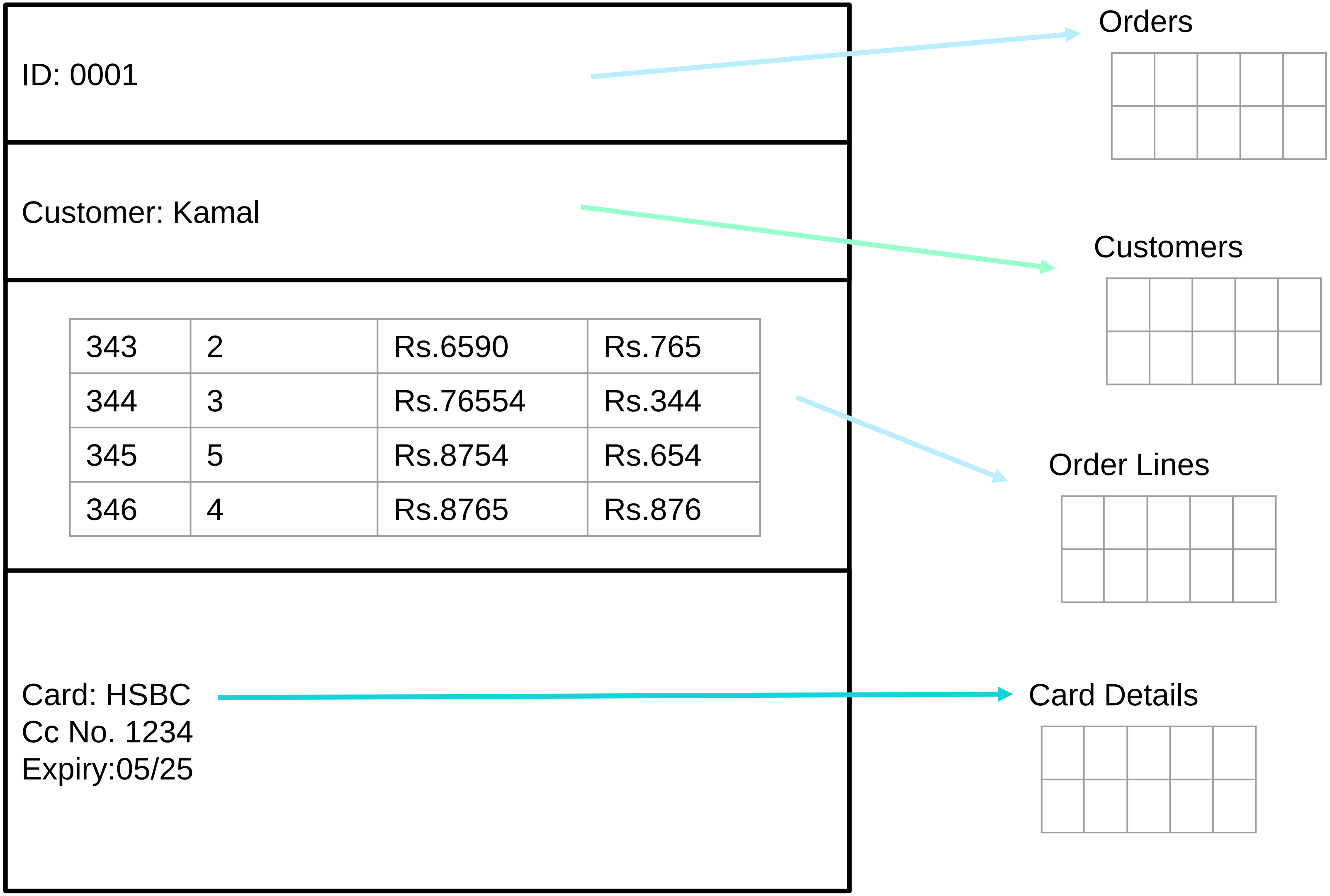
Impedance Mismatch

There are no such limitations for in-memory data structures, which can take on much richer structures than relations.

Consequently, a richer in-memory data structure is required to be translated into a relational representation to store it on disk.

These two different representations would cause the impedance mismatch which require translation from one representation to the other.

NoSQL Databases - Origins of NoSQL



ID: 0001

Customer: Kamal

343	2	Rs.6590	Rs.765
344	3	Rs.76554	Rs.344
345	5	Rs.8754	Rs.654
346	4	Rs.8765	Rs.876

Card: HSBC
Cc No. 1234
Expiry:05/25

NoSQL Databases - Origins of NoSQL

Impedance Mismatch

- Impedance mismatch is made much easier with the object-relational mapping frameworks, such as Hibernate that implement well-known mapping patterns.
- Although these mapping frameworks remove a lot of tedious work the mapping problem is still an issue since the problems in representing the in-memory data structures from the database perspective is not taken into consideration. This would hinder the query performance.

NoSQL Databases - Origins of NoSQL

Drawbacks of Shared Database Integration

- The database acts as an **integration database** with multiple applications, developed by separate teams, storing their data in a common database.
- To this end, the primary factor is the role of SQL as an integration mechanism between applications.
- Although shared database integration improves communication enabling all the applications to operate on a consistent set of persistent data there are certain drawbacks of database integration.

NoSQL Databases - Origins of NoSQL

Drawbacks of Shared Database Integration

The drawbacks of Database Integration are as given below:

- A structure that is designed to integrate many applications often become more complex than any single application needs.
- If an application requires to make changes to its data storage, it needs to coordinate with all the other applications using the database.
- Since different applications have different structural and performance needs, an index required by one application may hinder the performance of insert operations of another application.
- Usually, a separate team is responsible for each application. This means that the database cannot trust applications to update the data in a way that preserves database integrity. Thus, the responsibility of preserving database integrity is within the database itself.

NoSQL Databases - Origins of NoSQL

Drawbacks of Shared Database Integration

- A different approach is to treat the database as an **application database** which is directly accessed by only a single application codebase that is controlled by a single team.

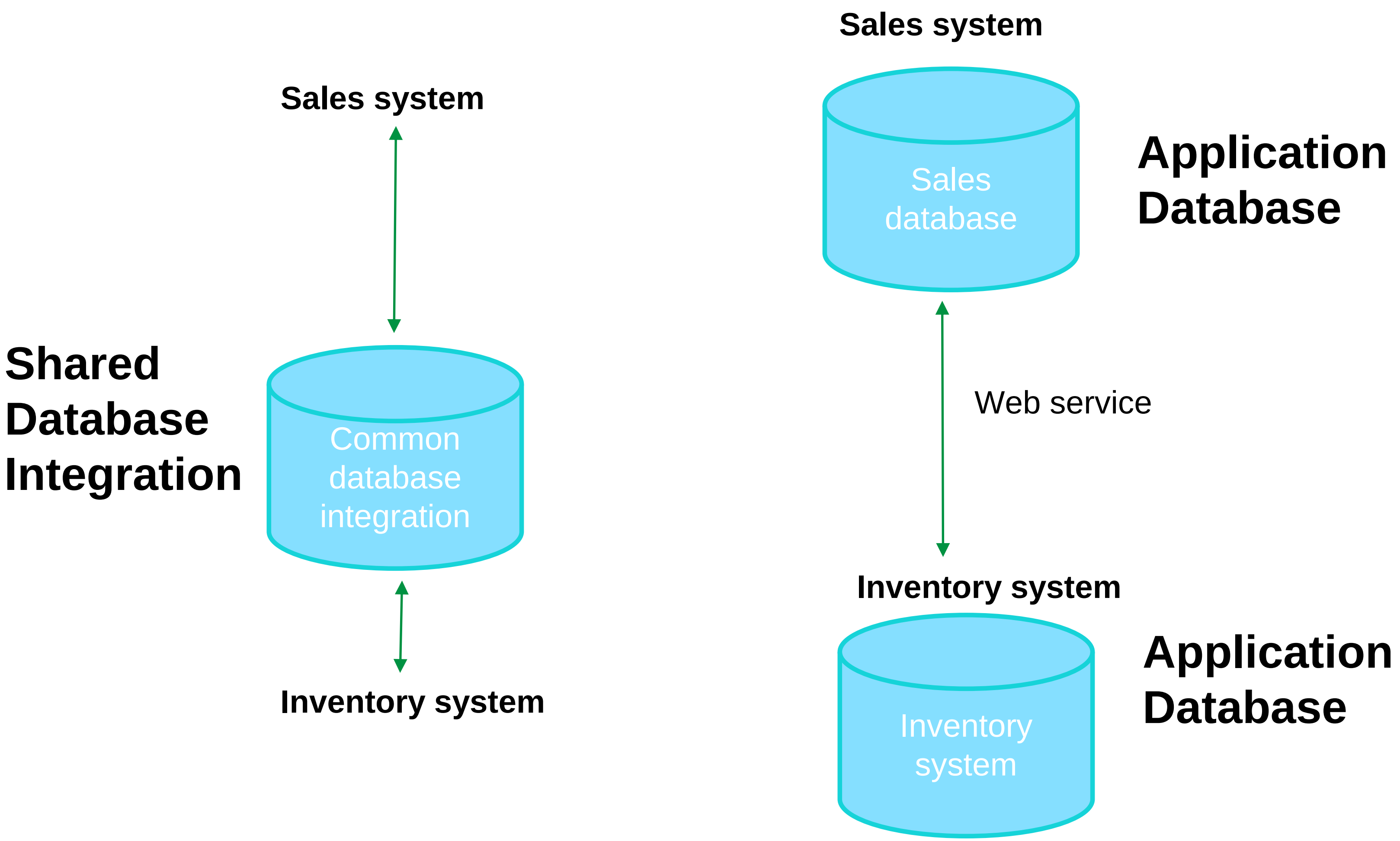
- With an application database, only the team using the application needs to know about the database structure, which makes it much easier to maintain and evolve the schema.

