UNIT 13 OBJECT-ORIENTED DATABASE

Structure **Page No.**

13.0 Introduction
13.1 Objectives
13.2 Why Object-Oriented Database?
    13.2.1 Limitations of Relational Databases
    13.2.2 The Need for Object-Oriented Databases
13.3 Object-Relational Database Systems
    13.3.1 Complex Data Types
    13.3.2 Types and Inheritances in SQL
    13.3.3 Additional Data Types of OOP in SQL
    13.3.4 Object Identity and Reference Type Using SQL
13.4 Object-Oriented Database Systems
    13.4.1 Object Model
    13.4.2 Object Definition Language
    13.4.3 Object Query Language
13.5 OODBMS Vs Object-Relational Database
13.6 Summary
13.7 Solutions/Answers

13.0 INTRODUCTION

Object-oriented software development methodologies have become very popular in the development of software systems. Database applications are the backbone of most of these commercial business software developments. Therefore, it is but natural that object technologies also have their impact on database applications. Database models are being enhanced in computer systems for developing complex applications. For example, a true hierarchical data representation like a generalisation hierarchy scheme in a rational database would require a number of tables but could be a very natural representation for an object-oriented system. Thus, object-oriented technologies have found their way into database technologies. The present-day commercial RDBMS supports the features of object orientation.

This unit introduces various features of object-oriented databases. In this unit, we shall discuss the need for object-oriented databases, the complex types used in object-oriented databases, how these may be supported by inheritance, etc. In addition, we also define object definition language (ODL) and object manipulation language (OML). We shall introduce the object-oriented and object-relational databases.

13.1 OBJECTIVES

After going through this unit, you should be able to:

- define the need for object-oriented databases;
- explain the concepts of complex data types;
- use SQL to define object-oriented concepts;
- familiarise yourself with object definition and query languages, and
- define object-relational and object-oriented databases.

13.2 WHY OBJECT-ORIENTED DATABASE?

An object-oriented database is used for complex data types. Such database applications require complex interrelationships among object hierarchies to be represented in database systems. These interrelationships are difficult to implement in relational systems. Let us discuss the need for object-oriented systems in advanced applications in more detail. However, first, let us discuss the weakness of relational database systems.

13.2.1 Limitations of Relational Databases

Relational database technology was not able to handle complex application systems such as Computer-Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Integrated Manufacturing (CIM), Computer Aided Software Engineering (CASE) etc. The limitation for relational databases is that they have been designed to represent entities and relationships in the form of two-dimensional tables. Any complex interrelationship, like multi-valued attributes or composite attributes, may result in the decomposition of a table into several tables; similarly, complex interrelationships result in a number of tables being created. Thus, the main asset of relational databases, viz., its simplicity for such applications, is also one of its weaknesses in the case of complex applications.

The data domains in a relational system can be represented in relational databases as standard data types defined in the SQL. However, the relational model does not allow extending these data types or creating the user's own data types. Thus, limiting the types of data that may be represented using relational databases.

Another major weakness of the RDMS is that concepts like inheritance/hierarchy need to be represented with a series of tables with the required referential constraint. Thus, they are not very natural for objects requiring inheritance or hierarchy.

However, one must remember that relational databases have proved to be commercially successful for text-based applications and have lots of standard features, including security, reliability and easy access. Thus, even though they may not be a very natural choice for certain applications, yet their advantages are far too many.
Thus, many commercial DBMS products are basically relational but also support object-oriented concepts.

13.2.2 The Need for Object-Oriented Databases

As discussed in the earlier section, relational database management systems have certain limitations. But how can we overcome such limitations? Let us discuss some of the basic issues with respect to object-oriented databases.

The objects may be complex, or they may consist of low-level objects (for example, a window object may consist of many simpler objects like a menu bar, scroll bar, etc.). However, to represent the data of these complex objects through relational database models you would require many tables – at least one each for each inherited class and a table for the base class. To ensure that these tables operate correctly, you would need to set up referential integrity constraints as well. On the other hand, object-oriented models would represent such a system very naturally through, an inheritance hierarchy. Thus, it is a very natural choice for such complex objects.

Consider a situation where you want to design a class (say an address class, which includes address lines, City, State and Pin code); the advantage of object-oriented

database management for such situations would be that they allow the representation of not only the structure but also the operation on newer user-defined database type such as finding the Address with similar Pin code. Thus, object-oriented database technologies are ideal for implementing such systems that support complex inherited objects, user defined data types (that define operations including the operations to support inheritance and polymorphism).

Another major reason for the need of object-oriented database system would be the seamless integration of this database technology with object-oriented applications. Software design is now mostly based on object-oriented technologies. Thus, object-oriented databases may provide a seamless interface for combining the two technologies.

Object-oriented databases are also required to manage complex, highly interrelated information. They provide solutions in the most natural and easy way that is closer to our understanding of the system. **Michael Brodie** related object-oriented systems to the human conceptualisation of a problem domain, which enhances communication among the system designers, domain experts, and the system end users.

The concept of an object-oriented database was introduced in the late 1970s, however, it became significant only in the early 1980s. The initial commercial product offerings appeared in the late 1980s. Some of the popular object-oriented database management systems were Objectivity/DB (developed by Objectivity, Inc.), VERSANT (developed by Versant Object Technology Corp.), Cache, ZODB, etc. An object-oriented database can be used in application areas such as e-commerce, engineering product data management, securities, medicine, etc.

Figure 1 traces the evolution of object-oriented databases. *Figure 2* highlights the strengths of object-oriented programming and relational database technologies. An object-oriented database system needs to capture the features from both these worlds. Some of the major concerns of object-oriented database technologies include access optimisation, integrity enforcement, archive, backup and recovery operations etc.

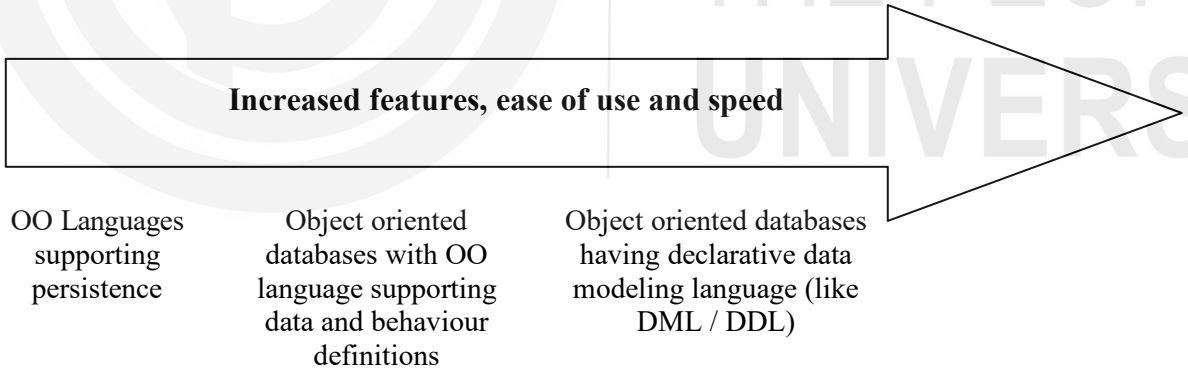

Figure 1: The evolution of object-oriented systems

Now, the question is, how does one implement an Object-oriented database system? As shown in *Figure 2* an object-oriented database system needs to include the features of object-oriented programming and relational database systems. Thus, the two most natural ways of implementing them will be either to extend the concept of object-oriented programming to include database features (OODBMS) or extend the relational database technology to include object-oriented features (Object Relational Database Systems). Let us discuss these two viz., the object relational and object-oriented database systems in more detail in the subsequent sections.

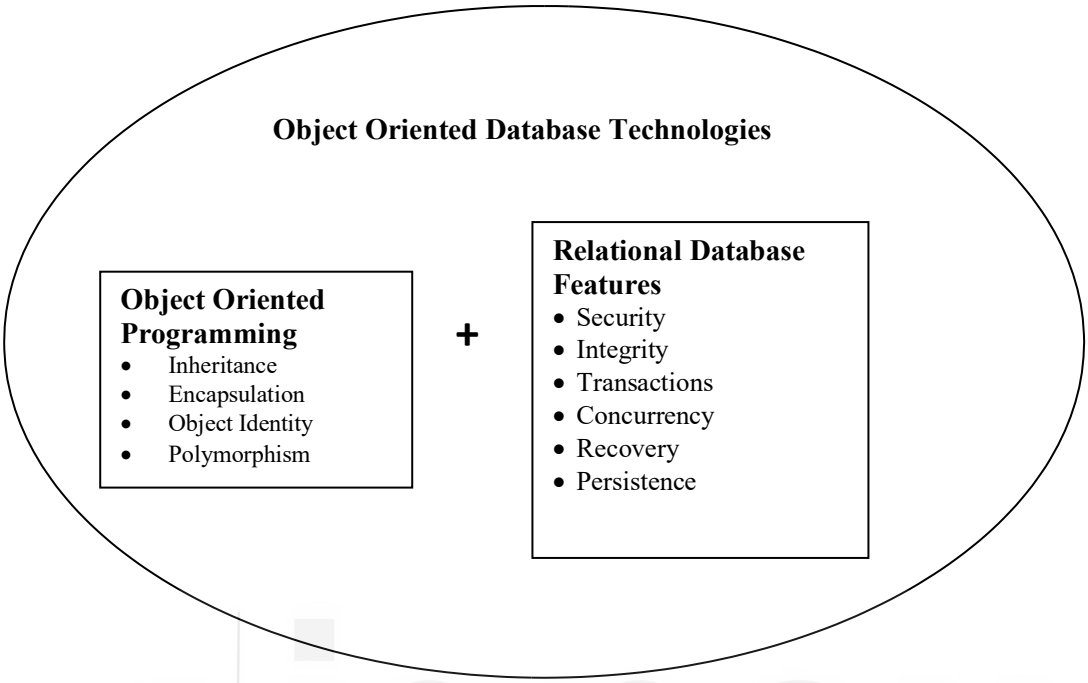

Figure 2: Makeup of an Object-Oriented Database System

13.3 OBJECT-RELATIONAL DATABASE SYSTEMS

Object-Relational Database Systems are the relational database systems that have been enhanced to include the features of object-oriented paradigm. This section provides details on how these newer features have been implemented in the SQL. Some of the basic object-oriented concepts that have been discussed in this section in the context of their inclusion into SQL standards include, the complex types, inheritance and object identity and reference types.

13.3.1 Complex Data Types

In the previous section, we have used the term complex data types without defining it. Let us explain this with the help of a simple example. Consider a composite attribute *Address*. The Address of a person in an RDBMS can be represented using the following:

House-no and apartment
Locality
City
State
Pin-code

When using RDBMS, such information either needs to be represented as separate attributes, as shown above, or just one string separated by comma or semicolon. The second approach is very inflexible, as it would require complex string related operations for extracting information. It also hides the details of an address; thus, it is not suitable.

If you represent the attributes of the *Address* as separate attributes, then the problem would be with respect to writing queries. For example, if you need to find the address of a person, you need to specify all the attributes that you have created for the Address

viz., House-no, Locality…. etc. The question is - Is there any better way of representing such information using a single field? If there is such a mode of representation, then that representation should permit distinguishing each element of the *Address*. The following may be one such possible solution:

```
CREATE TYPE Address AS     (
      Addressline1         Char(20)
      Addressline2         Char(20)
      City                 Char(12)
      State                Char(15)
      Pincode              Char(6)
      ) ;
```

Thus, *Address* is now a new type that can be used while creating a database system schema as:

```
CREATE TABLE STUDENT       (
      name                 Char(25)
      address              Address
      phone                Char(12)
      programme            Char(5)
      dob                  Date
      ) ;
```

But what are the advantages of such definition?

Consider the following queries:

Find the name and address of the students enrolled in the PGDCA programme.

```
SELECT     name, address
FROM       student
WHERE      programme = 'PGDCA' ;
```

Please note that the attribute 'address', although composite, is used as a single attribute in the query. But can you also refer to individual components of this attribute?
Find the name and address of all the PGDCA students of Mumbai.

```
SELECT  name, address
FROM    student
WHERE   programme = 'MCA' AND address.city='Mumbai';
```

Thus, such definitions allow you to handle a composite attribute as a single attribute with a user-defined type. You can also refer to any of the components of this attribute without any problems, so the data definition of the components of the composite attribute is still intact.

Complex data types also allow you to model a table with multi-valued attributes, which would require a new table in a relational database design. For example, a library database system would require representing the following information of a book.

- ISBN number
- Book title
- Authors
- Published by
- Subject areas of the book.

Clearly, in the table above, authors and subject areas are multi-valued attributes. You

need to design two relational database tables for these attributes – Author (ISBNnumber, author) and Area (ISBNnumber, subject area). (Please note that this design does not consider the author's position in the list of authors).

Although this database solves the immediate problem, yet it is a complex design. An object-oriented database system may solve this problem. This is explained in the next section.

13.3.2 Types and Inheritances in SQL

In the previous sub-section, we discussed the data type – *Address.* It is a good example of a structured type. In this section, let us give more examples for such types, using SQL. Consider the attribute:

- Name – that includes given name, middle name and surname.
- Address – that includes address details, city, state and pin code.

These types can be defined using SQL extensions, as given below:

```
CREATE TYPE  Name AS (
       Given_name     Char (20),
       Middle_name    Char (15),
       Sur_name       Char (20)
)
FINAL
```

This type/class cannot be inherited further due to the keyword FINAL.

```
CREATE TYPE      Address AS (
       Add_det   Char(20),
       City      Char(20),
       State     Char(20),
       Pincode   Char(6)
)
NOT FINAL
```

You can use this class to create inherited classes, like *Home_Address* and *Office_Address*, as this type/class is NOT FINAL.

The FINAL and NOT FINAL keywords have the same meaning as you learned in JAVA, i.e., a FINAL class cannot be inherited further.

These types can now be used to create a student class, which has data members and methods that work on objects of the student class, as follows:

```
CREATE  TYPE      Student AS (
      name        Name,
      address     Address,
      dob         Date
)
NOT FINAL
METHOD ageinyears (givendate Date)
      RETURN INTERVAL YEAR;
```

The method can be implemented separately using the following SQL Commands:

```
CREATE INSTANCE METHOD (givendate Date)
            RETURN INTERVAL YEAR
      FOR Student
      begin
            Return (givendate – self.dob);
      end
```

This method computes the age on a given date. Please note that Date is a data type of SQL. `FOR` is the looping construct and will result in the execution of this method for every student object's instance.

The possibility of using constructors also exists, but a detailed discussion on that is beyond the scope of this unit.

Type Inheritance

In the present standard of SQL, you can define inheritance. Let us explain this with the help of an example.

Consider a type *University_person* defined as:

```
CREATE TYPE      University_person AS (
      name       Name,
      address    Address
)
```

Now, this type can be inherited by the *Staff* type or the *Student* type. For example, the Student type, if inherited from the class given above, would be:

```
CREATE TYPE      Student
      UNDER      University_person (
      programme  Char(10),
      dob        Date
)
```

Similarly, you can create a sub-class for the *Staff* of the University as:

```
CREATE TYPE           Staff
      UNDER           University_person (
      designation     Char(10),
      basic_salary    Number(7)
)
```

Notice that both the inherited types shown above inherit the *name* and *address* attributes from the type `University_person`. Methods can also be inherited in a similar way; however, they can be overridden if the need arises.

Table Inheritance

The concept of table inheritance has evolved to incorporate implementation of generalisation/ specialisation hierarchy of an E-R diagram. SQL allows inheritance of tables. Once a new type is declared, it could be used in the process of creation of new tables with the usage of keyword "`OF`". Let us explain this with the help of an example. Consider the classes `University_person`, `Staff` and `Student`, as we have defined in the previous sub-section. You can create the table for the type `University_person` as:

```
CREATE TABLE university_members OF University_person;
```

The table inheritance would allow us to create sub-tables for such tables as:

```
CREATE TABLE student_list OF Student
          UNDER university_members;
```

Similarly, you can create a table for the staff as:

```
CREATE TABLE staff OF Staff
          UNDER university_members;
```

Please note the following points for table inheritance:

- The type that is associated with the sub-table must be the sub-type of the type of the parent table. This is a major requirement for table inheritance.
- All the attributes of the parent table – (`university_members` in our case) should be present in the inherited tables.
- Also, the three tables may be handled separately. However, any record present in the inherited tables is also implicitly present in the base table. For example, any record inserted in the `student_list` table will be implicitly present in `university_members` tables.
- A query on the parent table (such as `university_members`) would find the records from the parent table and all the inherited tables (in our case all three tables). However, the attributes of the result table would be the same as the attributes of the parent table.
- You can restrict your query to only the parent table by using the keyword `ONLY`. For example,
  `SELECT name FROM university_members ONLY;`

13.3.3 Additional Data Types of OOP in SQL

The object-oriented/relational database must support the data types that allow multi-valued attributes to be represented easily. Two such data types that exist in SQL are:

- Arrays – stores information in an order and
- Multisets – stores information in an unordered set.

Let us explain this with the help of an example of a book database as introduced in the section13.3.1 A *Book* type can be represented using SQL as:

```
CREATE TYPE          Book AS  (
     ISBNNO          Char (14),
     BOOK_TITLE      Char (25),
     AUTHORS         Char (25) ARRAY [5],
     PUBLISHED_BY    Char (20),
     KEYWORDS        Char (10) MULTISET
);
```

Please note, the use of the type `ARRAY`. Arrays not only allow authors to be represented but also allow the sequencing of the authors' names. Multiset allows a number of keywords without any ordering imposed on them.

But how can you enter data and query such data types? The following SQL commands would help in defining such a situation. But first, you need to create a table:

```
CREATE TABLE library OF Book;
INSERT INTO library VALUES('008-124476-x', 'Database
     Systems', ARRAY ['Silberschatz', 'Elmasri'],
     'XYZ Publication', MULTISET [ 'Database',
```

```
        'Relational', 'Object Oriented']) ;
```

The command above would insert information on a hypothetical book into the database. Let us now write a few queries on this database:

Find the list of books related to area Object Oriented:

```
SELECT   ISBNNO, BOOK_TITLE
FROM     library
WHERE    'Object Oriented' IN (UNNEST (KEYWORDS));
```

Find the first author of each book:

```
SELECT ISBNNO, BOOK_TITLE, AUTHORS[1]
FROM library ;
```

You can create many such queries. A detailed discussion on this can be found in the latest SQL standards and is beyond the scope of this unit.

13.3.4 Object Identity and Reference Type Using SQL

Till now, we have created the tables, but what about the situation when we have attributes that draw a reference to another attribute in the same table? This is a referential constraint. Thus, an object-relational database system should address the following two concerns:

- In relational databases, foreign keys are used to link the attributes in two different relations. Can such keys be used in an object-relational database?
- How will you identify the object that is being referenced?

The following example will address the concerns stated above.

Example: A library purchases books. Each book is given a unique book number called the catalogue number. The library maintains a `procurement` table, which can be created using the following SQL command:

```
CREATE TABLE procurement (
     CATALOGUE_NO    CHAR (5),
     ISBNNO REF (Book) SCOPE (library)
) ;
```

The SQL statement given above would create a `procurement` table, which would assign two basic information to a newly purchased book – first, it will give the book a unique `CATALOGUE_NO`, and second, it will link this book to a specific record in the `library` table through an `ISBNNO`.

To insert a new book in this table, you must first create a Book object using the `ISBNNO` of this book. Assuming that such an object already exists, then you may use the following command to add a book to the `procurement` table:

```
INSERT INTO procurement
           VALUES ('98765', NULL) ;
```

This command will add a new book to the procurement table having CATALOGUE_NO as '98765' and no reference to the ISBN number. The link to the ISBN number record will be created using the following SQL command:

```
UPDATE procurement
```

```
        SET ISBNNO = (SELECT book_id
        FROM library
        WHERE ISBNNO = '83-7758-476-6')
WHERE CATALOGUE_NO = '98765';
```

Please note that in the query above, the sub-query generates the object identifier (book_id) for the ISBNNO of the book whose accession number is 98765. It then sets the reference for the desired record in the `procurement`.

This is a slightly complex procedure, and several other mechanisms exist to perform this operation. You can refer to them in the further readings.

☞ Check Your Progress – 1

1) What is the need for object-oriented databases?

.....

.....

.....

2) How will you represent a complex data type?

.....

.....

.....

3) Represent an address using SQL that has a method for locating pin-code information.

.....

.....

.....

4) Create a table using the type created in question 3 above.

.....

.....

.....

5) How can you establish a relationship with multiple tables?

.....

.....

.....

13.4 OBJECT-ORIENTED DATABASE SYSTEMS

Object-oriented database systems are applying object-oriented concepts into database system models to create an object-oriented database model. This section describes the concepts of the object model, followed by a discussion on object definition and object manipulation languages that are derived from SQL.

13.4.1 Object Model

Object Data Management Group (ODMG) has designed the object model for the object-oriented database management system. The Object Definition Language (ODL) and Object Manipulation Language (OML) are based on this object model. Let us briefly define the concepts and terminology related to the object model.

Objects and Literal: These are the basic building elements of the object model. An object has the following four characteristics:

- A unique identifier
- A name
- A lifetime, defining whether it is persistent or not, and
- A structure that may be created using a type constructor. The structure in OODBMS can be classified as atomic or collection objects (like Set, List, Array, etc.).

A literal does not have an identifier but has a value that may be constant. The structure of a literal does not change. Literals can be atomic, such that they correspond to basic data types like int, short, long, float, etc. or structured literals (for example, current date, time, etc.) or collection literal defining values for some collection object.

Interface: Interfaces define the operations that can be inherited by a user-defined object. Interfaces are non-instantiable. All objects inherit basic operations (like copy object, delete object) from the interface of Objects. A collection object inherits operations – such as an operation to determine an empty collection – from the basic collection interface.

Atomic Objects: An atomic object is an object that is not of a collection type. They are user-defined objects that are specified using class keywords. The properties of an atomic object can be defined by its attributes and relationships. An example is the book object given in the next subsection. **Please note** here that a *class* is instantiable.

Inheritance: The interfaces specify the abstract operations that can be inherited by classes. This is called behavioural inheritance and is represented using the " : " symbol. Sub-classes can inherit the state and behaviour of super-class(s) using the keyword EXTENDS.

Extents: An extent of an object contains all the persistent objects of that class. A class having an extent can have a key.

In the following section, we shall discuss the use of the ODL and OML to implement object models.

13.4.2 Object Definition Language

Object Definition Language (ODL) is a standard language on the same lines as the DDL of SQL, that is used to represent the structure of an object-oriented database. It uses unique object identity (OID) for each object such as library item, student, account, fees, inventory etc. In this language, objects are treated as records. Any class in the design process has three properties that are attribute, relationship, and methods. A class in ODL is described using the following syntax:

```
class <name>
{
<list of properties>
};
```

Here, *class* is a keyword, and the properties may be attributes, methods or relationships. The attributes defined in ODL specify the features of an object. It could be simple, enumerated, structured or complex type.

```
class Book
{
      attribute string ISBNNO;
      attribute string BOOKTITLE;
      attribute enum CATEGORY
            {text,reference,journal} BOOKTYPE;
      attribute struct AUTHORS
            {string fauthor, string sauthor,
             string tauthor} AUTHORLIST;
};
```

Please note that, in this case, we have defined authors as a structure and a new field on Book type as an enum.

These books need to be issued to the students. For that, you need to specify a relationship. The relationship defined in ODL specifies the method of connecting one object to another. You can specify the relationship by using the keyword "relationship". For example, to connect a student object with a book object, you need to specify the relationship in the student class as:

```
relationship set <Book> receives
```

Here, for each object of the class student, there is a reference to the book object and the set of references is called *receives*.

But if we want to access the student based on the book, then the "inverse relationship" could be specified as

```
relationship set <Student> receivedby
```

You can specify the connection between the relationship *receives* and *receivedby* by using the keyword "inverse" in each declaration. If the relationship is in a different class, it is referred to by the relationship name followed by a scope resolution operator (::) and the name of the other relationship.

The relationship could be specified as:

```
class Book
{
      attribute string ISBNNO;
      attribute string BOOKTITLE;
      attribute integer PRICE;
      attribute string PUBLISHEDBY;
      attribute enum CATEGORY
            {text,reference} BOOKTYPE;
      attribute struct AUTHORS
                 {string fauthor
                  string sauthor
                  string tauthor} AUTHORLIST;
      relationship set <Student> receivedby
            inverse Student::receives;
      relationship set <Supplier> suppliedby
            inverse Supplier::supplies;
};
class Student
{
      attribute string ENROLMENT_NO;
```

```
        attribute string NAME;
        attribute integer MARKS;
        attribute string COURSE;
        relationship set <Book> receives
              inverse Book::receivedby;
};
class Supplier
{
        attribute string SUPPLIER_ID;
        attribute string SUPPLIER_NAME;
        attribute string SUPPLIER_ADDRESS;
        attribute string SUPPLIER_CITY;
        relationship set <Book> supplies
              inverse Book::suppliedby;
};
```

Methods could be specified with the classes along with input/output types. These declarations are called "signatures". These method parameters could be *in*, *out* or *inout*. Here, the '*in*' parameter is passed by value, whereas the '*out*' or '*inout*' parameters are passed by reference. Exceptions could also be associated with these methods.

```
class Student
{
        attribute string ENROLMENT_NO;
        attribute string NAME;
        attribute string st_address;
        relationship set <Book> receives
              inverse Book::receivedby;
        void findcity(in set<string>,out set<string>)
              raises(notfoundcity);
};
```

In the method findcity, the name of the city is passed with the objective to find the name of the students who belong to that specific city. In case blank is passed as a parameter for city name, then the exception notfoundcity is raised. The implementation of this method can be done separately.

The ODL could be atomic type or class names. The basic type uses many class constructors such as set, bag, list, array, dictionary and structure. We have shown the use of some in the example above. You can refer to the further readings for more detail on these.

Inheritance is implemented in ODL using subclasses with the keyword "extends".

```
class Journal extends Book
{
        attribute string VOLUME;
        attribute string emailauthor1;
        attribute string emailauthor2;
};
```

Multiple inheritance is implemented by using extends separated by a colon (:). If there is a class Fee containing fee details, then multiple inheritance could be shown as:

```
class StudentFeeDetail extends Student:Fee
{
        void deposit(in set <float>, out set <float>)
```

```
                    raises(refundToBeDone)
};
```

Like the difference between relation schema and relation instance, ODL uses the class and its extent (set of existing objects). The objects are declared with the keyword "extent".

```
class Student (extent firstStudent)
{
      attribute string ENROLMENT_NO;
      attribute string NAME;
      ..........
};
```

It is not necessary in the case of ODL to define keys for a class. But if one or more attributes have to be declared as keys, then it may be done with the declaration of a key for a class with the keyword "key".

```
class student (extent firstStudent key ENROLMENT_NO)
{
      attribute string ENROLMENT_NO;
      attribute string NAME;
      ..........
};
```

Assuming that the ENROLMENT_NO and ACCESSION_NO form a key for the issue table, then:

```
class Issue(extent thisMonthIssue key
                    (ENROLMENT_NO,ACCESSION_NO))
{     attribute string ENROLMENT_NO;
      attribute string ACCESSION_NO;
      ..........
};
```

The major considerations while converting ODL designs into relational designs are as follows:

a) It is not essential to declare keys for a class in ODL, but in Relational design new attributes have to be created as a key.

b) Attributes in ODL could be declared as non-atomic, whereas in Relational design they have to be converted into atomic attributes.

c) Methods could be part of the design in ODL, but they cannot be directly converted into a relational schema, as they are not the property of a relational schema.

d) Relationships are defined in inverse pairs for ODL, but in the case of relational design only one pair is defined.

For example, for the book class schema, the relation is:

```
Book(ISBNNO,TITLE,CATEGORY,fauthor,sauthor,tauthor)
```

ODL has been created with the features required to create an object-oriented database in OODBMS. You can refer to the further readings for more details on it.

13.4.3 Object Query Language

Object Query Language (OQL) is a standard query language that takes high-level declarative programming of SQL and object-oriented features of OOPs. Let us explain it with the help of examples.

Find the list of authors for the book titled "OODBMS".

```
SELECT b.AUTHORS
FROM Book b
WHERE BOOK_TITLE="OODBMS" ;
```

Display the title of the book which has been issued to the student whose name is Anand.

```
SELECT BOOK_TITLE
FROM Book b, Student s
WHERE s.NAME ="Anand" ;
```

This query can also be written by using a relationship as:

```
SELECT BOOK_TITLE
FROM Book b
WHERE b.receivedby.NAME ="Anand" ;
```

In the previous case, the query creates a bag of strings, but when the keyword DISTINCT is used, the query returns a set.

```
SELECT DISTINCT BOOK_TITLE
FROM Book b
WHERE b.receivedby.NAME ="Anand" ;
```

When you add the ORDER BY clause to return a sorted list.

```
SELECT BOOK_TITLE
FROM Book b
WHERE b.receivedby.NAME ="Anand"
ORDER BY b.CATEGORY ;
```

Aggregate operators like SUM, AVG, COUNT, MAX, MIN could be used in OQL. If you want to compute the maximum marks obtained by any student, then the OQL command is

```
Max(SELECT s.MARKS FROM Student s);
```

Group By and Having clauses can also be used; however, you may refer to further readings for details on them.

Union, intersection and difference operators are applied to set or bag type with the keywords UNION, INTERSECT and EXCEPT. If you want to display the details of suppliers from PATNA and SURAT, then the OQL command is:

```
(SELECT DISTINCT su
      FROM Supplier su
      WHERE su.SUPPLIER_CITY="PATNA")
UNION
(SELECT DISTINCT su
      FROM Supplier su
      WHERE su.SUPPLIER_CITY="SURAT");
```

The result of the OQL expression could be assigned to host language variables. If costlyBooks is a set <Book> variable to store the list of books whose price is more than Rs. 500, then:

```
costlyBooks = SELECT DISTINCT b
                  FROM Book b
                  WHERE b.PRICE > 500
```

In this section, you have been introduced to OQL. You can refer to further readings for more details on OQL.

☞ Check Your Progress – 2

1) Create a class staff using ODL that also references the Book class given in section 13.4.

.....

.....

2) What modifications would be needed in the Book class because of the table created by the above query?

.....

.....

3) Find the list of books that have been issued to "Shashi".

.....

.....

.....

13.5 OODBMS VERSUS OBJECT RELATIONAL DATABASE

An object-oriented database management system is created on the basis of persistent programming paradigm, whereas an object-relational is built by creating object-oriented extensions of a relational system. In fact, both the products have clearly defined objectives. The following table shows the difference between them:

Object-Relational DBMS	Object-Oriented DBMS
The features of these DBMS include: • Support for complex data types • Powerful query languages support through SQL • Good protection of data against programming errors	The features of these DBMS include: • Support complex data types, • Very high integration of database with the programming language, • Very good performance • But not as powerful at querying as Relational.
One of the major assets here is SQL. Although, SQL is not as powerful as a Programming Language, but it is none-the-less essentially a fourth-generation language, thus, it provides excellent protection of data from the Programming errors.	It is based on object-oriented programming languages, thus, are very strong in programming, however, any error of a data type made by a programmer may affect many users.
The relational model has a very rich foundation for query optimisation, which helps in reducing the time taken to execute a query.	These databases are still evolving in this direction. They have reasonable systems in place.

These databases make the querying as simple as in relational even, for complex data types and multimedia data.	The querying is possible but somewhat difficult to get.
Although the strength of these DBMS is SQL, it is also one of the major weaknesses from the performance point of view of in memory applications.	Some applications that are primarily run in the RAM and require a large number of database accesses with high performance may find such DBMS more suitable. This is because of rich programming interface provided by such DBMS. However, such applications may not support very strong query capabilities. A typical example of one such application is databases required for CAD.