- Since the application team takes care of both the database and the application code, the responsibility for database integrity can be passed onto the application code.

- Web services (where applications would communicate over HTTP) enabled a new form of a widely used communication mechanism which challenged the use of SQL with shared databases.

NoSQL Databases - Origins of NoSQL

Shared Database Integration VS Application Database



Sales system

Shared Database Integration

Common database integration

Inventory system

Sales system

Sales database

Application Database

Web service

Inventory system

Inventory system

Application Database

NoSQL Databases - Origins of NoSQL

Advantages of Application Database

- Using web services as an integration mechanism under **application database** resulted in giving more flexibility for the structure of the data that was being exchanged.
- Communication with SQL, requires the data to be structured as relations. However, use of a web service, enables data structures with nested records and lists to be used.
- These data are usually represented as documents in XML or, more recently, JSON.

NoSQL Databases - Origins of NoSQL

Scalability & Availability issues

- When the large volumes of data is getting generated and the amount of data traffic is getting incremented, people understood that there is a requirement for more computer resources to store the data.
- It was suggested to go for big storage machines to scale up, and a higher processing power. However, more memory is needed to handle this kind of increase.
- When the sizes of machines increased, it becomes more expensive.
- Alternatively, a set of small machines which are working parallelly as a cluster could be used.

NoSQL Databases - Origins of NoSQL

Scalability & Availability issues

- A collection of small machines that are working in a cluster is comparatively cheaper than buying a bigger machine.
- It also provides high reliability and thus high availability since even when a single machine fails, the cluster will be up and running independent of the failures.

NoSQL Databases - Origins of NoSQL

NoSQL databases emerged mainly to provide the following advantages

a) Flexible data modeling: NoSQL emerged as a solution to the impedance mismatch problem between relational data models and object-oriented data models. NoSQL covers four different data organization models as given below which are highly-customizable to different businesses' needs.

i) **Document databases**: store data as documents similar to JSON (JavaScript Object Notation) objects. Each document has pairs of fields and values.

ii) **Key-Value databases**: represent a simpler type of database where each item contains keys and values. A value can only be retrieved by referencing its key and thus querying for a specific key-value pair is simple.

iii) **Column-Family databases**: store data in tables, rows, and in dynamic columns. This data model provides a lot of flexibility over relational databases because each row is not required to have the identical columns.

NoSQL Databases - Origins of NoSQL

iv) **Graph databases**: store data in nodes and edges representing a graph structure. Nodes for example store information about people, places, and things while edges store information about the relationships between the nodes.

The details of these data models are explained in detail in next slides.

b) **High Scalability**: NoSQL enables to manage growing data volumes since it is cluster-friendly. This feature makes it easily scalable, as large data sets can easily be split across clusters, with or without replication.

c) **High Availability**: NoSQL assures 100% database uptime. For businesses that process huge numbers of transactions, the goal of accessibility (which is related with the speed) is to have the data/page available quickly, even if it is not 100% accurate. This is due to the fact that if a data request does not produce a visible result fast, the user will navigate away, assuming something is wrong.

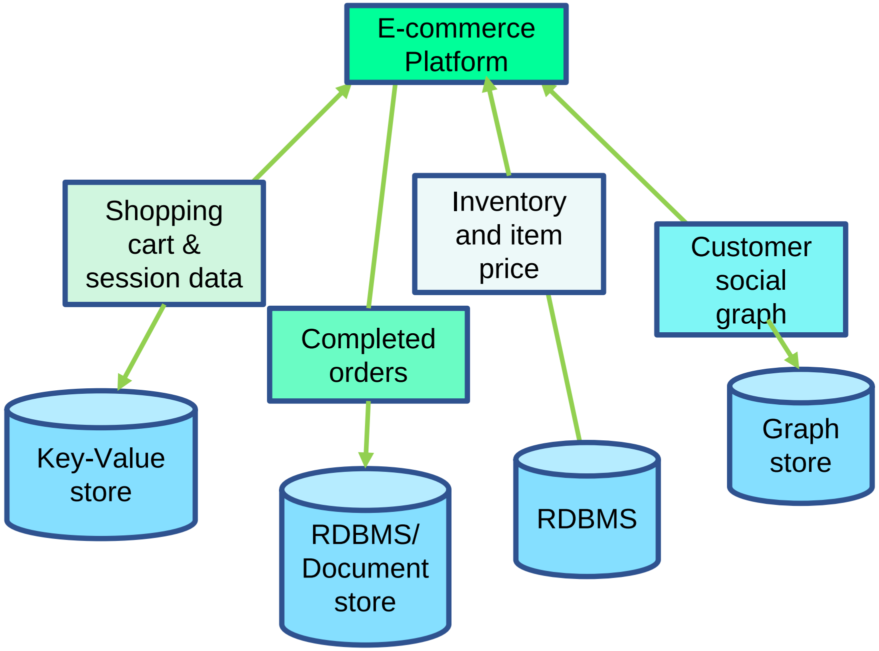
NoSQL Databases - Origins of NoSQL

Important rise of NoSQL with Polyglot Persistence

- Different kinds of data could best be dealt with different data storage methods. In short, it means picking the right data storage for the right use case.
- Rather than using Relational databases for different data types, it is possible to select a data storage based upon
  - the nature of data
  - the ways of manipulating it.
- **Polyglot Persistence** is to use many data storage technologies to suit the data requirement.

NoSQL Databases - Origins of NoSQL

Polyglot Persistency



Reference: https://www.jamesserra.com/archive/2015/07/what-is-polyglot-persistence/

Functionality	Considerations	Database Type
User Sessions	Rapid Access for reads and writes. No need to be durable.	Key-Value
Shopping Cart	High availability across multiple locations. Can merge inconsistent writes.	Document, (Key Value maybe)
Financial Data	Needs transactional updates. Tabular structure fits data.	RDBMS
POS Data	Depending on size and rate of ingest. Lots of writes, infrequent reads mostly for analytics.	RDBMS (if modest), Key Value or Document (if ingest very high) or Column if analytics is key.
Recommendations	Rapidly traverse links between friends, product purchases, and ratings.	Graph, (Column if simple)
Product Catalog	Lots of reads, infrequent writes. Products make natural aggregates.	Document

NoSQL Databases - Origins of NoSQL

Common characteristics of NoSQL databases

- NoSQL databases are not using SQL. However, there are some NoSQL databases which are using query languages equivalent to SQL. Those languages are easy to learn.
- Most of the NoSQL projects are open-source.
- Majority of the NoSQL databases have the ability of running on clusters.
- The capacity of running on clusters come with the effect on the data model as well as the degree of consistency provided.