☞ Check Your Progress – 3

State True or False.

1) Object-relational databases cannot represent inheritance but can represent complex data types. T ☐ F ☐

2) The class extent defines the limit of a class. T ☐ F ☐

3) The query language of object-oriented DBMS is stronger than object-relational databases. T ☐ F ☐

4) SQL commands cannot be optimised. T ☐ F ☐

5) Object-oriented DBMS supports a very high level of integration of databases with OOP. T ☐ F ☐

13.6 SUMMARY

Object-oriented technologies are one of the most popular technologies in the present era. Object orientation has also found its way into database technologies. The object-oriented database systems allow the representation of user-defined types, including operation on these types. They also allow the representation of inheritance using the type and table inheritance. The idea here is to represent the whole range of newer types if needed. Such features help in enhancing the performance of a database application that would otherwise have many tables. SQL support these features for object-relational database systems.

The object definition languages and object query languages have been designed for the object-oriented DBMS on the same lines as that of SQL. These languages try to simplify various object-related representations using OODBMS.

The object-relational and object-oriented databases do not compete with each other but have different kinds of application areas. For example, relational and object-relational DBMS are most suited for simple transaction management systems, while OODBMS may find applications with e-commerce, CAD and other similar complex applications.

13.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) The object-oriented databases are need for:

- Representing complex types.
- Representing inheritance, polymorphism
- Representing highly interrelated information
- Providing object-oriented solution to databases bringing them closer to OOP.

2) Primarily by representing it as a single attribute. All its components should also be referenced separately.

3)

```
CREATE TYPE Addrtype AS
      (
            houseNo    CHAR(8),
            street     CHAR(10),
            colony     CHAR(10),
            city       CHAR(8),
            state      CHAR(8),
            pincode    CHAR(6),
      );

METHOD pin() RETURNS CHAR(6);
CREATE METHOD pin() RETURNS CHAR(6);
      FOR Addrtype
            BEGIN
                  Return pincode;
            END
```

4)
```
CREATE TABLE addresswithpin OF Addrtype
(
      REF IS addressid,
      PRIMARY KEY
            (street,colony,city,pincode)
);
```

5) The relationship can be established with multiple tables by specifying the keyword “SCOPE”. For example:

```
CREATE TABLE mylibrary
(
      mybook REF(Book) SCOPE library;
      myStudent REF(Student) SCOPE student;
      mySupplier REF(Supplier) SCOPE supplier;
);
```

Check Your Progress 2

1)

```
class Staff
{
      attribute string STAFF_ID;
```

```
        attribute string STAFF_NAME;
        attribute string DESIGNATION;
        relationship set <Book> issues
              inverse Book::issuedto;
};
```

2) The Book class needs to represent the relationship that is with the Staff class.

This would be added to it by using the following commands:

```
RELATIONSHIP SET < Staff > issuedto
      INVERSE :: issues Staff
```

3)
```
SELECT DISTINCT b.TITLE
FROM BOOK b
WHERE b.issuedto.NAME = "Shashi"
```

Check Your Progress 3

1) False 2) False 3) False 4) False 5) True

UNIT 14 DATA WAREHOUSING AND DATA MINING

Structure

14.0 Introduction
14.1 Objectives
14.2 What Is Data Warehousing?
14.3 Basic Components of a Data Warehouse
    14.3.1 The Data Sources
    14.3.2 Data Extraction, Transformation and Loading (ETL)
14.4 Multidimensional Data Model of a Data Warehouse
14.5 Data Mining Technology
14.6 Classification
    14.6.1 Classification Using Distance (K-Nearest Neighbour)
    14.6.2 Decision Tree
14.7 Clustering
14.8 Association Rule Mining
14.9 Applications of Data Mining
14.10 Summary
14.11 Solutions/Answers
14.12 Further Readings

14.0 INTRODUCTION

With the advancement of communication technology, a large amount of data is generated and collected by various organisations. This large data can be used for making meaningful decisions, which can be attributed to discovering useful knowledge and patterns from their existing data. One of the ways of storing vast reservoirs of data is data warehousing. Data warehouses provide superior capabilities for data storage, processing, and responsiveness to analytical queries compared to transaction-oriented databases. In decision-making applications, users need to access a larger volume of data very rapidly – much faster than what can be conveniently handled by traditional database systems. Often, such data is extracted from multiple operational databases. The data of a data warehouse can be used by a data mining application. Data mining is an interdisciplinary field that takes its approach from different areas like databases, statistics, artificial intelligence and data structures to extract hidden knowledge from large volumes of data. The data mining concept is used by the research community and the industry for various kinds of data analysis.

This unit aims to introduce you to some of the fundamental techniques used in data warehousing and data mining. This unit introduces only those data warehousing concepts which relate to structured data.

14.1 OBJECTIVES

After going through this unit, you should be able to:

- Illustrate the concepts of a data warehouse;
- discuss data warehousing schemas;
- identify the multi-dimensional data modelling of a data warehouse;
- explains the purpose of data mining and its applications;
- illustrate classification and clustering approaches of data mining;
- explains how association rules can be identified in data mining.

14.2 WHAT IS DATA WAREHOUSING?

Let us first try to answer the question: What is a data warehouse? A simple answer could be: A data warehouse is a tool that manages data *after and outside* of operational systems. Thus, it is not a replacement for the operational systems but is a major tool that acquires data from the operational systems. Data warehousing technology has evolved in business applications for the process of strategic decision-making. Data warehouses, sometimes, may be considered the key components of the IT strategy and architecture of an organisation.

The formal definition of the data warehouse was given by **W.H. Inman:** "A Data warehouse is a collection of data, which is (i) *subject-oriented,* (ii) *integrated,* (iii) *nonvolatile* (iv) *time-variant* that supports decision-making by the management". *Figure 1* presents some uses of data warehousing in various industries.

S.No.	Industry	Functional Areas of Use	Strategic Uses
1	Banking	Creating new schemes for loans and other banking products; helps in operations; identifies information for marketing.	Finding trends for customer service; product and service promotions; reduction of expenses.
2	Airline	Operations; marketing.	Crew assignment; aircraft maintenance plans; fare determination; analysis of route profitability; frequent - flyer program design.
3	Hospital	Operation optimisation.	Reduction of operational expenses; scheduling of resources.
4	Investment and Insurance	Insurance product development; marketing	Risk management; financial market analysis; customer tendencies analysis; portfolio management

Figure 1: Uses of Data Warehousing

A data warehouse offers the following advantages:

- It provides historical information that can be used in many different forms of analysis.
- Improves the data quality by predicting missing data.
- It can help support disaster recovery, although not alone, but with other backup resources.

One of the major advantages of a data warehouse is that it allows a large collection of historical data from many operational databases that can be analysed through one data warehouse interface. A data warehouse does not create value of its own in an organisation. However, the value can be generated by the users of the data warehouse. For example, a data warehouse of a manufacturing unit may answer some of the following questions:

- What would be the income, expenses and profit for a year?
- What would be the sales amount this month?
- Who are the vendors for a product that is to be procured?
- How much of each product is manufactured in each production unit?
- How much is to be manufactured?

- What percentage of the product is defective?
- Are customers satisfied with the quality? etc.

The data warehouse supports various business intelligence applications. Some of these may be - online analytical processing (OLAP), decision-support systems (DSS), data mining etc. We shall discuss some of these terms in more detail in the later sections.

14.3 BASIC COMPONENTS OF A DATA WAREHOUSE

A data warehouse is defined as a *subject-oriented, integrated, nonvolatile, time-variant collection,* but how can we achieve such a collection? To answer this question, let us define the basic architecture that helps a data warehouse achieve the objectives stated above. We shall also discuss various processes that are performed by these components on the data.

Figure 2 defines the basic architecture of a data warehouse. The analytical reports are not a part of the data warehouse but are one of the major business application areas including OLAP, DSS and Data Mining

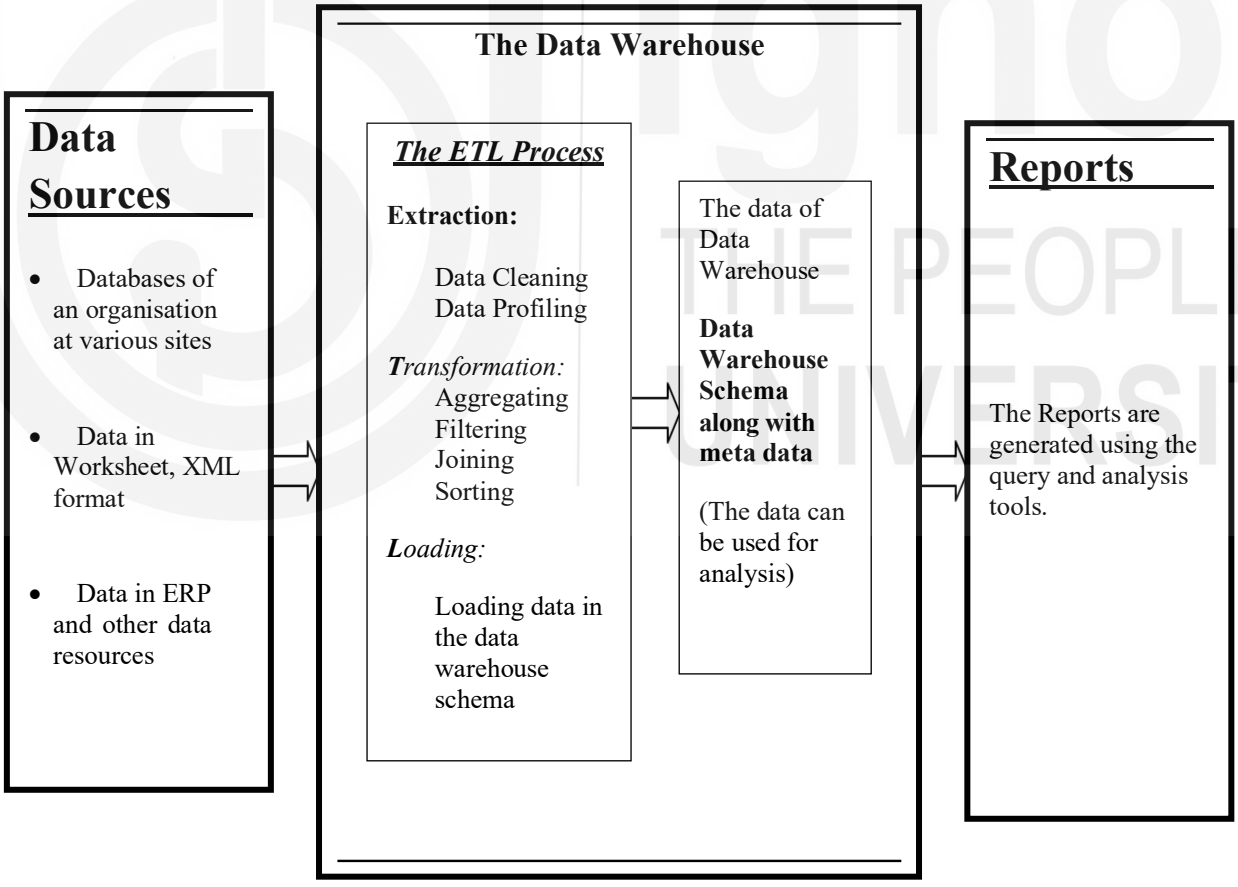

Figure 2: The Data Warehouse Architecture

14.3.1 The Data Sources

The data of the data warehouse can be obtained from many operational systems. A data warehouse interacts with the environment that provides most of the source data for the data warehouse. By the term environment, we mean traditionally developed database

systems and other applications. In a large installation, hundreds or even thousands of these database systems or files-based systems exist with plenty of redundant data.

The warehouse database obtains most of its data from such different forms of legacy systems – files and databases. Data may also be sourced from external sources as well as other organisational systems. This data needs to be integrated into the warehouse. But how do you integrate the data of these large numbers of operational systems into the data warehouse system? You need the help of ETL tools to do so. These tools capture the data that is required to be put in the data warehouse.

Data in Data Warehouse

The basic characteristics of the data in a data warehouse can be described in the following way:

i) **Subject Oriented:** The first characteristic of the data warehouse's data is that its design and structure can be oriented to important objects of the organisation. These objects for a university data warehouse can be STUDENT, PROGRAMME, REGIONAL CENTRES etc., whereas the operational systems may be designed around applications and functions such as ADMISSION, EXAMINATION and RESULT DECLARATIONS (in the case of a University). Refer to *Figure 3.*

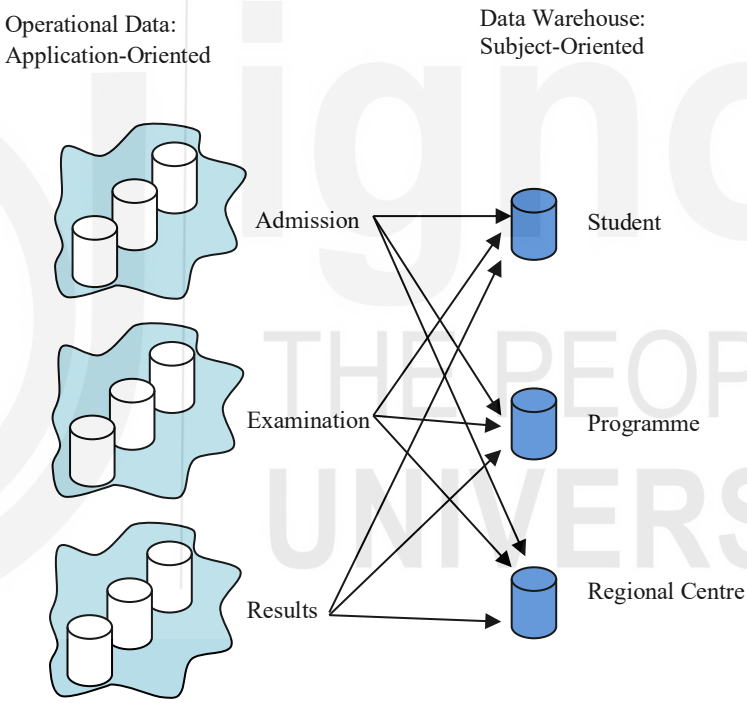

Figure 3: Operations system data vs Data warehouse data

ii) Integrated: Integration means bringing together data from multiple dissimilar operational sources on the basis of an enterprise data model. A data model of a data warehouse can be a basic template that uniquely defines the organisation's key data items. Data integration requires:

- Standardising the data naming and definition: for example, "student enrolment number" can be standardised over the data of the entire university.
- Standardising of encodings: for example, the first two digits of an enrolment number represent the year of admission.
- Standardising the Measurement of variables: for example, all the units would be expressed in the metric system and all monetary details would be given in Indian Rupees.

iii) **Time-Variant:** The third defining characteristic of the data in a data warehouse is that it is time-variant or historical in nature. The entire data in the data warehouse is/was accurate at some point in time. This is in contrast with operational data that changes over a shorter time period. *Figure 4* defines this characteristic of a data warehouse.

OPERATIONAL DATA	DATA WAREHOUSE DATA
It is the current value data	Contains a snapshot of historical data
Time span of data = 60-90 days	Time span of data = 5-10 years or more
Data can be updated in most cases	After making a snapshot of the data record cannot be updated
May or may not have a timestamp	Will always have a timestamp

Figure 4: Characteristics of data of operational data and data in a data warehouse

iv) **Non-volatile:** Data loaded in a data warehouse is subsequently scanned and used but is not updated in the same classical ways as the operational system's data, which is updated through the transaction processing cycles.

Metadata Directory

The metadata directory component defines the repository of the information stored in the data warehouse. The metadata can be used by general users as well as data administrators. It contains the following information:

i) the structure of the contents of the data warehouse data,
ii) the source of the data,
iii) the data transformation processing requirements, such that data can be passed from the legacy systems into the data warehouse database,
iv) the process summarisation of data,
v) the data extraction history, and
vi) how the data needs to be extracted from the data warehouse.

14.3.2 Data Extraction, Transformation and Loading (ETL)

The first step in data warehousing is to perform data extraction, transformation, and loading of data into the data warehouse. This is called ETL, which is Extraction, Transformation, and Loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into a target data warehouse.

What happens during the ETL Process?

The following are the sub-processes of the ETL process of the data warehouse:

Data Extraction: The ETL is a three-stage process. During the *Extraction* phase, the desired data is identified and extracted from many different sources. These sources may be different databases or non-databases. The process of extraction sometimes involves some basic transformation. For example, if the data is being extracted from two Sales databases where the sales in one of the databases are in Dollars and the other in Rupees, then a simple transformation would be required in the data. The size of the extracted data may vary from hundreds of kilobytes to hundreds of gigabytes, depending on the data sources and business systems.

The *extraction* process involves data cleansing and data profiling. Data cleansing can be defined as the process of removal of inconsistencies in the data. For example, the

state name may be written in many ways, and they can be misspelt too. For example, the state Uttar Pradesh may be written as U.P., UP, Uttar Pradesh, Utter Pradesh etc. The cleansing process may try to correct the spellings as well as resolve such inconsistencies. But how does the cleansing process do that? One simple way may be to create a Database of the States with possible fuzzy matching algorithms that may map various variants into one state name. Thus cleansing the data to a great extent.

Data profiling involves creating the necessary data from the point of view of a data warehouse application. Another concern here is to eliminate duplicate data. For example, an address list collected from different sources may be merged and purged to create an address profile with no duplicate data.

Data Transformation: One of the most time-consuming tasks - data *transformation* follows the extraction stage. The data transformation process includes the following:

- Use of data filters,
- Data validation against the existing data,
- Checking data duplication, and
- Information aggregation.

Transformations are useful for transforming the source data according to the requirements of the data warehouse. The process of transformation should ensure the quality of the data that needs to be loaded into the target data warehouse. Some of the common transformations are:

Filter Transformation: Filter transformations are used to filter the rows in a mapping that do not meet specific conditions. For example, the list of employees of the Sales department who made sales above Rs.50,000/- may be filtered out.

Joiner Transformation: This transformation is used to join the data of one or more different tables that may be stored in two different locations and could belong to two different sources of data that may be relational or other data formats like XML.

Aggregator Transformation: Such transformations perform aggregate calculations on the extracted data. Some such calculations may find the sum or average.

Sorting Transformation: requires creating an order in the required data based on the application requirements of the data warehouse.

Data Loading: Once the data for the data warehouse is properly extracted and transformed, it is *loaded* into a data warehouse. This process requires creation and execution of programs that perform this task. One of the key concerns here is to propagate source data updates. Sometimes, this problem is equated to the problem of maintenance of the materialised views.

When should we perform the ETL process for the data warehouse? ETL process should normally be performed at night or when the load on the operational systems is low. **Please note** that the integrity of the extracted data can be ensured by synchronising different operational applications feeding the data warehouse and the data in the data warehouse.

☞ Check Your Progress – 1

1) What is a Data Warehouse?

……

……

……

2) What are the characteristics of data in a data warehouse?

……

……

……

3) What is ETL? What are the different transformations that are needed during the ETL process?

……

……

……

14.4 MULTIDIMENSIONAL DATA MODEL OF A DATA WAREHOUSE

A data warehouse is a huge collection of data. Such data may involve grouping of data on multiple attributes. For example, the enrolment data of the students at a University may be represented using a student schema such as:

Student_enrolment (year, programme, region, number)

Some data values for such schema are (Also refer to *Figure 5*, which shows this data):

- In the year 2002, BCA enrolment at Region (Regional Centre Code) RC-07 (Delhi) was 350.
- In the year 2003, BCA enrolment in Region RC-07 was 500.
- In the year 2002, MCA enrolment in all the regions was 8000.

Please note that for representing the value of a number of students, you need to refer to three attributes: year, programme and region. Each of these attributes is identified as a dimension attribute. Thus, the data of the *Student_enrolment* table can be modelled using three-dimensional attributes (year, programme, region) and a measure attribute (number). Such kind of data is referred to as multidimensional data. Thus, a data warehouse may use multidimensional matrices referred to as a data cube model. If the dimensions of the matrix are greater than three, then it is called a hypercube. Figure 5 represents the multidimensional data of a university:

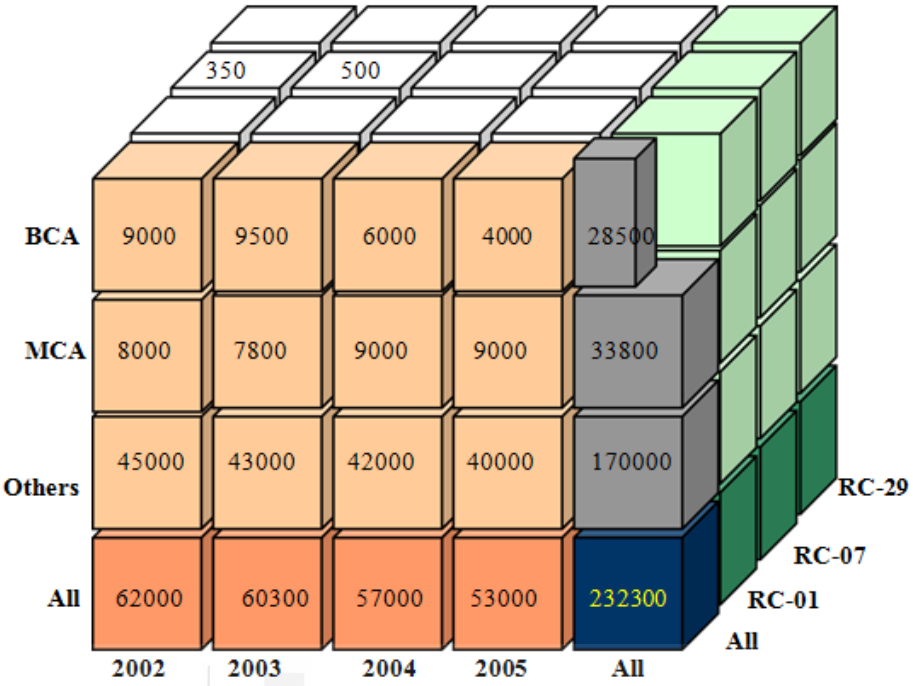

Figure 5: Sample multidimensional data

Multidimensional data may be a little difficult to analyse. Therefore, Multidimensional data may be displayed on a certain pivot, for example, consider the following table:

Region: ALL THE REGIONS				
	BCA	**MCA**	**Others**	**All the Programmes**
2002	9000	8000	45000	62000
2003	9500	7800	43000	60300
2004	6000	9000	42000	57000
2005	4000	9000	40000	53000
ALL the Years	**28500**	**33800**	**170000**	**232300**

Figure 6: Pivot table on Programme and Year dimension.

The table given above shows the multidimensional data in *cross-tabulation*. This is also referred to as a ***pivot table***. **Please note that** cross-tabulation is a two-dimensional structure. Therefore, if the data warehouse has three dimensions, then the cross-tabulation will be done on only two dimensions and the third dimension would be kept as ALL. For example, the table above has two dimensions Year and Programme, the third dimension Region has a fixed value ALL for the given table. The last row of Figure 6 is on one dimension, viz. Programme.

Now, the question is, how can multidimensional data be represented in a data warehouse? or, more formally, what is the schema for multidimensional data?

Three common multidimensional schemas are the Star schema, the Snowflake schema and Galaxy or Fact Constellation schema. In this Unit, we will define the Star and Snowflake schema. Galaxy schema contains multiple fact tables. You can find more detail on this schema in the further readings. A multidimensional storage model contains two types of tables: the dimension tables and the fact table. The dimension tables have tuples of dimension attributes, whereas the fact tables have one tuple each for a recorded fact. Let us demonstrate this with the help of an example. Consider the University data warehouse where one of the data tables is the *Student_enrolment* table. The three dimensions in such a case would be:

- Year
- Programme, and
- Region

The star schema for such a data is shown in *Figure* 7.

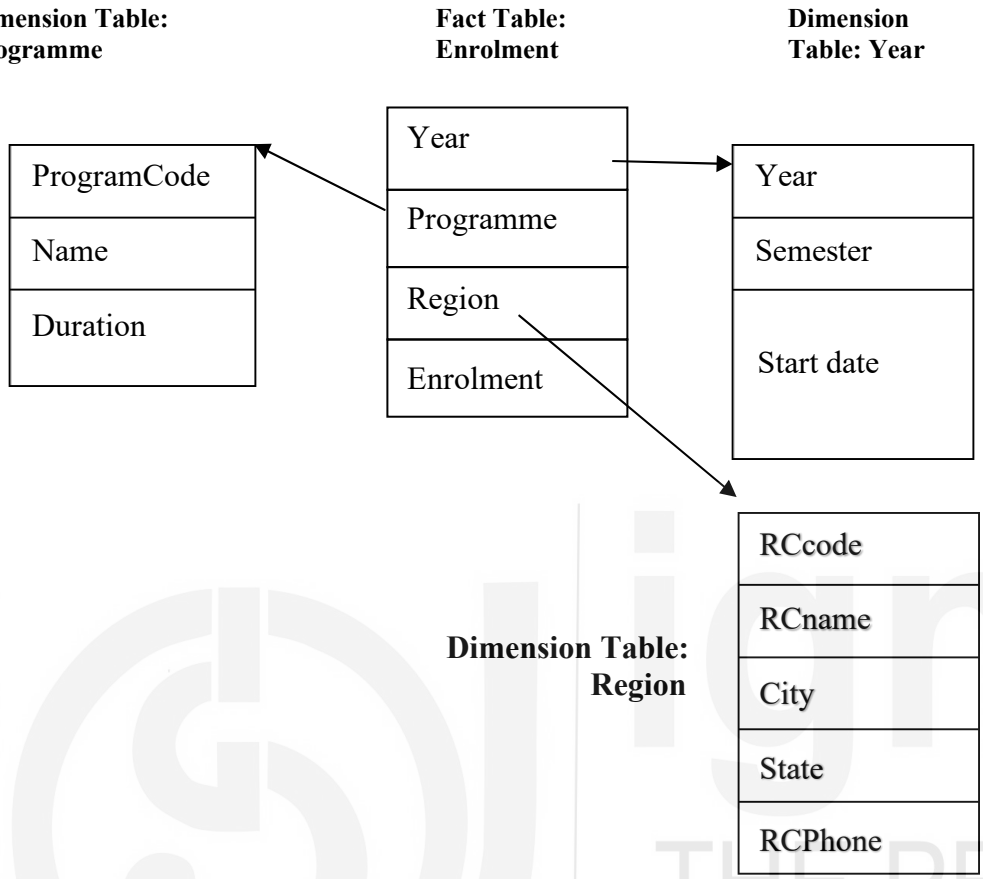

Figure 7: A Star Schema

Please note that in *Figure 7,* the fact table points to different dimension tables, thus, ensuring the reliability of the data. Please also note that each Dimension table is a table for a single dimension only and that is why this schema is known as a Star schema. However, a dimension table may not be normalised. Thus, a new schema named the Snowflake schema was created. A Snowflake schema has normalised but hierarchical dimensional tables. For example, consider the Star schema shown in *Figure* 7, if, in the Region dimension table, the value of the field Rcphone is multivalued, then the Region dimension table is not normalised. Thus, we can create a Snowflake schema for such a situation (Refer to Figure 8).

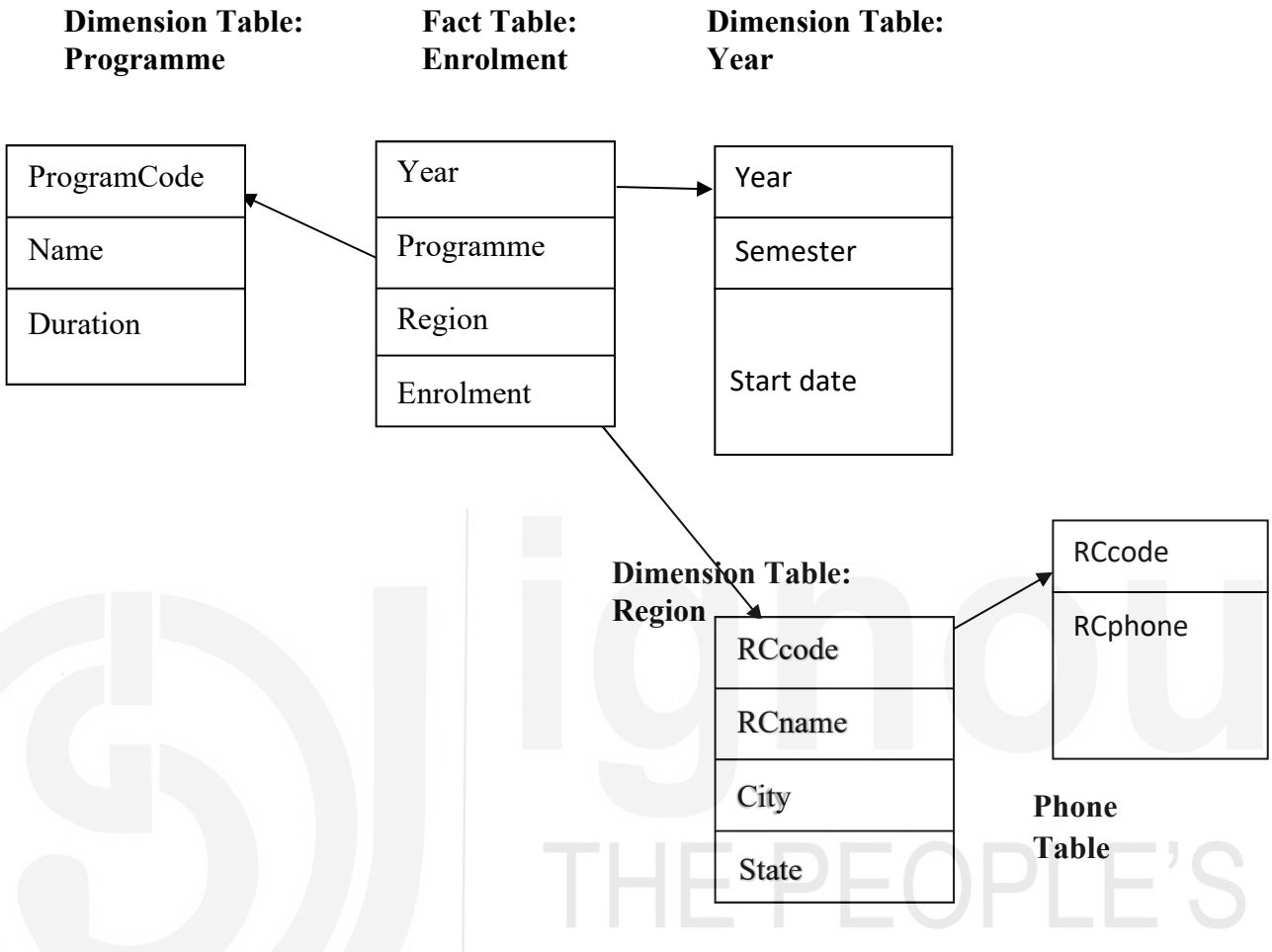

Figure 8: Snowflake Schema

Data warehouse storage can also utilise indexing to support high-performance access. As the data of a data warehouse is non-volatile, the data warehouse allows storage and access to summary data.

A data warehouse is an integrated collection of data and can help the process of making better business decisions. Several tools and methods are available to process the data of a data warehouse to create information and knowledge that supports business decisions. Two such techniques are Decision-support systems and Online Analytical processing. A detailed discussion on these topics is beyond. The scope of this unit. In this unit, we define data mining, which can extract useful information from a data warehouse.