NoSQL Databases - Origins of NoSQL

Common characteristics of NoSQL databases

- In relational databases, ACID properties are preserved and ensure the consistency across the whole database.
- However, the scenario in a clustered environment is different. In NoSQL environment, there are several levels for consistency and a range of options for distribution.
- There are certain NoSQL databases which do not facilitate in running on clusters.
- Graph database is one of the NoSQL data models that offer a distribution similar to relational databases which is more suitable for the data with complex relationships.
- NoSQL databases are schemaless which means, it is possible to add fields freely into the records except making changes in the defined structure.

NoSQL Databases - Data models in NoSQL

Introduction to Aggregate data models

- Relational model:- every operation is performed on tuples.
- Aggregate orientation:- operations can be performed on data in units. When comparing with tuples, it is a complex structure.
- It is possible to consider this structure as a record where we can operate other structures nested inside it.

NoSQL Databases - Data models in NoSQL

Introduction to Aggregate data models

- An aggregate is a set of related objects that is treated as a unit.
- Aggregates have the feature of making natural unit to help in creating replication and sharding.
- Because of that it is easy to operate as clusters.
- Aggregates also make the life of application programmers easy as they manipulate data through aggregate.

NoSQL Databases - Data models in NoSQL

Data model for complex relationship structures

- Aggregates are helpful as they put together data that is accessed together.
- However, related data could be accessed differently. Consider the following example.

  The connection between a client and the requests he gets for orders:

  - Applications will need to see the request history as they access the client. The possibility is joining the client with his request history to create an individual result/aggregate. Different applications, might need to handle orders as individual aggregates.

NoSQL Databases - Data models in NoSQL

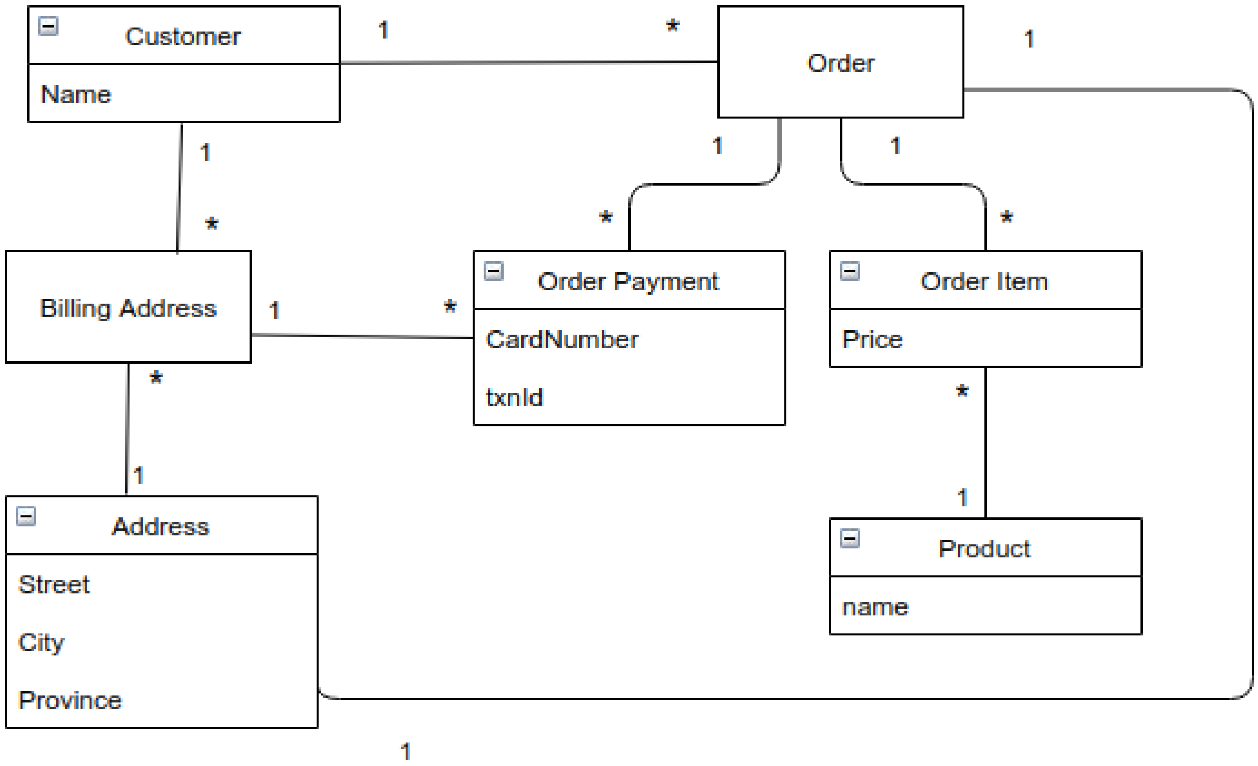
Data model for complex relationship structures

- The mechanism of handling updates is important when studying about relationships between aggregates.
- Aggregates are treated as the unit in the data retrieval process when it comes to the Aggregate-oriented databases.
- This treatment allows the property of Atomicity within the contents of aggregates.
- In Relational databases we can modify a set of records in the same transaction because ACID guarantees when we do such modifications.
- NoSQL databases came into the picture as there was a need to run on clusters. This paved the way to aggregate-oriented data models which have a high number of records but simple connections.

NoSQL Databases - Data models in NoSQL

E.g: - Suppose we are going to implement an e-commerce portal.

We will be storing data about orders, users, payments, product catalog, and a set of addresses. We will be using this data model in a relational environment.



© e-Learning Centre, UCSC

NoSQL Databases - Data models in NoSQL: Sample of data stored in the tables

Customer	
Id	Name
1	Shantha

Order		
Id	CustomerId	ShippingAddressId
99	1	77

Product	
Id	Name
27	Laptop

Billing Address		
Id	CustomerId	AddressId
55	1	77

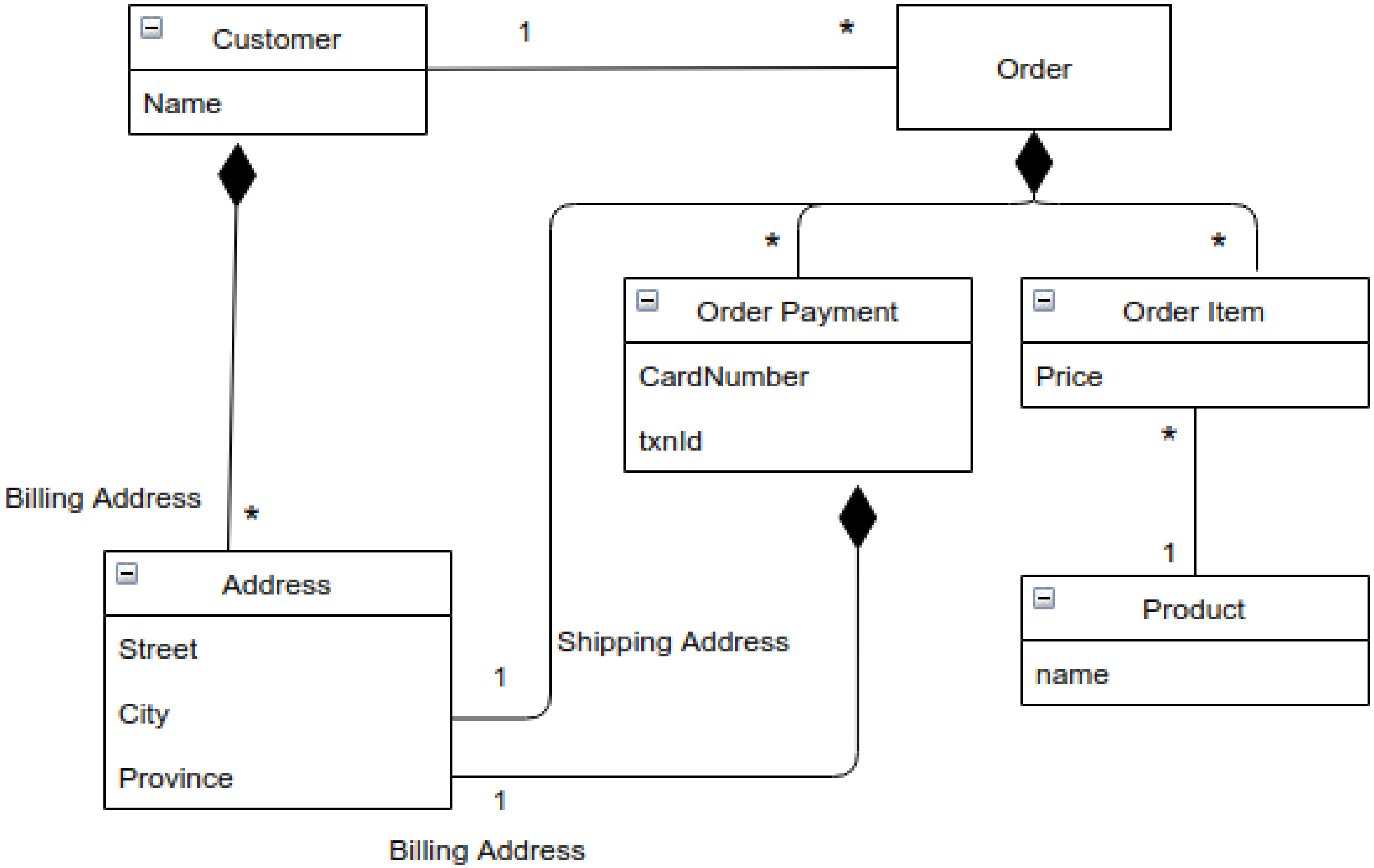
Order Item			
Id	OrderId	ProductId	Price
100	99	27	145000