14.5 DATA MINING TECHNOLOGY

Data is growing at a phenomenal rate today and users expect more sophisticated information from data. There is a need for techniques and tools that can automatically generate useful information and knowledge from large volumes of data. Data mining is one such technique of generating hidden information from the large data.

Data mining can be defined as: “an automatic process of extraction of non-trivial or implicit or previously unknown but potentially useful information or patterns from data in large databases, data warehouses or in flat files”.

Data mining uses the data of a data warehouse, which is well equipped for providing data as input for data mining. The advantages of using the data of a data warehouse for data mining are listed below:

- Data quality and consistency are essential for data mining to ensure the accuracy of the predictive models. Data is loaded in a data warehouse after data extraction, cleaning and transformation; therefore, is good quality data.
- As stated earlier, the data of a data warehouse is integrated from multiple data sources for a specific purpose and, therefore, is suited for data mining.
- Some data mining techniques would require aggregated or summarised data. A data warehouse contains such data.

As defined earlier, data mining generates potentially useful information or patterns from data. In fact, the information generated through data mining can be used to create knowledge. So let us first define the three terms: data, information and knowledge.

1. **Data (Symbols)** are raw facts, for example, "Miral" is a name, BCA is a programme, and 2200105000 is an enrolment number.
2. **Information:** Information is the processed data. It provides answers to "who", "what", "where", and "when" questions. For example, Miral is a BCA student of IGNOU having enrolment number "2200105000". He has scored 761 marks out of 1000.
3. **Knowledge:** Knowledge is the application of data and information, it answers the "how" questions. This is not explicit in the database. A student who scores more than 075% marks is brilliant; therefore, Miral is brilliant.

Data Mining Approaches

The approaches to data mining are based on the type of information/ knowledge to be mined. We will emphasise three different approaches: Classification, Clustering, and Association Rules.

The classification task maps data tuples into predefined groups or classes. The task of clustering is to group tuples with similar attribute values into a new class. So, classification is supervised by the goal attribute, while clustering is an unsupervised classification. The task of association rule mining is to find relationships between/among data values of a set of transactional data. Its original application was on "market basket data".

In most of these approaches, a notion of distance measure is used. A distance measure is used to find the distance or dissimilarity between objects. The two most common distance measures are:

- Euclidean distance: $\text{edis}(t_i,t_j) = \sqrt{\sum_{h=1}^{k} (t_{ih} - t_{jh})^2}$
- Manhattan distance: $\text{mdis}(t_i,t_j) = \sum_{h=1}^{k} \left|(t_{ih} - t_{jh})\right|$

where t_i and t_j are tuples and h can take the values from 1 to k, each of which represents a different attribute that is being used to compute the distance.
In the next section, we discuss the three data mining approaches.

☞ Check Your Progress – 2

1) What is a dimension? How is it different from a fact table?

……

……

……

2) How is the Snowflake schema different from the Star schema?

……

……

……

3) Define what data mining is.

……

……

……

4) What are different data mining tasks?

……

……

……

14.6 CLASSIFICATION

The classification task maps data tuples into predefined groups or classes.

Given a database/dataset consisting of tuples t_i, where i varies from 1 to n, i.e. D=$\{t_1, t_2, \ldots, t_n\}$;
and a set of known classes C=$\{C_1, \ldots, C_m\}$, where $m >> n$.

The classification problem is to map each t_i to a C_i.

Some simple examples of classification are:

- Teachers classify students' marks data into a set of grades as A, B, C, D, or F.
- You can clarify the height of a set of students in the classes: tall, medium or short.

Basic Principle:

1. The classification involves learning using the training data, which is the data that has already been assigned to one of the classes. This training results in a classification model.
2. This model is then tested using the test data to find the effectiveness of the model. The test data also has assigned classes, which are checked against the class predicted by the model. The accuracy of the model can be ascertained on the basis of correctly predicted classes of the test data.
3. A good model is then used to classify the data that has not been classified.

Some of the common techniques used for classification are Decision Trees, Neural Networks etc. In this section, we present two examples of classification.

14.6.1 Classification Using Distance (K-Nearest Neighbour)

This approach places items in the class to which they are "closest" by determining the distance of an item from a class. Classes are represented by a central point called centroid. The K-nearest neighbour algorithm has the following steps:

1) Create a training data set consisting of attributes or features that would be used for classifying data and the identified classes for these attributes. Figure 9 shows an example training data set.

2) Defines the number of *near items* (items that have less distance to the attributes of concern) from the training data that should be used to classify data. The value of K should be <= $\sqrt{Number\_of\_Training\_Items}$

3) A new item is placed in the class in which most of its *near items* are placed.

Example: Consider the following data, which classifies each person's class <Short, Medium, Tall> depending upon height attribute.

Name	Height	Class
Sunita	1.6m	Short
Ram	2.0m	Tall
Namita	1.9m	Medium
Radha	1.88m	Medium
Jully	1.7m	Short
Arun	1.85m	Medium
Shelly	1.6m	Short
Avinash	1.7m	Short
Sachin	2.2m	Tall
Manoj	2.1m	Tall
Sangeeta	1.8m	Medium
Anirban	1.95m	Medium
Krishna	1.9m	Medium
Kavita	1.8m	Medium
Pooja	1.75m	Medium

Figure 9: Sample Height data with classification

1) You are given the task of classifying the tuple xyz <XYZ, 1.6> using the data that is given to you.

2) The *height* attribute is used for distance calculation, and suppose K=5, then the following are the five nearest tuples to the tuple xyz. Please note that we have used the Manhattan distance on attribute *height* as a measure of classification.

Height	Class
1.6m	Short
1.7m	Short
1.6m	Short
1.7m	Short
1.75m	Medium

3) On examination of the tuples above, we classify the tuple xyz<XYZ, 1.6> to the *Short* class since most of the nearest tuples belong to the *Short* class.

Thus, in K-nearest neighbour classification, the classification is controlled by the neighbours.

14.6.2 Decision Tree

Given a data set D = $\{t_1, t_2 \ldots, t_n\}$, where tuple $t_i = < A_1, A_2, \ldots A_h >$, that is, each tuple is represented by *h* attributes. Also, let us suppose that the classes are C = $\{C_1, \ldots, C_m\}$, then the Decision or Classification Tree is a tree associated with D such that:

- The tree consists of internal and leaf nodes. A label using an attribute A_i is assigned to each internal node.
- An arc from a parent node is labelled with a predicate on the attribute label of the parent node.
- Every leaf node has a class label.

The basic steps in the Decision Tree are as follows:

- Building the tree by using the training set dataset/database.
- Applying the tree to the new dataset/database.

This decision tree can then be used for classification. We show an example of a decision tree without giving an algorithm for drawing it. You may refer to further reading for details.

Consider the following data in which the *Position* attribute acts as a predicted class, and department and salary are attributes of determining the class of a tuple.

Department	Age	Salary	Position
Personnel	31-40	Medium Range	Boss
Personnel	21-30	Low Range	Assistant
Personnel	31-40	Low Range	Assistant
MIS	21-30	Medium Range	Assistant
MIS	31-40	High Range	Boss
MIS	21-30	Medium Range	Assistant
MIS	41-50	High Range	Boss
Administration	31-40	Medium Range	Boss
Administration	31-40	Medium Range	Assistant
Security	41-50	Medium Range	Boss
Security	21-30	Low Range	Assistant

Figure 10: Sample data for classification

You may analyse the data given in Figure 10 on each individual attribute. The following tables present the analysis of class Position on the attributes Age, Department and Salary, respectively.

Age	Assistant	BOSS
21-30	4	0
31-40	2	3
41-50	0	2

Department	Assistant	BOSS
Personal	2	1
MIS	2	2
Administration	1	1
Security	1	1

Salary	Assistant	BOSS
Low Range	3	0
Medium Range	3	3
High Range	0	2

Out of the three attributes, the Age attribute predicts the Position class the best, as it predicts that all the person in the range 21-30 are Assistants and all the person in the age group 41-50 are Boss. Thus, it should be used as the first splitting attribute. However, for a person in the age range of 31-40, the Postion cannot be defined. So, we

have to find the spitting attribute for this age range 31-40. The tuples that belong to this range are as follows:

Tuples Only for 31-40 age range

Department	Salary	Position
Personnel	Medium Range	Boss
Personnel	Low Range	Assistant
MIS	High Range	Boss
Administration	Medium Range	Boss
Administration	Medium Range	Assistant

The tuples as given above can be mapped as:

Department	Assistant	BOSS
Personal	1	1
MIS	0	1
Administration	1	1

Salary	Assistant	BOSS
Low Range	1	0
Medium Range	1	2
High Range	0	1

The Salary attribute can perform better prediction than the Department attribute, so we select Salary as the next splitting attribute. In the middle range Salary, the Position class is not defined while for other ranges it is defined. So, we have to find the spitting attribute for this middle range. Since only Department attribute is left, so, Department will be the next splitting attribute. Now, the tuples that belong to this salary range are as follows:

Department	Position
Personnel	Boss
Administration	Boss
Administration	Assistant

In the Personnel department, all person are Boss, while, in the Administration there is a tie between the classes. So, the person can be either Boss or Assistant in the Administration department. The final decision tree, based on presented data, for predicting the Position class is as follows:

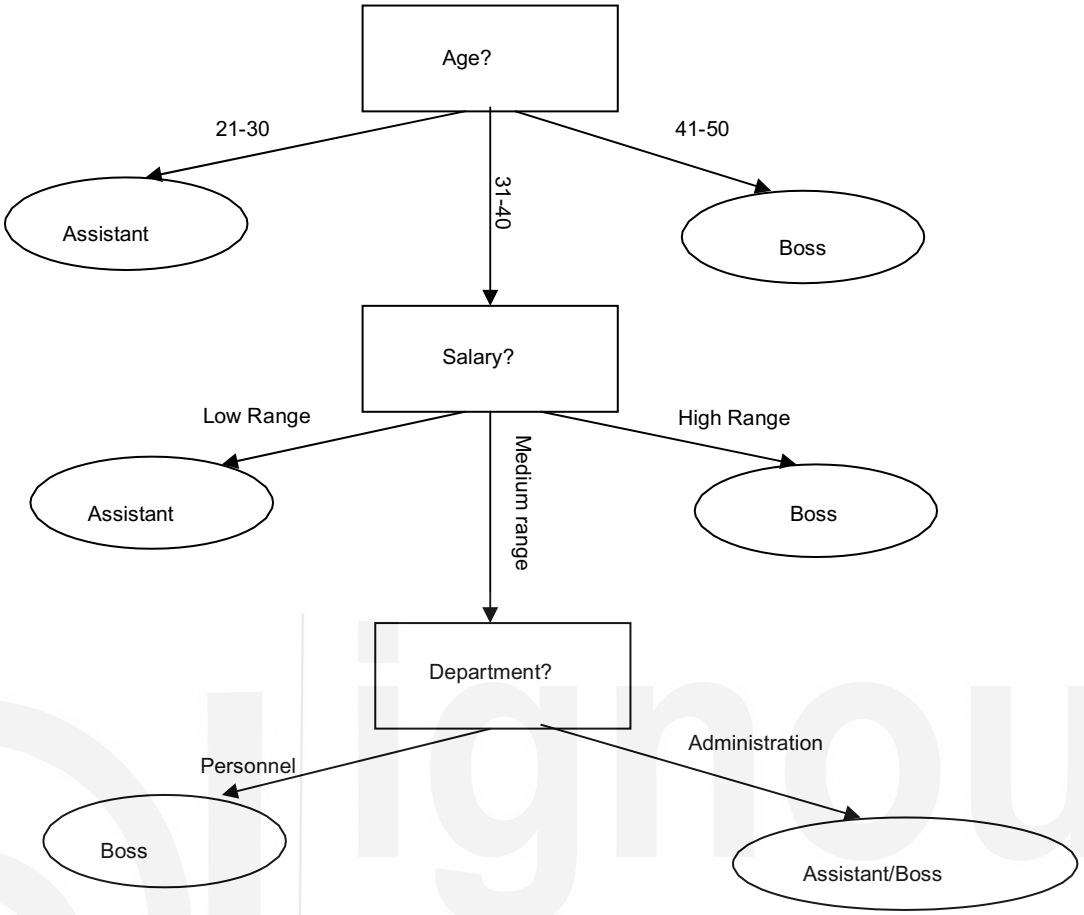

Figure 11: The decision tree for the sample data of Figure 10

Now, you can use the decision tree to predict the Position of a person if his/her Age, Salary and Department attributes are known.

There are many more techniques of classification, however, they are beyond the scope of this unit, you may refer to further readings for these.

14.7 CLUSTERING

Clustering is grouping tuples with similar attribute values into the same group. Given a database D=$\{t_1,t_2,\ldots,t_n\}$ of tuples and an integer value K, the Clustering problem is to define a mapping where each tuple t_i is assigned to one cluster K_j, $1<=j<=K$.

Unlike the classification problem, clusters are not known in advance. The user has to enter the value of the number of clusters K. In other words, a *cluster* can be defined as the collection of data objects that are similar in nature, as per certain defining properties, but these objects are dissimilar to the objects in other clusters. Clustering is a very useful exercise, especially for identifying similar groups in the given data. Such data can be about buying patterns, geographical locations, web information and many more. For example, you can use clustering to segment the customer database of a departmental store based on similar buying patterns. Similarly, to identify similar Web usage patterns, you may use clustering. Some of the clustering issues are as follows:

- **Outlier handling:** How will the outliers be handled? (Outliers are data objects that have values far beyond the average data limits of those data objects). Outlier handling requires answering the question: Whether an outlier be

considered or left aside while calculating the clusters?

- **Dynamic data:** How will you handle dynamic data?
- **Interpreting results:** How will the result be interpreted?
- **Evaluating results:** How will the result be calculated?
- **Number of clusters:** How many clusters will you consider for the given data?
- **Data to be used:** whether you are dealing with quality data or noisy data? If the data is noisy, how is it to be handled?
- **Scalability:** Whether the algorithm that is used is to be scaled for small as well as large data sets/databases.

There are many algorithms for clustering. However, we will discuss only one basic algorithm. You can refer to more details on clustering from further readings.

Partitioning Clustering

The partitioning clustering algorithm constructs K partitions from a given set of n objects of data. Here, $K \leq n$, and each partition must have at least one data object while one object belongs to **only one** of the partitions. A partitioning clustering algorithm normally requires users to input the desired number of clusters, K. We define one such technique called K-means clustering with the help of an example.

K-Means clustering: In K-Means clustering, initially a set of K clusters is randomly chosen. Then iteratively, items are moved among sets of clusters until the desired set of clusters is reached. A high degree of similarity among elements in a cluster is obtained by using this algorithm. For this algorithm, a set of clusters $K_i = \{t_{i1}, t_{i2}, \ldots, t_{im}\}$ is chosen. A cluster mean is computed for each cluster using the equation:

$$m_i = (1/m)\,(t_{i1} + \ldots + t_{im}) \qquad \ldots\ldots\ldots\ldots\ldots\ldots(1)$$

Where t_i represents the tuples and m represents the mean. This mean is used to refine the clusters further. We explain this algorithm with the help of an example.

Example: Consider an input set is given as:

Input set= {1, 2, 3, 5, 10, 15, 16,18, 22, 30} and $K = 2$.

Find the cluster of input values.

Step 1: Randomly assign means to two clusters, K_1 and K_2, as m_1=3 and m_2=5

Assign Input set values to clusters based on the smallest distance to the mean. This will result in:

K_1= {1, 2, 3} and K_2= {5, 10, 15, 16, 18, 22, 30}

Step 2: Compute the mean (use the formula given in equation 1) of the clusters after the assignment of the input set. This will result in m_1=2 and m_2=*16.57*.

Redefine clusters as per the closest mean from the mean of the cluster. For example, for the mean of step 2, input value 5 is closer to 2, so add it to K_1. Thus, the revised clusters are:

K_1= {1, 2, 3, 5} and K_2= {10, 15, 16, 18, 22, 30}.

Step 3: Compute the mean for the latest cluster assignment. This will result in m_1=2.75 and m_2=18.5.

Repeat the process of finding the cluster as per the closest mean from the mean of the cluster. The revised clusters remain as:

K_1= {1, 2, 3, 5, 10} and K_2= {15, 16, 18, 22, 30}.

Step 4: Compute the mean, it will be m_1=4.2, m_2=20.2

No change in clusters, so STOP. The final clusters are:

K_1= {1, 2, 3, 5, 10} and K_2= {15, 16, 18, 22, 30}.

You can refer to more details on the clustering algorithms from further readings.

☞ Check Your Progress – 3

1) What is the classification of data? Give some examples of classification.

……

……

……

2) What is clustering?

……

……

……

3) How is clustering different from classification?

……

……

……

14.8 ASSOCIATION RULE MINING

The task of association rule mining is to find certain association relationships among a set of items in a dataset/database. The association relationships are described as association rules. In association rule mining there are two measurements, *support* and *confidence*. The confidence measure indicates the rule's strength, while support corresponds to the frequency of the pattern.

A typical example of an association rule created by data mining often termed "market basket data" is "While shipping for basic groceries, it has been found that about 80% of the customers who purchase bread from the store also purchase butter along with."

Applications of association rule mining include cache customisation, advertisement personalisation, store layout designing, customer segmentation etc. All these applications try to determine the associations between data items, if it exists, to optimise performance.

Formal Definition:
Given a set of items $I = \{i_1, i_2, \ldots, i_m\}$ and a set of transactions T, where a transaction T_i comprises of items and is a subset I. Each transaction is identified by a TID. For the given sets, an association rule is defined as $X \rightarrow Y$, where both X and Y are the subsets of I such that $X \cap Y = \emptyset$.

The support (s) for the stated association rule $X \rightarrow Y$ is defined as the percentage of transactions in T that contains $X \cup Y$.

The confidence (c) for the stated association rule $X \rightarrow Y$ is defined as the percentage of transactions that include X, also include Y.

Support indicates how frequently the pattern occurs, while confidence indicates the strength of the rule.

The objective of association rule mining is to find all those rules which have better support than the defined minimum support and better confidence than the defined

minimum confidence. The association rule mining consists of two sub-problems:

(1) Find the frequent item sets (*FreqItem*) having support more than a predetermined minimum support.
(2) Derive association rules from *FreqItem*, which have confidence more than the minimum confidence.

There are a lot of ways to find the large item sets, but in this unit, we will only discuss the Apriori Algorithm.

Apriori Algorithm: For finding frequent item sets.

The Apriori algorithm applies the concept that all the subsets of a frequent itemset should be frequent. The Apriori algorithm generates the candidate item sets from the item sets that were found to be frequent in the previous iteration of the algorithm.

This algorithm first finds the frequent item sets having just 1 item, which are combined to form item sets of 2 items and so on. The algorithm iterations terminates when no additional higher item sets can be generated.

Notations that are used in the Apriori algorithm are given below:

k-itemset	An itemset having k items
L_k	Set of frequent k-itemset (those with minimum support)
C_k	Set of candidates k-itemset (for finding frequent item sets)

The Apriori algorithm is implemented as an iterative function. It takes L_{k-1} as an input parameter and returns L_k. It consists of a join step and a pruning step. It is explained with the help of the following example:

Example: Finding frequent item sets:
Consider the following transactions and find the frequent item sets by applying the Apriori algorithm assuming minimum support (*s*) =30%.

Transaction ID	Item(s) purchased
1	Shirt, Trouser
2	Shirt, Trouser, Coat
3	Coat, Tie, Tiepin
4	Coat, Shirt, Tie, Trouser
5	Trouser, Belt
6	Coat, Tiepin, Trouser
7	Coat, Tie
8	Shirt
9	Shirt, Coat
10	Shirt, Handkerchief

Figure 12: Sample Transactions data

The method of finding the frequent itemset is shown in the Figure 13.

Iteration Number	Candidates	Frequent itemset (s ≥ 3, i.e. 30% of 10 Transactions)
1	C_1={Belt 1, Coat 6, Handkerchief 1, Shirt 6, Tie 3, Tiepin 2, Trouser 5 }	L_1={Coat 6, Shirt 6, Tie 3, Trouser 5 }
2	C_2= {{Coat, Shirt} 3, {Coat, Tie} 3, {Coat, Trouser} 3, {Shirt, Tie} 1, {Shirt, Trouser} 3, {Tie, Trouser} 1 }	L_2={{Coat, Shirt} 3, {Coat, Tie} 3, {Coat, Trouser} 3, {Shirt, Trouser} 3 }
3	C_3= {{Coat, Shirt, Trouser} 2}	L_3= Ø

Figure 13: Frequent Itemset for different k using Apriori algorithm

In pass number 1, you may notice that Belt in purchased in only one (5th transaction) of all the 10 transaction that in why it has a value 1. Similarly, the support of each item is computed from the transaction data.

Likewise, in pass 2 {Coat, shirt} together are purchased in transactions 2, 4, 9.

Also please note that set C_2 is computed by joining set L_1 with itself, which will result in

C_2: {{Coat, Shirt}, {Coat, Tie}, {Coat, Trouser}, {Shirt, Tie}, {Shirt, Trouser}, {Tie, Trouser}}

Set L_2 is computed after finding the support of each element of set C_2 as shown in Figure 13.

The calculation of 3-itemsets is mentioned below:

Join operation on L_2 onto itself yields 3 item sets as:

{{Coat, Shirt, Tie}, {Coat, Shirt, Trouser}, {Coat, Tie, Trouser}}

However, the Prune operation removes two of these items from the candidate set C_3 due to the following reasons:

- {Coat, Shirt, Tie} is pruned as one of its subset {Shirt, Tie} is not in L_2
- {Coat, Shirt, Trouser} is retained as {Coat, Shirt}, {Coat, Trouser} and {Shirt, Trouser} all three are in L_2
- {Coat, Tie, Trouser} is pruned as its subset{Tie, Trouser} is not in L_2

The algorithm terminates, as L_3 is a NULL set. Thus, the frequent 2 items are:

L_2={{Coat, Shirt}, {Coat, Tie}, {Coat, Trouser}, {Shirt, Trouser}}

Finding Association Rules: Assuming minimum confidence (*c*) of 60%.

Confidence for rule Coat→Shirt

$$= \frac{\text{Transactions containing } \{Coat, Shirt\}}{\text{Transactions containing only } \{Coat\}} \times 100$$

$$= \frac{3}{6} \times 100 = 50\%$$

Confidence for rule Shirt →Coat $= \frac{3}{6} \times 100 = 50\%$

Confidence for rule Coat →Tie $= \frac{3}{6} \times 100 = 50\%$

Confidence for rule Tie →Coat $= \frac{3}{3} \times 100 = 100\%$

Confidence for rule Coat →Trouser = $\frac{3}{6} \times 100 = 50\%$

Confidence for rule Trouser →Coat = $\frac{3}{5} \times 100 = 60\%$

Confidence for rule Shirt →Trouser = $\frac{3}{6} \times 100 = 50\%$

Confidence for rule Trouser →Shirt = $\frac{3}{5} \times 100 = 60\%$

Only the following three association rules fulfil the confidence criteria of confidence >= 60%.

Tie →Coat; Trouser →Coat; Trouser →Shirt

Thus, the Arpioi algorithm is able to determine the frequent item sets, which can be used to determine the association rules. One of the major advantages of the Apriori algorithm is that it is a very easy algorithm to implement; however, it requires the transaction database to be memory resident.

14.9 APPLICATIONS OF DATA MINING

Some of the applications of data mining are as follows:

- **Marketing and sales data analysis:** A company can use customer transactions in their database to segment the customers into various types. Such companies may launch products for specific customer types.
- **Investment analysis:** Customers can look at the areas where they can get good returns by applying data mining.
- **Loan approval:** Companies can generate rules for giving loans depending upon the dataset they have. On that basis, they may decide to whom and what amount of loan should be given.
- **Fraud detection:** By finding the correlation between faults, new faults can be detected by applying data mining.
- **Network management:** By analysing patterns generated by data mining for the networks and their faults, the faults can be minimised as well as future hardware and software needs of the network can be predicted.
- **Brand Loyalty:** Given a customer and the product he/she uses, predict whether the customer will change their products.

☞ Check Your Progress – 4

1) What is association rule mining?

……

……

……

2) What are the applications of data mining in the banking domain?

……

……

……

3) Apply the Apriori algorithm for generating frequent itemset in the following dataset:

Transaction ID	Items purchased
T_A	PI_2 PI_5
T_B	PI_1 PI_3
T_C	PI_1PI_2 PI_3PI_4
T_D	PI_1PI_2 PI_3

……

……

……

14.10 SUMMARY

This unit first introduces the concepts of data warehousing systems. The data warehouse is a technology that collects operational data from several operational systems, refines it and stores it in its own multidimensional model, such as Star schema or Snowflake schema. The data of a data warehouse can be indexed and can be used for analyses through data mining. Data mining is the process of automatic extraction of interesting but not known information in large databases. Basic data mining tasks are Classification, Clustering and Association rules. The classification task maps data into predefined classes. Clustering task groups objects with similar properties/behaviour into the same group. Association rules find the association relationship among a set of objects. Data mining can be applied in many areas, whether it is Games, Marketing, Bioscience, Loan approval, Fraud detection etc.

Please go through further readings for more details on data warehousing and data mining.

14.11 SOLUTIONS/ANSWERS

☞ Check Your Progress – 1

1) A Data Warehouse is a repository of processed but integrated information that can be used for queries and analysis. Data and information are extracted from heterogeneous sources. It is subject-oriented, time-variant integrated data, which is not changed once put in a data warehouse.

2)

- Integrated data
- Subject-oriented data
- Time-variant data
- Non-volatile

3) ETL is Extraction, transformation, and loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into the target data warehouse. The following are some of the transformations that may be used during ETL:

- Filter Transformation
- Joiner Transformation
- Aggregator transformation
- Sorting transformation.

☞ Check Your Progress – 2

1) A dimension may be equated with a reference object. For example, in a sales organisation, the dimensions may be *salesperson*, *product* and *period* of information. Each of these is a dimension. The fact table will represent the fact relating to the dimensions. For this example, a fact table will include *sales* (in rupees) made by a particular *salesperson* for a specific *product* for a certain *period*. Fact is actual data. A fact, thus, represents an aggregation of relational data on the dimensions.

2) The primary difference between them is that the Snowflake schema uses a normalised dimensional table.

3) Data mining is the process of automatic extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) information or patterns from large data stored in large databases or data warehouses.

4) Different data-mining tasks are - Classification, Clustering and Association Rule Mining.

☞ Check Your Progress – 3

1) The classification task maps data into predefined groups or classes. Data are represented as tuples, which consist of a set of predicating attributes and a goal attribute. The task is to discover some kind of relationship between the predicating attributes and the goal attribute so that the discovered knowledge can be used to predict the class of new tuple(s).

   Some examples of classification are:

   Classification of students' grades depending on their marks in previous examinations.

   Classification of customers as good or bad customers in a bank.

2) The task of clustering is to group the tuples with similar attribute values into the same class. Given a database of tuples and an integer value *K*, Clustering defines mapping, such that tuples are mapped to *K* different clusters.

3) In classification, the classes are predetermined, but in the case of clustering, the groups are not predetermined. Only the number of clusters is decided by the user.

☞ Check Your Progress – 4

1) The basic idea of association rule mining is to find unknown and interesting associations among various data items.

2) Data mining applications in banking are as follows:

   1) Detecting patterns of fraudulent credit card use.
   2) Identifying good customers.
   3) Determining whether to issue a credit card to a person or not.
   4) Finding hidden correlations between different financial indicators.

3) The dataset D given for the problem is:

Transaction ID	Items purchased
T_A	PI_2 PI_5
T_B	PI_1 PI_3
T_C	PI_1 PI_2 PI_3 PI_4
T_D	PI_1 PI_2 PI_3

Assuming the minimum support as 50% for calculating the large item sets. As we have 4 transactions, at least 2 transactions should have the data item.

First Scan:

C_1: PI_1:3, PI_2:3, PI_3:3, PI_4:1, PI_5:1

L_1: PI_1:3, PI_2:3, PI_3:3

C_2: PI_1PI_2, PI_1PI_3, PI_2PI_3

Second Scan:

C_2: PI_1PI_2:2, PI_1PI_3:3, PI_2PI_3:2

L_2: PI_1PI_2:2, PI_1PI_3:3, PI_2PI_3:2

C_3: $PI_1PI_2PI_3$

Pruned C_3: $PI_1PI_2PI_3$

Third scan

L_3: $PI_1PI_2PI_3$: 2

Frequent item sets L={L_1, L_2, L_3}

14.12 FURTHER READINGS

1) *Data Mining Concepts and Techniques*, J Han, M Kamber, Morgan Kaufmann Publishers, 2001.

2) *Data Mining*, A K Pujari, 2004.

UNIT 15 NoSQL DATABASE

Structure **Page Nos.**

15.0 Introduction
15.1 Objectives
15.2 Introduction to NoSQL
  15.2.1 What is NoSQL
  15.2.2 Brief History of NoSQL Databases
  15.2.3 NoSQL Database Features
  15.2.4 Difference between RDBMS and NoSQL
15.3 Types of NoSQL Databases
  15.3.1 Column Based
  15.3.2 Graph Based
  15.3.3 Key-Value Pair Based
  15.3.4 Document Based
15.4 Summary
15.5 Solutions/Answers
15.6 Further Readings

15.0 INTRODUCTION

In the previous Unit of this Block, you have gone through the concept of relational database management systems and data warehousing. However, these technologies are somewhat slower for scalable web applications. NoSQL databases arose because databases at the time were not able to support the rapid development of scalable web-based applications.

NoSQL databases have changed the manner in which data is stored and used, despite the fact that relational databases are still commonly employed. Most applications come with features like Google-style search, for instance. The growth of data, online surfing, mobile use, and analytics have drastically altered the requirements of contemporary databases. These additional requirements have spurred the expansion of NoSQL databases, which now include a range of types such as key-value, document, column, and graph.

In this Unit, we will discuss the many kinds of NoSQL databases, including those that are built on columns, graphs, key-value pairs, and documents respectively.

15.1 OBJECTIVES

After going through this unit, you should be able to:

- define what is NoSQL;
- differentiate between NoSQL and SQL;
- explain the basic features of Column based NoSQL Database;
- explain the Graph-based NoSQL Database;
- explain the Key-value pair based NoSQL Database and
- explain the Document based NoSQL Database.

15.2 INTRODUCTION TO NoSQL

Databases are a crucial part of many technological and practical systems. The phrase "NoSQL database" is frequently used to describe any non-relational database. NoSQL is sometimes referred to as "non SQL," but it is also referred to as "not only SQL." In either case, the majority of people agree that a NoSQL database is a type of database that stores data in a format that is different from relational tables.

Whenever you want to use the data, it must first be saved in a particular structure and then converted into a usable format. On the other hand, there are some circumstances in which the data are not always presented in a structured style, which means that their schemas are not always rigorous. This unit provides an in-depth look into NoSQL and the features that make it unique.

15.2.1 What is NoSQL?

NoSQL is a way to build databases that can accommodate many different kinds of information, such as key-value pairs, multimedia files, documents, columnar data, graphs, external files, and more. In order to facilitate the development of cutting-edge applications, NoSQL was designed to work with a variety of different data models and schemas.

The great functionality, ease of development, and performance at scale offered by NoSQL have helped make it a popular name. NoSQL is sometimes referred to as a non-relational database due to the numerous data handling features it offers. Due to the fact that it does not adhere to the guidelines established by Relational Database Management Systems (RDBMS), you cannot query your data using conventional SQL commands. We can think of such well-known examples as MongoDB, Neo4J, HyperGraphDB, etc.