Address	
Id	Name
77	Kandy

Order Payment				
Id	OrderId	Card Number	BillingAddressId	TxnId
33	99	34-886-89	55	act-564

NoSQL Databases - Data models in NoSQL

The same model can be represented like this, in an aggregate-oriented environment.



© e-Learning Centre, UCSC

NoSQL Databases - Data models in NoSQL

- The two main aggregates here are,
    - customer
    - order
- Aggregation structure is represented with the UML composition symbol (black color diamond).
- When we pay attention to the properties hold by each entity,
    - Customer -> billing addresses.
    - Order -> order items

      a shipping address

      payments.
- Payment -> billing address.

© e-Learning Centre, UCSC

NoSQL Databases - Data models in NoSQL

- In the example data, **A single logical address record** can be seen three times.
- But instead of using IDs, here in the aggregate oriented diagram, a single value is copied in several places.
- This is suitable for a scenario which we assure that either of the addresses do not change.
- In a relational database, when we want to keep the addresses without changing, we can add a new row.
- When using aggregates, a copy of the entire address structure can be used as we prefer.

NoSQL Databases - Data models in NoSQL

Reason for using Aggregate data models

- Aggregate orientation facilitate in running on clusters.
- When creating the data clusters, it is necessary to consider the count of nodes that needs to be accessed when retrieving data.
- Aggregates have an important consequence for transactions.
- In a relational database, different combination of rows from different tables is possible to be manipulated in one transaction.

NoSQL Databases - Data models in NoSQL

- **Key-Value Model and suitable Use Cases**
  - Lookup based on keys can be used to access and aggregate in a key-value store.
  - Queries to the database can be submitted based on the fields in the aggregate in a document database. Database creates indexes based on the content of an aggregate. Moreover, part of an aggregate can be retrieved.
  - The aggregate being stored could be any data structure; it does not have to be a domain object. (eg: Redis)
  - Redis supports storing lists, sets, hashes. It can also perform operations such as range, diff, union, and intersection.
  - Redis can be used in multiple ways rather than just a general key-value store because of these features.

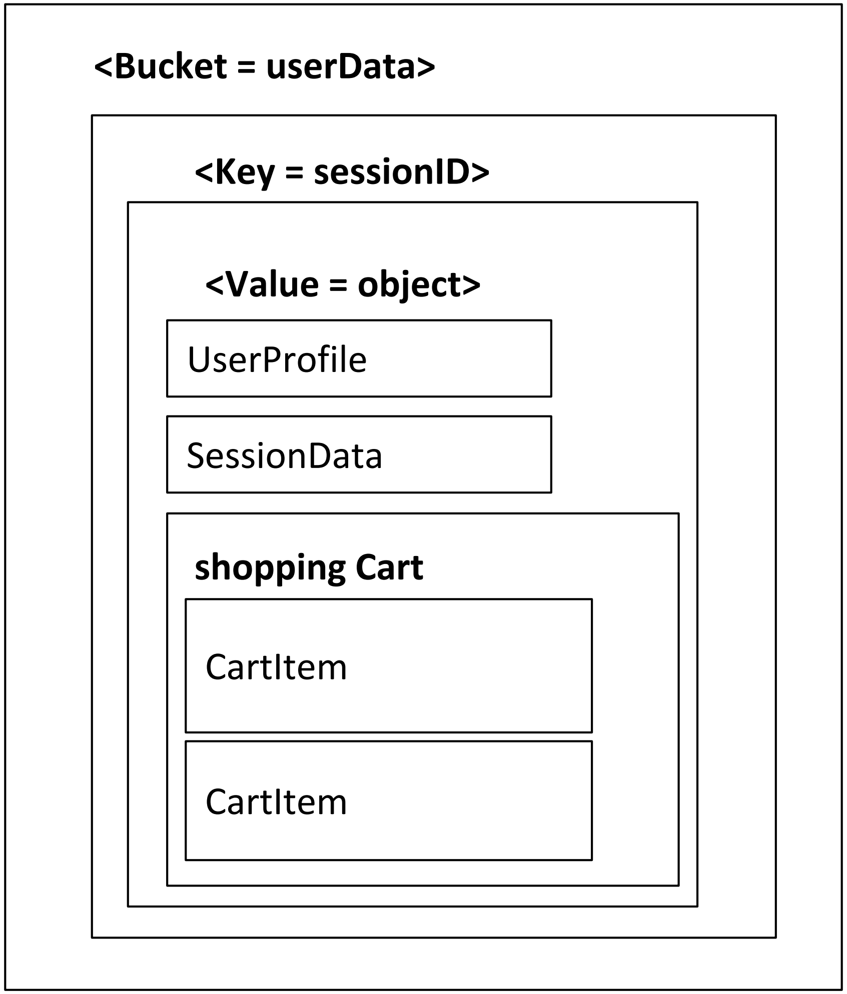
NoSQL Databases - Data models in NoSQL

Key-Value Model and suitable Use Cases

- There are many key-value databases.
- Riak, Redis, Oracle NoSQL are examples for key-value databases.
- In Riak database, the keys are segmented by storing in separate buckets.
- Consider a scenario where we want to store several information such as user session data, shopping cart data, and location data.
- We can store all of that information in one bucket as single key and single value objects.
- Now we have a single object with all the data that is stored in a single bucket.

NoSQL Databases - Data models in NoSQL

- Key-Value Model and suitable Use Cases



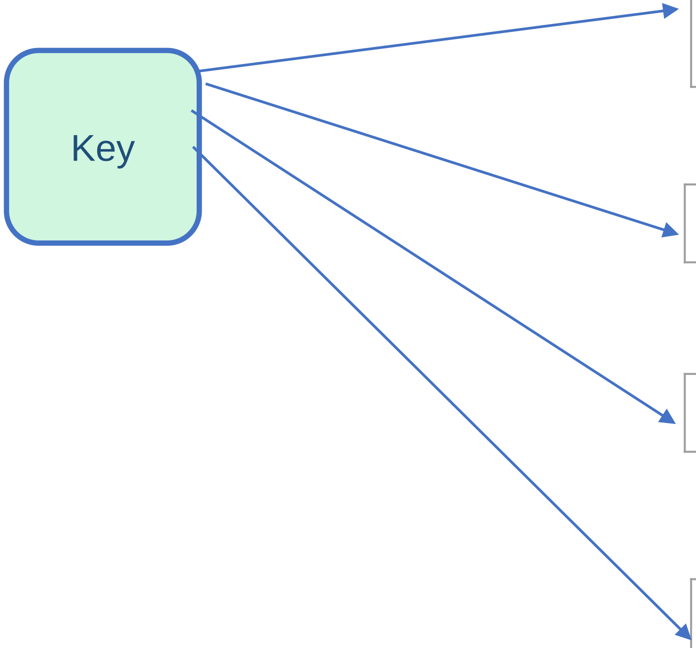
NoSQL Databases - Data models in NoSQL

Key-Value Model and its Use Cases

- Session Information Use Case
  - We know that web sessions are unique and each of those has a unique session- id value.
  - All most all the applications store that session-id while the applications use key-value stores get more benefits.
  - This is because all session details can be inserted using one PUT request and fetched using a GET.
  - It makes the processing faster since all the data on a single operation can be stored in one object.