15.2.2 Brief History of NoSQL Databases

In the late 2000s, as the price of storage began to plummet, No-SQL databases began to gain popularity. No longer is it necessary to develop a sophisticated, difficult-to-manage data model to prevent data duplication. Because developers' time was quickly surpassing the cost of data storage, NoSQL databases were designed with efficiency in mind.

Table 1: History of Databases

Year	Database Solutions	Company / Database Technology
1970-2000	Mainly RDBMS related	Oracle, IBM DB2, SQL Server, MySQL
2000-2005	DotCom boom – new scale solutions, start of NoSQL dev, whitepapers	Google, Facebook, IBM, amazon
2005-2010	New open source & mainstream databases	Cassandra, Riak, Apache Hbase, neo4j, MongoDB, CouchDB, Redis

2010 onwards	Adoption of Cloud	DBaaS (Database as a Service)

As storage costs reduced significantly, the quantity of data that applications were required to store and query grew. This data came in all forms— structured, semi-structured, and unstructured — and sizes making it practically difficult to define the schema in advance. NoSQL databases give programmers a great deal of freedom by enabling them to store enormous amounts of unstructured data.

In addition, the Agile Manifesto was gaining momentum, and software developers were reconsidering their approach to software development. They were beginning to understand the need of being able to quickly adjust to ever-evolving requirements. They required the flexibility to make rapid iterations and adjustments to all parts of their software stack, including the underlying database. They were able to achieve this flexibility because of NoSQL databases.

The use of the public cloud as a platform for storing and serving up data and applications was another trend that arose, as cloud computing gained popularity. To make their applications more robust, to expand out rather than up, and to strategically position their data across geographies, they needed the option to store data across various servers and locations. These features are offered by some NoSQL databases like MongoDB.

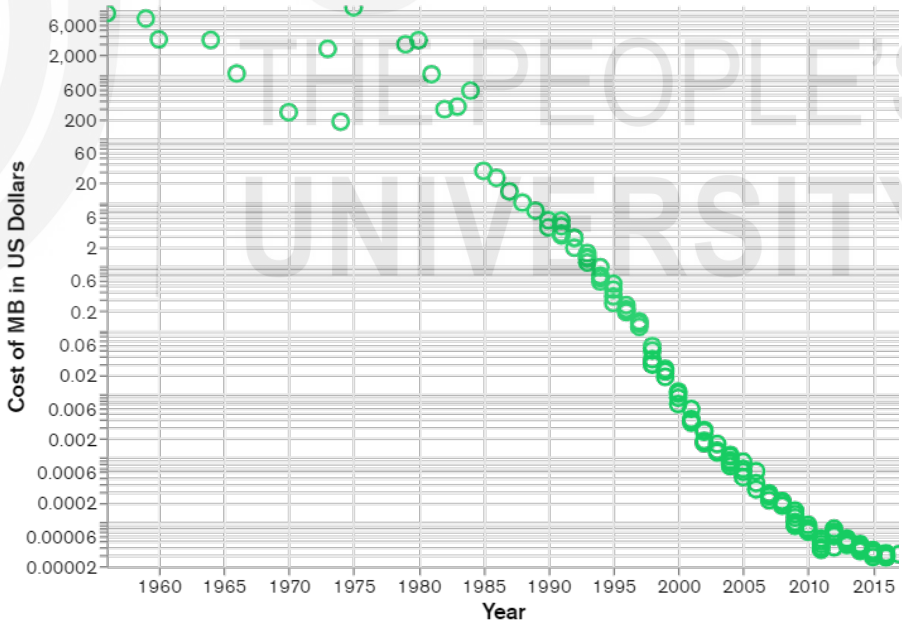

Figure 1: Cost per MB of Data over Time (Log Scale)
(Adapted from https://www.mongodb.com)

15.2.3 NoSQL database features

Every NoSQL database comes with its own set of one-of-a-kind capabilities. The following are general characteristics shared by several NoSQL databases:

- Schema flexibility

- Horizontal scaling
- Quick responses to queries as a result of the data model
- Ease of use for software developers

15.2.4 Difference between RDBMS and NoSQL

The differences and similarities between the two DBMSs are as follows:

- For the most part, NoSQL databases fall under the category of non-relational or distributed databases, while SQL databases are classified as Relational Database Management Systems (RDBMS).
- Databases that use the Structured Query Language (SQL) are table-oriented, while NoSQL databases use either document-oriented or key-value pairs or wide-column stores, or graph databases.
- Unlike NoSQL databases, which have dynamic or flexible schema to manage unstructured data, SQL databases have a strict or static schema.
- Structured data is stored using SQL, whereas both structured and unstructured data can be stored using NoSQL.
- SQL databases are thought to be scalable in a vertical direction, whereas NoSQL databases are thought to be scalable in a horizontal direction.
- Increasing the computing capability of your hardware is the first step in the scaling process for SQL databases. In contrast, NoSQL databases scale by distributing the load over multiple servers.
- MySQL, Oracle, PostgreSQL, and Microsoft SQL Server are all examples of SQL databases. BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB, and Neo4j are a few examples of NoSQL databases.

Vertical scalability is required for SQL databases. This means that an excessive amount of load must be able to be managed by increasing the amount of CPU, SSD, RAM, GPU, etc. on your server. When it comes to NoSQL databases, the ability to scale horizontally is one of their defining characteristics. This means that the addition of additional servers will make the task of managing demand more manageable.

☞ **Check Your Progress 1**

1) What is NoSQL?

……

……

……

2) What are the features of NoSQL databases?

……

……

……

3) Differentiate between the NoSQL and SQL.

……

……

……

15.3 TYPES OF NoSQL DATABASES

In this section, we will discuss the many classifications of NoSQL databases. There are typically four types of NoSQL databases:

1) Column-based: Instead of accumulating data in rows, this method organizes it all together into columns, which makes it easier to query large datasets.
2) Graph-based: These are systems that are utilized for the storage of information regarding networks, such as social relationships.
3) Key-value pair based: This is the simplest sort of database, in which each item of your database is saved in the form of an attribute name (also known as a "key") coupled with the value.
4) Document-based: Made up of sets of key-value pairs that are kept in documents.

15.3.1 Column Based

A column store, in contrast to a relational database, is arranged as a set of columns, rather than rows. This allows you to read only the columns you need for analysis, saving memory space that would otherwise be taken up by irrelevant information. Because columns are frequently of the same kind, they are able to take advantage of more efficient compression, which makes data reading even quicker. The value of a specific column can be quickly aggregated using columnar databases.

Although columnar databases are excellent for analytics, because of the way they publish data, it is challenging for them to remain consistent because writes to all the columns need several write events on the disk. However, this problem never arises with relational databases because row data is continuously written to disk.

How Does a Column Database Work?

A columnar database is a type of database management system (DBMS) that allows data to be stored in columns rather than rows. It is accountable for reducing the amount of time needed to return a certain query. Additionally, it is accountable for the significant enhancement of the disk I/O performance. Both data analytics and data warehousing benefit from it. Additionally, the primary goal of a Columnar Database is to read and write data in an efficient manner. Column-store databases include Casandra, CosmoDB, Bigtable, and HBase, to name a few.

Columnar Database Vs Row Database:
When processing big data analytics and data warehousing, there are a number of different techniques that can be used, including columnar databases and row databases. But they each take a different method.

For instance:
• Row Database: “Customer 1: Name, Address, Location". (The fields for each new record are stored in a long row).
• Columnar Database: “Customer 1: Name, Address, Location”. (Each field has its own set of columns). Refer Table 2 for relational database example.

Table 2: Relational database: an example

ID Number	First Name	Last Name	Amount
A01234	Sima	Kaur	4000
B03249	Tapan	Rao	5000
C02345	Srikant	Peter	1000

In a Columnar DBMS, the data will be stored in the following format:
A01234, B03249, C02345; Sima, Tapan, Srikant; Kaur, Rao, Peter; 4000, 5000, 1000.

In a Row-oriented DBMS, the data will be stored in the following format:
A01234, Sima, Kaur, 4000; B03249, Tapan, Rao, 5000; C02345, Srikant, Peter, 1000.

Columnar databases: advantages

The use of columnar databases has various advantages:

- Column stores are highly effective in compression, making them storage efficient. This implies that you can conserve disk space while storing enormous amounts of data in a single column.
- Aggregation queries are fairly quick with column-store databases because the majority of the data is kept in a column, which is beneficial for projects that need to execute a lot of queries quickly.
- Load times are also quite good; a table with a billion rows can be loaded in a matter of seconds. This suggests that you can load and query practically instantly.
- A great deal of versatility because columns do not have to resemble one another. The database would not be affected if you add new or different columns, however, updating all tables is necessary to input whole new record queries.
- Overall, column-store databases are excellent for analytics and reporting due to their quick query response times and capacity to store massive volumes of data without incurring significant costs.

Column databases: Disadvantages

While there are many benefits to adopting column-oriented databases, there are also a few drawbacks to keep in mind.

- It takes a lot of time and effort to create an efficient indexing schema.
- Incremental data loading is undesirable and is to be avoided, if at all possible, even though this might not be a problem for some users.
- This applies to all forms of NoSQL databases, not just those with columns. Web applications frequently have security flaws, and the absence of security features in NoSQL databases does not help. If security is your top goal, you should either consider using relational databases or, if it's possible, use a clearly specified schema.
- Due to the way data is stored, Online Transaction Processing (OLTP) applications are incompatible with columnar databases.

Are columns databases always NoSQL?

Before we conclude, we should note that column-store databases are not always NoSQL-only. It is frequently argued that column-store belongs firmly in the NoSQL camp because it differs so much from relational database approaches. The debate between NoSQL and SQL is generally quite nuanced, therefore this is not usually the case. They are essentially the same as SQL techniques when it comes to column-store databases. For instance, keyspaces function as schema, so schema management is still necessary. A NoSQL data store's keyspace contains all column families. The concept is comparable to relational database management systems' schema. There is typically only one keyspace per program. Another illustration is the fact that the metadata occasionally resembles a conventional relational DBMS perfectly. Ironically, column-store databases frequently adhere to ACID and SQL standards. However, NoSQL databases are often either document-store or key-store, neither of which are column-store. Therefore, it is difficult to claim that column-store is a pure NoSQL system.

15.3.2 Graph Based

The initial hardware hurdles that made it feasible for SQL to handle vast quantities of data are no longer there, despite the fact that SQL is an excellent superb RDBMS and has been used for many years to manage massive amounts of data. As a result, NoSQL has rapidly emerged as the dominant form of contemporary database management and many of the largest websites, we rely on today, are powered by NoSQL, like Twitter's use of FlockDB and Amazon's DynamoDB.

A database that stores data using graph structures is known as a graph database. It represents and stores data using nodes, edges, and attributes rather than tables or documents. Relationships between the nodes are represented by the edges. This makes data retrieval simpler and, in many circumstances, only requires one action. Additionally, it works fantastically as a database for fast, threaded data structures like those used on Twitter

How does a Graph Database Work?

Graphs, which are not relational databases, rely heavily on the idea of multi-

relational data "pathways" for their functionality. However, the structure of graph databases is typically simple. They are largely made up of two elements:

- The Node: This represents the actual data itself. It may be the number of people who watched a video on YouTube; it could be the number of people who read a tweet; or it could even be fundamental information like people's names, addresses, and other such details.
- The Edge: This clarifies the real connection between the two nodes. It is interesting to note that edges can also have their own data, such as the type of connection between two nodes. Similar to edges, mentioned directions may also describe the direction in which the data is flowing.

Graph databases are mostly utilized for studying relationships. For instance, businesses might extract client information from social media using a graph database. For example, some organization might use a graph database to extract data about relationships between Person, Restaurant, and City, as shown in Figure 2.

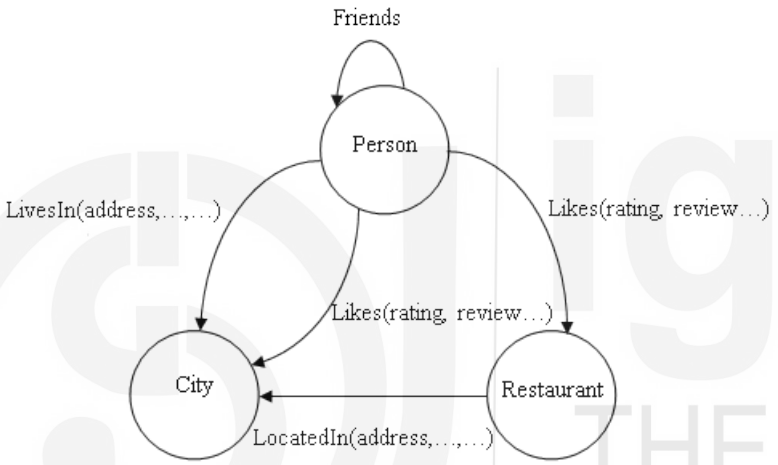

Figure 2. Different Nodes and Edges in Graph Database.
(Adapted from https://www.kdnuggets.com/)

When do we need Graph Database?

1) It resolves issues with many-to-many relationships. For example, many-to-many relationships include friends of friends.
2) When connections among data pieces are more significant. For example, there is a profile with some unique information, but the main selling point is the relationship between these different profiles, which is how you get connected inside a network.
3) Low latency with big amounts of data. The relational database's data sets will grow significantly as you add more relationships, and when you query it, its complexity will increase and it will take longer than usual. However, graph databases are specifically created for this purpose, and one can easily query relationships.

Now, let’s look at a more specific illustration to explain a group of people's complicated relationships. For example, five friends share a social network. These friends are Binny, Bhawna, Chaitaya, Manish, and Mohit. Their personal data may be kept in a graph database that resembles this, as shown in Figure 3 and Table 3:

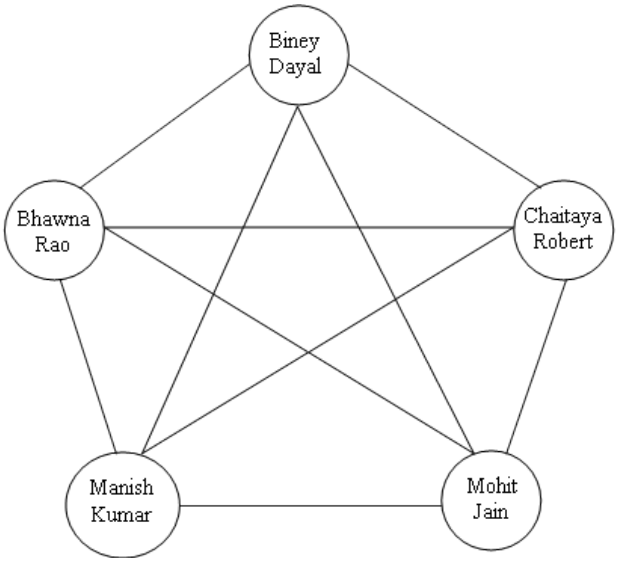

Figure 3. Example-Five friends sharing Social network.

Table 3: Relational database: an example

Id	Firstname	Lastname	Email	Mobile
1001	Biney	Dayal	binnya@example.com	8645212321
1002	Bhawna	Rao	bhawanrao@example.com	9645212323
1003	Chaitaya	Robert	chaitayarob@example.com	7645212356
1004	Manish	Kumar	mkumar@example.com	9955212320
1005	Mohit	Jain	mjain@example.com	9945212329

This means we will need yet another table to keep track of user relationships. Our friendship table (refer Table 4) will resemble the following:

Table 4: Friendship Table

user_id	friend_id
1001	1002
1001	1003
1001	1004
1001	1005
1002	1001
1002	1003
1002	1004
1002	1005
1003	1001
1003	1002
1003	1004
1003	1005
1004	1001
1004	1002
1004	1003
1004	1005
1005	1001
1005	1002
1005	1003
1005	1004

We won't go too deeply into the theory of the database's main key and foreign key. Instead, presume that the friendship table uses both friends' ids. Let's say that every member on our social network gets access to a feature that lets them view the personal information of their other users who are friends with them. This means that if Chaitaya were to ask for information, it would be regarding Biney, Bhawna, Manish and Mohit. We shall address this issue in a conventional

(relational database) manner. First, we need to locate Chaitaya's user id in the database's Users table (refer Table 5).