NoSQL Databases - Data models in NoSQL

Key-Value Model Example



title	isbn	year
Sinhala Natyaye Wikashaya	111-444-3333	1867

111-444-3333	222-555-7777	222-555-777

111-444-3333	222-555-7777	222-555-777

Score 1867 111-444-3333	Score 1899 111-111-33	Score 1920 333-444-55

NoSQL Databases - Data models in NoSQL

Document Data Model and suitable Use Cases

- By adding a field called ID into a document database, it can be considered as a lookup of key-value pairs.
- It is allowed to add different structures of data into a key-value database.
- The main item in a document databases is the documents.
- It stores and retrieves documents such as XML, JSON, and BSON
- The features of these documents are:
  - self-describing
  - hierarchical data structures which consist of different types of values.

NoSQL Databases - Data models in NoSQL

Document Data Model and suitable Use Cases

- There’s a similarity in the stored documents.
- In document databases, documents are store as the value and the ID is stored as the key.
- It is similar to a key-value stores which has an examinable the value part.
- There can be different schemas in different documents.
- But in a relational database each row of a given table has to have the same schema.
- But here, these documents can belong to the same collection.

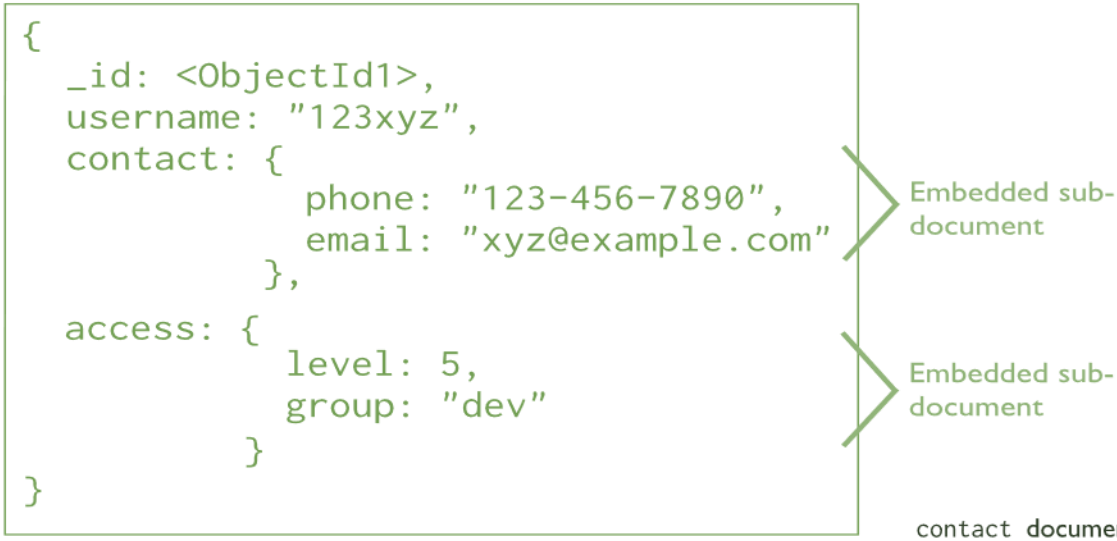
NoSQL Databases - Data models in NoSQL

Document Data Model and suitable Use Cases

- Use case of Logging of events
  - Different event logging needs of applications can be stored in a Document databases.
  - Document database will then become the central store of event data and it is important to be kept in dynamic situations.
  - Name of the application, type of event can be used when you want to share the event.

NoSQL Databases - Data models in NoSQL

Document Data Model Example

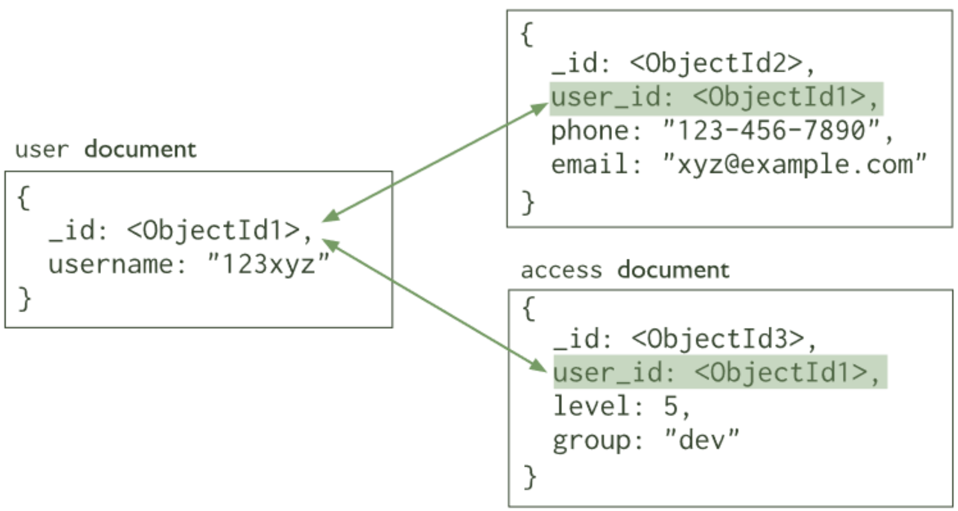


```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
          }
}
```

Embedded sub-document

Embedded sub-document

Relational model	Document model
Tables	Collections
Rows	Documents
Columns	Key/value pairs
Joins	Can be applied where necessary



contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

NoSQL Databases - Data models in NoSQL

Column-Family Stores and suitable Use Cases

- Column stores can be thought about as databases with a data model of the style of a big table.
- Even though rows act as the storage unit in database who helps write operations, there could be situations when writes are infrequent but multiple reads in a set of columns and rows are present.
- Using the method of storing columns groups for all rows as the fundamental unit of storage is the wise thing to do in such cases. Hence the reason for the name “column databases”.

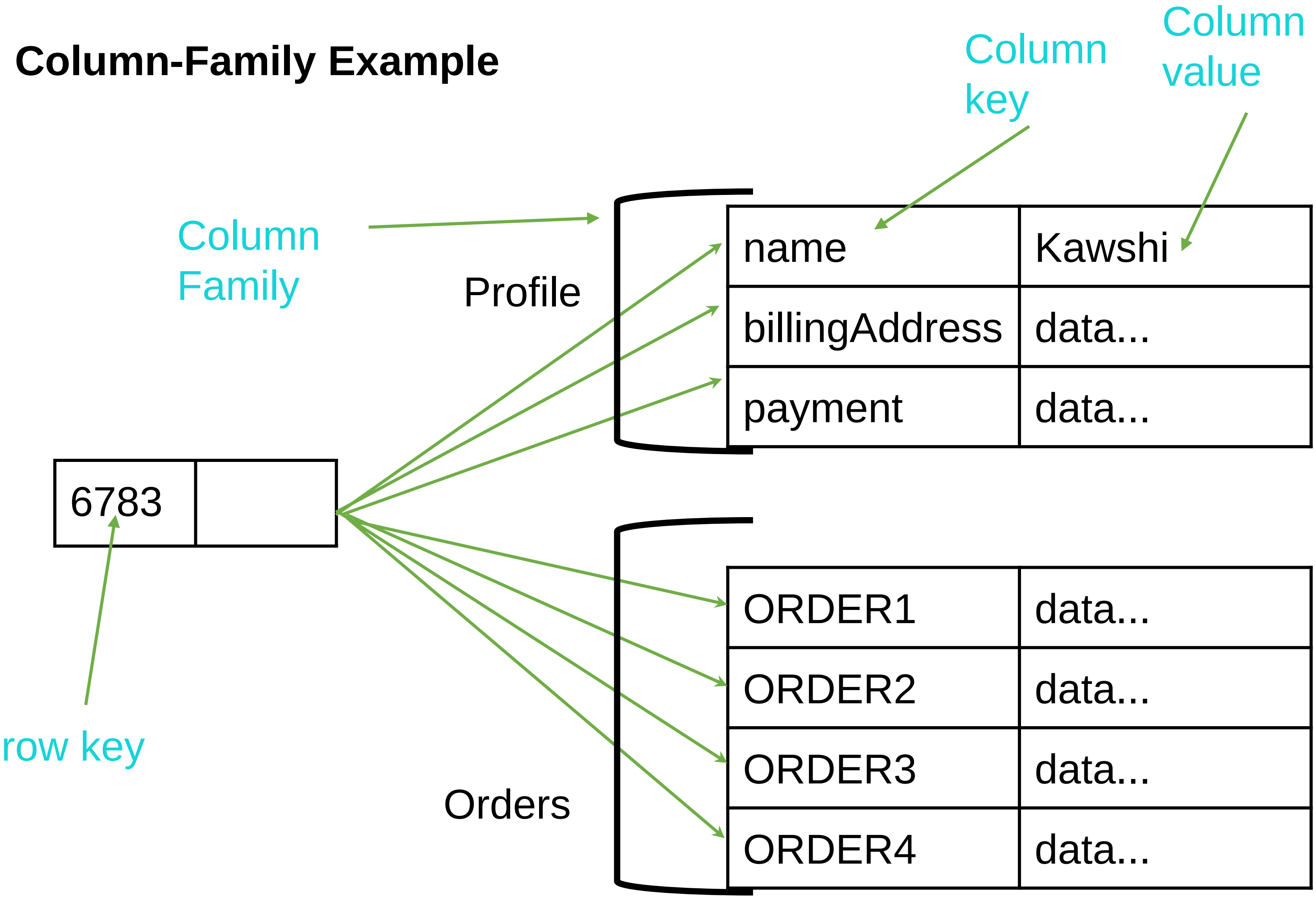
NoSQL Databases - Data models in NoSQL

Column-Family Stores and suitable Use Cases

- An aggregate structure which spans two levels can be thought of as a Column-family model.
- In key-value stores, the first key in key-value model is an identifier which uniquely recognises each row of our interest.
- This row aggregate is similar to a map which contains descriptive values.
- These values in the second level are what we call columns.

NoSQL Databases - Data models in NoSQL

Column-Family Example



NoSQL Databases - Data models in NoSQL

Column-Family Stores and suitable Use Cases

- Content Management Systems, Platforms used for blogs Use Case
  - Column families allow to:
    - Record blog entries with features such as tags, classifications, connections, and tracing options.
    - In addition to that, we can store comments in the same row or if we like, we can move them to a different key space.
  - Separate column-families can be used to record users of blog and the real blogs.

NoSQL Databases - Data models in NoSQL

Graph Data Stores

- Graph databases have a different model which is small records which have interconnections that are complex.
- Graph databases specialise in capturing interconnected information. But it’s much easier than reading these interconnections on a diagram could.
- Best uses are in extracting complex connections such as in social networks, users’ preferences for products, or rules for eligibility.
- Graph database has a data model which is nodes connected by edges (also called arcs).

NoSQL Databases - Data models in NoSQL

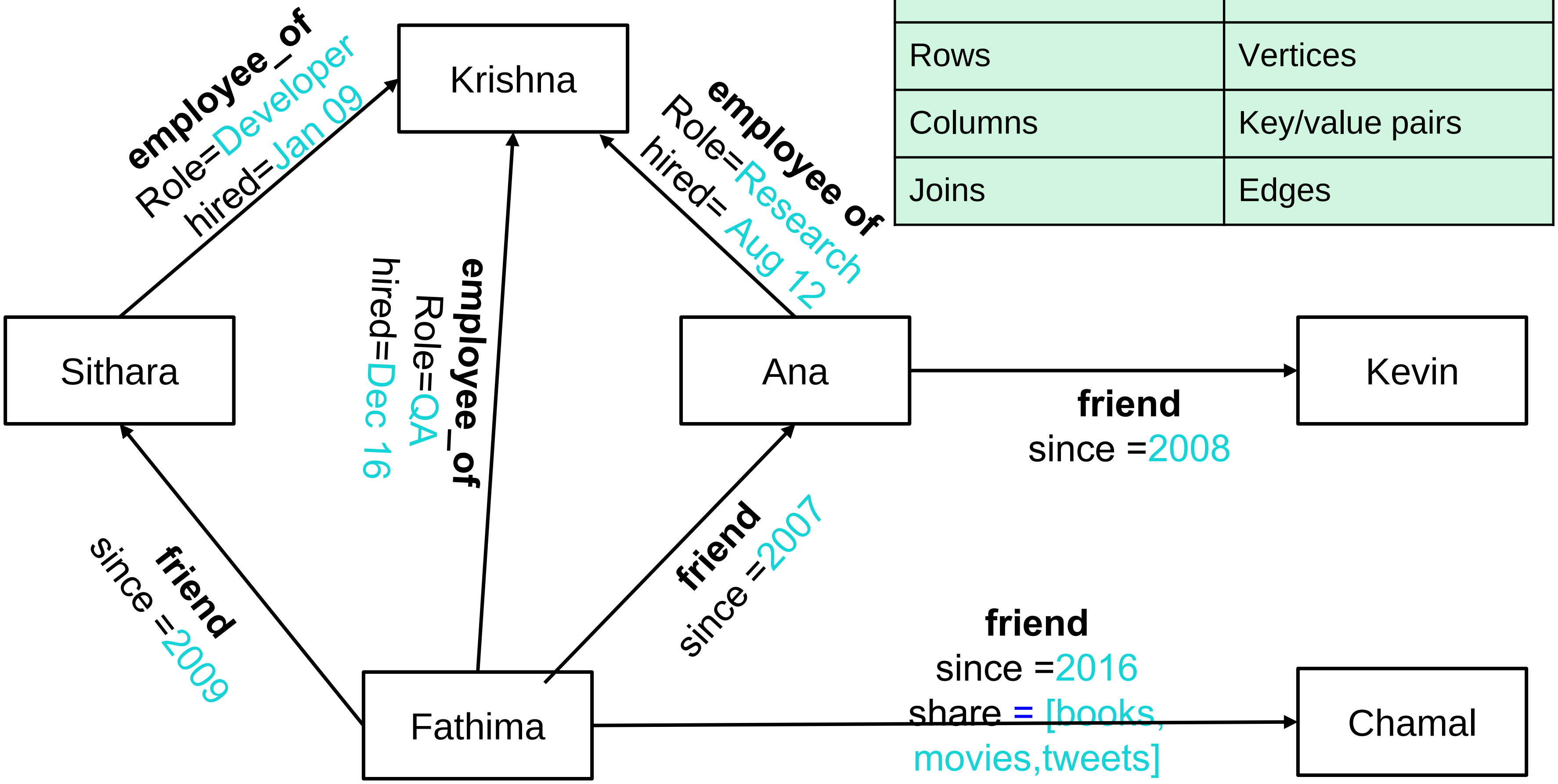
Graph Data Stores

- Graph databases enables to traverse the joins fast.
- Calculations for the relationship between nodes is not performed at query time. It is continued as a relationship itself.
- This is because traversing persisted relationships is efficient than performing the calculation on each individual query.
- There could be different types of relationships between nodes.

NoSQL Databases - Data models in NoSQL

Graph Data Stores

Relational model	Graph model
Tables	Vertices and Edges set
Rows	Vertices
Columns	Key/value pairs
Joins	Edges



Activity

Match the most relevant description, with the data models given.

Data Model	**Description**
Document	Store all key-value pairs together in a single namespace.
Key-value	Have a row key and within that, stores a combination of columns that fit together.
Column family	Enables to traverse the joins fast.
Graph data stores	Organise documents into groups called *collections*

Activity

You have given a set of NoSQL databases and data models. Drop the databases in left hand side to its relevant data model in right hand side.

Key-value

Document

Column-family

Graph

HBase

Neo4j

HyperTable

Redis

CouchDB

MongoDB

Riak

FlockDB

Cassandra

Infinite Graph

Activity

Sandun is a software Developer who is planning to implement a system on network and IT operations. He has to decide on what is the most suitable NoSQL database model for his system. The database model he selects should be able to represent the connection of IT managers, catalog assets and their deployments. With the application, network managers can do analyses, answering the following questions:

- Which applications or services do particular customers rely upon? (Top-down analysis)
- In case of a failure in a network element, which applications and services, and customers will be affected? (Bottom-up analysis)

What is the most suitable NoSQL database model for the above use case?

Activity

Rehana is an architect working on a software project on Event logging. It has the following features.

- Simple setup and maintenance
- Very high velocity of random read & writes
- Less secondary index needs
- Wide column requirements.

What is the most suitable NoSQL database model for the above software product?

Activity

Mohammad wants to build an e-commerce website to have a shopping cart. He wants the shopping carts to have the capability to be accessible across browsers, and sessions. His friend, Karthigai suggested to add all the shopping information and user-id into two entities which will generate a connection.

What is the most suitable NoSQL database model for the above software product?

Activity

Drag and drop the correct answer to the blanks from the given list.

(Consistency, Availability, Partition Tolerance, master-slave, Replication, replica sets, column, memtable)

The CAP theorem states that we can ensure only two features of_______, _________, and _________________.__In Document databases, availability is improved by data replication using the ______ setup. With __________, we can access data stored in multiple nodes and the clients do not have to worry about the failure of the primary node since data is available in other nodes as well. In MongoDB, availability is achieved using ___________. Cassandra is a column-family database which uses _____ as the basic unit of storage. The procedure of receiving a write by Cassandra is as follows. Data is stored into memory only after stored in a commit log. The term used to describe that in-memory structure is ________.

Activity

Drag and drop the correct answer to the blanks from the given list.

(inter-aggregate, intra-aggregate, node and edge, schemaless, materialised views, map-reduce)

It is difficult to handle _________ relationships than ___________ relationships.

____________ are more suitable for the application which have connected relationships.

One of the main advantages we have in ___________databases is the ability of adding fields to records freely.

In graph databases, __________ are provided as directed connections of two node entities.

Activity

Drag and drop the correct answer to the blanks from the given list.

(aggregate , unit, ACID, aggregate-oriented, clusters)

A data collection that we consider as a unit is known as an _________.

___________ properties guarantee that once a transaction is complete, its data is consistent and stable on disk.

Key-value, document, and column-family are different forms ___________database.

Aggregates make the database easier to handle in _________.

Object databases and Relational databases

- **Handling Relationships**
  - In ODBs, features of a relationship which is also called reference attributes are used to manage relationships between the objects. In relational DBs, attributes with matching values are used to specify the relationship among the tuples(records). Foreign keys are used for referencing relation in relational DBs.
  - Single reference or collection of references can be used in ODBs, but the basic relational model only support single valued references. Hence, representation of many to many relationships are not straight forward in relational model, a separate relation should be created to represent M:N relationships.
  - However, mapping of binary relationships in ODBs is not a direct process. Therefore, designer should specify which side should possess the attributes.

Object databases and Relational databases

- **Handling Inheritance**
  - In ODB a construct is already available in the inner-workings for handling inheritance. But basic relational model does not have such options to handle inheritance.
- **Specifying Operations**
  - In ODB, operations should be designed during the design, as a part of class specification. Relational model does not require the designed to specify the operations during the design phase.
  - Relational model supports ad-hoc queries while ad-hoc queries will violate the encapsulation in ODBs.

XML and Relational databases

- **Relationships**
  - XML databases follow a hierarchical tree structure with simple and composite elements and attributes representing relationships. But relational databases have relationships among tables where one table is the parent table, and the other table is the dependent table.
- **Self - Describing Data**
  - In XML databases, the tags define both the meaning and explanation of the data items together with the data values. Hence different data types are possible within a single document. In relational databases, data in a single column should be of the same type and column definition defines the data definition.

XML and Relational databases

Inherent Ordering

- In XML databases, the order of the data items is defined by the ordering of the tags. The document does not include any other type of ordering. But in relational databases, unless an order by clause is given, the ordering of the data rows is according to the row ordering inside tables.

NoSQL and Relational databases

Data modelling difference

- The definition of a data model stands as the way we can interact with the database and the stored data, which is not the sae as a storage model.
- Storage model's definition says it's how the storing and manipulation of data happens in the database.
- In contrast, aggregate orientation realises the need to use data which have a complex structure than a using set of records.

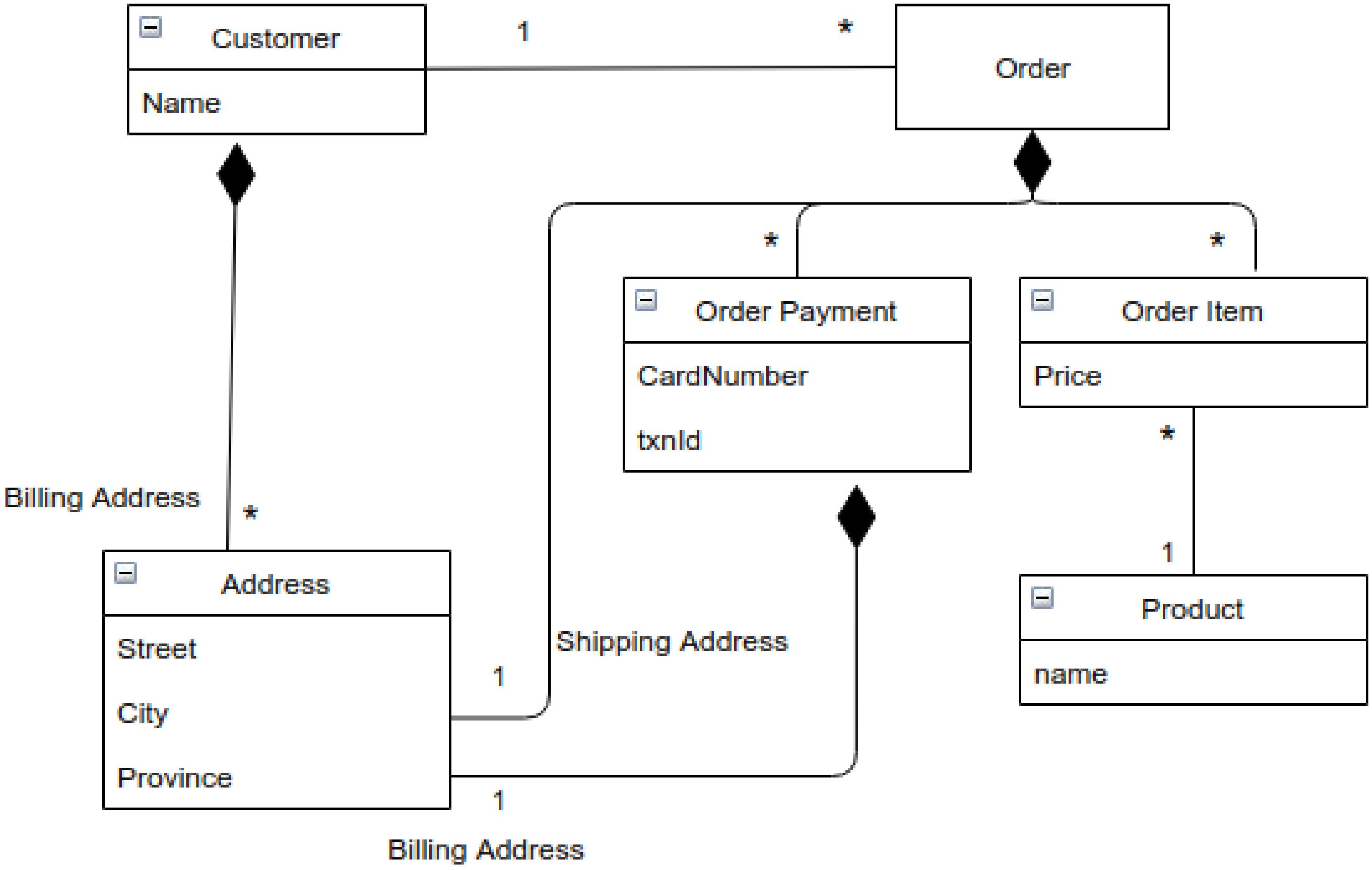
NoSQL and Relational databases

Data modelling difference

- All NoSQL data models make use of a complex record that lets us nest lists and other structures inside it.
- Domain-Driven Design gave birth to the term aggregate.
- Therefore, an aggregate is defined in domain driven design as a collection of related objects that are treated as a unit.

NoSQL and Relational databases

Data modelling difference



© e-Learning Centre, UCSC

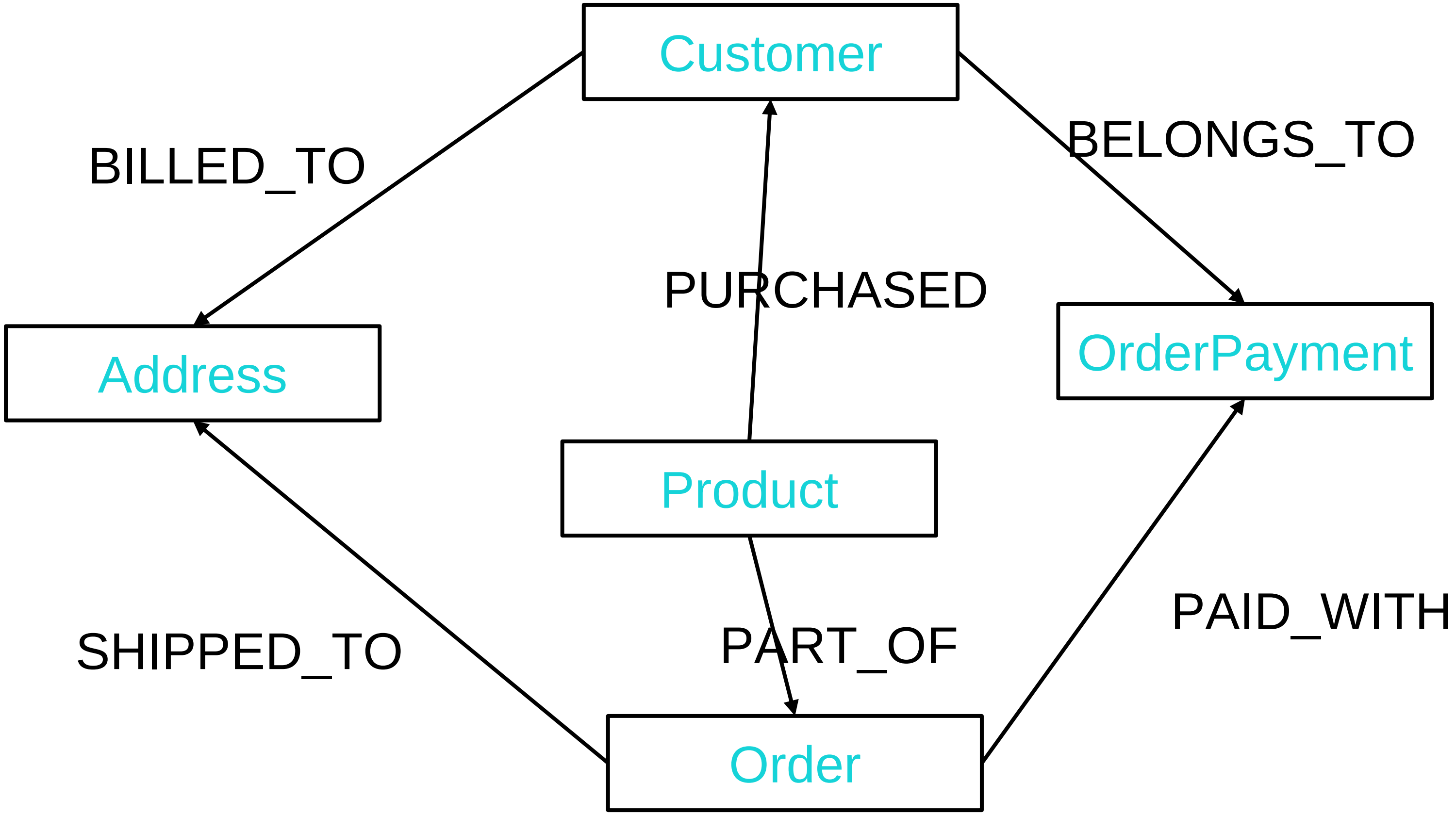
NoSQL and Relational databases

Modelling for Data Access

- Aggregates can be used for analysis purposes. The denormalisation of the data lets us get to the interested data item quickly.
- Having the information of the columns in an ordered manner lets us make it easy to find the mostly-accessed columns fast through naming conventions.
- However, when it comes to modelling data with column families, we need to do it on a basis of queries rather than on the writing purpose.
- The general rule to follow is to make the querying task easy and denormalise the data at the point of write.

NoSQL and Relational databases

Modelling for Data Access



NoSQL and Relational databases

Modelling for Data Access

- In the above figure, what we need to do to find all the Customers who PURCHASED a product with the name Refactoring Databases, is pass a query to the product node with Refactoring Databases and check all the Customers with the incoming PURCHASED relationship.
- As it's clear, using graph databases for this type of relationship traversal is convenient.
- It is also convenient at the point when you need to utilise the information to generate recommendations or to discover patterns.
- In modelling data with graph databases, all objects are modelled as nodes and relations go as relationships.
- Relationships have two features: types and directions.

NoSQL and Relational databases

Aggregate-oriented vs aggregate ignorant

- The complexity in Inter-aggregate relationships is higher than intra-aggregate relationships.
- While schemaless databases allows to add fields to records with no restriction, there is an implicit schema which the users, who utilise the data, need.
- Materialised views are processed by Aggregate-oriented databases. This gives a different organisation to data from primary aggregates. The mechanism used in this case is map-reduce computations.

NoSQL and Relational databases

Schemalessness in NoSQL

- NoSQL databases have no schema; useful when we have to work with nonuniform data.
- A key is used to store data in a key-value store.
- Since a document database achieves the same and therefore, we have no restrictions on the document structure.
- We can store data in columns in Column-family databases.

NoSQL and Relational databases

Overview of Materialised Views

- In Relational databases, the absence of aggregate feature help the users access data in different ways.
- Therefore, users can see the data in different views.
- Even though a view is similar to a relational table, it is not created physically in the database. It is defined on the base tables.
- Views can be generated with derived data as well as with the data available in base tables. But the source of data is not exposed to the client.
- Materialised views are data which are computed early and stored in the cache.
- It is beneficial for an application which has more read operations.

NoSQL and Relational databases

Overview of Materialised Views cont.

- There are no views in NoSQL databases. But there are precomputed, cached queries which can be reused. Those are called “materialised view”.
- In an aggregate-oriented databases, there can be queries that don’t match exactly with the aggregate structure.
- We can use materialised views inside the same aggregate.
- For an example, suppose we have an order document with a summary of the order included in it. When we are querying on order summary data, there is no requirement to transfer the order document in full.
- When we consider column-family databases, we can use several column families for materialised views.
- The benefit of this approach is the ability to update the materialised view within a single atomic operation.

Activity

State whether the following statements are true or false.

When an instant query for searching through the database needs to be executed, the most successful output will be given by relational databases compared to Object databases.	
In relational databases all the data items need to be of the same data type while in XML databases, different data types are allowed.	
Foreign key concept is available in both relational and XML databases.	
If the functionalities of a certain entity in a database is not known at the time of creating the entities, it is better to use the relational model.	
Both Object databases and relational databases support object oriented concepts at the creation of the database.	

Activity

Select the correct option from True/False column.

Statement	True/False
A view is not like a relational table	
Since NoSQL databases don’t have views, they cannot have precomputed and cached queries	
Graph databases organise data into node and edge graphs	
Storage model describes how the database stores and manipulates the data internally	
An aggregate is a unit for data manipulation and management of consistency.	

Summary

Major concepts of object-oriented, XML, and NoSQL databases

Object Databases	Overview of Object Database concepts
XML Databases	Reasons for the origin of XML Structured, Semi structured, Unstructured data XML hierarchical data model
NoSQL Databases	Origins of NoSQL Data models in NoSQL

Summary

Contrast and compare relational databases concepts and non-relational databases

Object DB and Relational DB

XML and Relational DB

NoSQL and Relational DB

Data modelling difference, Aggregate oriented vs aggregate ignorant, Schemalessness in NoSQL, Overview of Materialised views