Table 5: Chaitaya's Record

Id	Firstname	Lastname	Email	Mobile
1003	Chaitaya	Robert	chaitanyarob@example.net	7645212356

We would now search the friendship table (refer Table 6) for all tuples with the user id of 3. The resulting relationship would look like this:

Table 6: Friendship Table for user id 3

user_id	friend_id
1003	1001
1003	1002
1003	1004
1003	1005

Let us now examine the time required for this Relational database strategy. This will be close to log (N) times, where N is the number of tuples in the friendship table. In this case, the database continues to keep the entries in sequential order based on their ids. So, in general, the time complexity for 'M' number of queries is M*log (N). Only, if we had used a graph database strategy the overall time complexity has been O (N). For the simple reason that once Chaitaya has been located in the database, all the rest of her friends may be found with a single click, as shown in Figure 4.

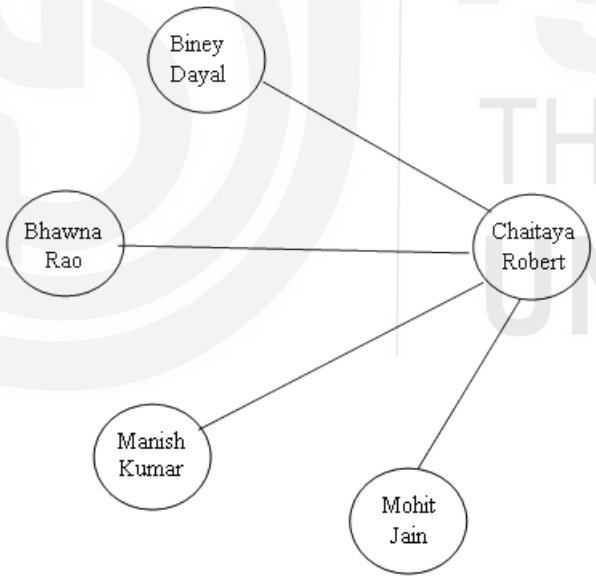

Figure 4. Accessing other data with a single click.

Graph Database Examples

Although graph databases are not as widely used as other NoSQL databases, there are a handful that have become de facto standards when discussing NoSQL:

Neo4j is both an open-source and an interestingly developed on Java graph database. It is considered to be one of the best graph databases. In addition to that, it comes with its own language known as Cypher, which is comparable to the declarative SQL language but is designed to work with graphs. In addition to Java, it supports a number of other popular programming languages, including Python, .NET, JavaScript, and a few others. Neo4j excels in applications such as

the administration of data centers and the identification of fraudulent activity.

RedisGraph is a graph module that is integrated into Redis, which is a key-value NoSQL database. RedisGraph was developed to have its data saved in RAM for the same reason that Redis itself is constructed on in-memory data structures. As a result, a graph database with excellent speed and quick searching and indexing is created. RedisGraph also makes use of Cypher, which is ideal if you're a programmer or data scientist looking for greater database flexibility. Applications that require blazing-fast performance are the main uses.

OrientDB It is interesting to note that OrientDB supports graph, document store, key-value store, and object-based data formats. Having stated that, the graph model, which uses direct links between databases, is used to hold all of the relationships. Although it does not use Cypher, OrientDB is open-source and developed in Java, just like Neo4j and the two prior graph databases. OrientDB is designed to be used in situations when many data models are necessary, and as a result, it is optimized for data consistency as well as minimizing data complexity.

15.3.3 Key-value pair Based

Key-value stores are perhaps the most widely used of the four major NoSQL database formats because of their simplicity and quick performance. Let us examine key-value stores' operation and application in more detail. With some of the most well-known platforms and services depending on them to deliver material to users with lightning speed, NoSQL has grown in significance in our daily lives. Of course, NoSQL includes a range of database types, but key-value store is unquestionably the most used.

Because of its extreme simplicity, this kind of data model is built to execute incredibly quickly when compared to relational databases. Furthermore, because key-value stores adhere to the scalable NoSQL design philosophy, they are flexible and simple to set up.

How Does a Key-Value Work?

In reality, key-value storage is quite simple. A value is saved with a key that specifies its location, and a value can be pretty much any piece of data or information. In reality, this design idea may be found in almost every programming language as an array or map object, refer Figure 5. The fact that it is persistently kept in a database management system makes a difference in this case.

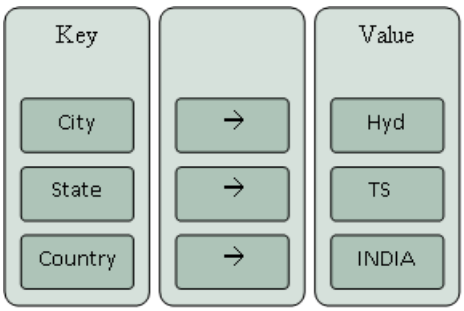

Figure 5. Example Key-Value database.

Popularity of key-value stores is due to the fact that information is stored as a single large piece of data instead of as discrete data. As a result, indexing the database is not really necessary to improve its performance. Instead, because of

the way it is set up, it operates more quickly on its own. Similar to that, it mostly uses the get, put, and delete commands rather than having a language of its own.

Of course, this has the drawback that the data you receive in response to a request is not screened. Under certain conditions, this lack of data management may be problematic, but generally speaking, the trade-off is worthwhile. Because key-value stores are both quick and reliable, the vast majority of programmers find ways to get around any filtering or control problems that may arise.

Benefits of Key-Value

Key-value data models, one of the more well-liked types of NoSQL data models, provide many advantages when it comes to creating a database:

Scalability: Key-value stores, like NoSQL in general, are infinitely scalable in a horizontal fashion, which is one of its main advantages over relational databases. This can be a huge advantage for sophisticated and larger databases compared to relational databases, where expansion is vertical and finite, as shown in Figure 6.

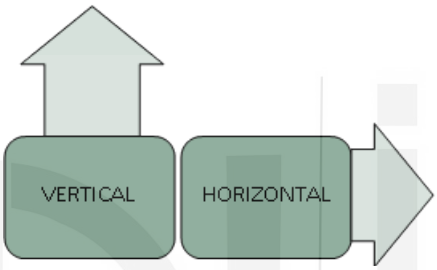

Figure 6. Horizontal and Vertical Scalability.

More specifically, partitioning and replication are used to manage this. Additionally, by avoiding things like low-overhead server calls, it decreases the ACID guarantees.

No/Simpler Querying: With key-value stores, querying is really not possible except in very particular circumstances when it comes to querying keys, and even then, it is not always practicable. Because there is just one request to read and one request to write, key-value makes it easier to manage situations like sessions, user profiles, shopping carts, and so on (due to the blob-like nature of how the data is stored). Similar to this, concurrency problems are simpler to manage because only one key needs to be resolved.

Mobility: Because key-value stores lack a query language, it is simple to move them from one system to another without modifying the architecture or the code. Thus, switching operating systems is less disruptive than switching relational databases.

When to Use Key-Value

Key-value stores excel in this area because traditional relational databases are not actually designed to manage a large number of read/write operations. Key-value can readily scale to thousands of users per second due to its scalability. Additionally, it can easily withstand lost storage or data because of the built-in redundancy.

As a result, key-value excels in the following instances:

- Profiles and user preferences
- Large-scale user session management
- Product suggestions (such as in eCommerce platforms)

- Delivery of personalized ads to users based on their data profiles
- Cache data for infrequently updated data

There are numerous other circumstances where key-value works nicely. For instance, because of its scalability, it frequently finds usage in big data research. Similar to how it works for web applications, key-value is effective for organizing player sessions in MMOG (massively multiplayer online game) and other online games.

Key-Value Database Examples

Some key-value database models, for instance, save information to a solid-state drive (SSD), while others use random-access memory (RAM). We depend on key-value stores on a daily basis in our lives since they are some of the most popular and frequently used databases. The fact is that some of the most popular and commonly used databases are key-value stores.

Amazon DynamoDB is most likely the database that is used the most often for key-value storage. In point of fact, study into Amazon DynamoDB was the impetus for the rise in popularity of NoSQL.

Aerospike is a free and open-source database that was designed specifically for use with in-memory data storage.

Berkeley DB: Another free and open-source database, Berkeley DB is a high-performance framework for storing databases, despite the fact that it has a very simple interface.

Couchbase: Text searches and querying in a SQL-like format are both possible with Couchbase, which is an interesting feature.

Memcached not only saves cached data in RAM, which helps websites load more quickly, but it is also free and open source.

Riak was designed specifically for use in the app development process, and it plays well with other databases and app platforms.

Redis: A database that serves as both a memory cache and a message broker.

15.3.4 Document Based

A non-relational database that stores data as structured documents is known as a document database (also known as a NoSQL document store). Instead of using standard rows and columns, JSON format is a more recent technique to store data. An XML or JSON file, or a PDF, are all examples of documents. NoSQL is everywhere nowadays; just look at Twitter and its use of FlockDB or Amazon and their use of DynamoDB. Figure 7 shows the difference between the Relational and Document Store model.

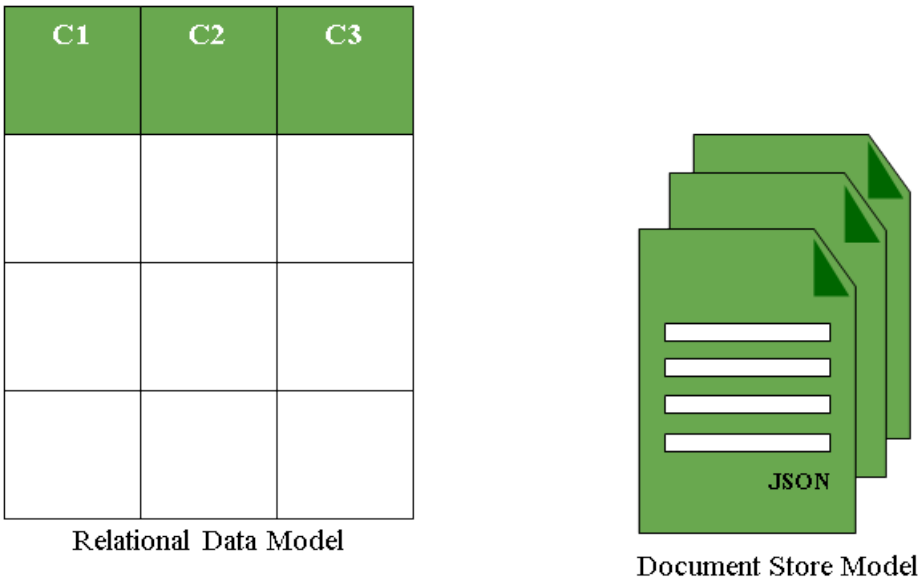

Figure 7: Relational Vs Document Store Model.

In spite of the fact that there are a great deal of data models, each of which contains hundreds of databases, the one we are going to investigate today is called Document-store. One of the most common database models now in use, document-store functions in a manner that is somewhat similar to that of the key-value model in the sense that documents are saved together with particular keys that access the information. Figure 8 (a) shows the document that holds information about a book. This file is a JSON representation of a book's metadata, which includes the book's BookID, Title, Author, and Year and Figure 8 (b) shows the same metadata for Key value database.

A Document

{
"BookID": "978-1449396091",
"Title": "DBMS",
"Author": "Raghu Ramakrishnan",
"Year": "2022",
}

(a)

Key	**Value**
BookID	**978-1449396091**
Title	**DBMS**
Author	**Raghu Ramakrishnan**
Year	**2022**

(b)

Figure 8: Example of Document and Key-value database

When to use a document database?

- When your application requires data that is not structured in a table format.
- When your application requires a large number of modest continuous reads and writes and all you require is quick in-memory access.
- When your application requires CRUD (Create, Read, Update, Delete) functionality.
- These are often adaptable and perform well when your application has to run across a broad range of access patterns and data kinds.

How does a Document Database Work?

It appears that document databases work under the assumption that any kind of information can be stored in a document. This suggests that you shouldn't have to worry about the database being unable to interpret any combination of data types. Naturally, in practice, most document databases continue to use some sort of schema with a predetermined structure and file format.

Document stores do not have the same foibles and limitations as SQL databases, which are both tubular and relational. This implies that using the information at hand is significantly simpler and running queries may also be much simpler. Ironically, you can execute the same types of operations in a document storage that you can in a SQL database, including removing, adding, and querying.

Each document requires a key of some kind, as was previously mentioned, and this key is given to it through a unique ID. This unique ID processed the document directly instead of being obtained column by column.

Document databases often have a lower level of security than SQL databases. As a result, you really need to think about database security, and utilizing Static Application Security Testing (SAST) is one approach to do so. SAST, examines the source code directly to hunt for flaws. Another option is to use DAST, a dynamic version that can aid in preventing NoSQL injections.

Document database advantages

One major benefit of document-store is that all of the data is stored in a single location, rather than being spread out over many interconnected databases. As a result, if you do not employ relational processes, you perform better than a SQL database.

- **Schema-less:** Because there are no constraints on the format and structure of data storage, they are particularly effective at keeping huge quantities of existing data.
- **Faster creation of document and maintenance:** The creation of a document is a fairly straightforward process, and apart from that, the upkeep requirements are virtually nonexistent.
- **Open formats**: It offers a relatively easy construction process that makes use of XML, JSON, and other formats.
- **Built-in versioning:** Because it contains built-in versioning, it means that when the documents expand in size, there is a possibility that they will also expand in complexity. Versioning makes conflicts less likely.

More precisely, document stores are excellent for the following applications because schema can be changed without any downtime or because you could not know future user needs:

- eCommerce giants (Like Amazon)
- Blogging platforms (such as Blogger, Tumblr)
- CMS (Content management systems) (Like WordPress, windows registry)
- Analytical platforms (such as Tableau, Oracle server)

Document databases' drawbacks

- **Weak Atomicity:** Multi-document ACID transactions are not supported. We will need to perform two different queries, one for each collection, in order to handle a change in the document data model involving two collections. This is where the atomicity criteria are violated.
- **Consistency Check Limitations:** A database performance issue may arise from searching for documents and collections that aren't linked to an author collection.
- **Security:** In today's world, many online apps do not have enough security, which in turn leads to the disclosure of critical data. Thus, web app vulnerabilities become a cause for concern.

Document databases examples

- One of the best NoSQL database engines is **MongoDB**, which is not only well-known but also uses JSON like format. It has its own query language.
- A search engine built on the document-store data architecture is **Elasticsearch**. Database searching and indexing may be accomplished using this straightforward and easy-to-learn tool.
- **CouchDB:** In addition to Ubuntu, it also works with the social networking site Facebook. It utilizes Javascript and is developed in the Erlang programming language.
- **BaseX** is a simple, open-source, XML-based DBM that makes use of Java.

☞ **Check Your Progress 2**

1) How Does a Column Database Work? Discuss.

……

……

……

2) What are the different Graph Database Examples?

……

……

……

3) Explain document based NoSQL database.

……

……

……

15.4 SUMMARY

This unit covered the fundamentals of NoSQL as well as the many kinds of NoSQL databases, such as those based on columns, graphs, key-value pairs, and

documents. Numerous businesses now use NoSQL. It is difficult to pick the best database platform. NoSQL databases are used by many businesses because of their ability to handle mission-critical applications while decreasing risk, data spread, and total cost of ownership.

Despite their incredible capability, column-store databases do have their own set of problems. Due to the fact that columns require numerous writes to the disk, for instance, the way the data is written results in a certain lack of consistency. Graph databases can be used to offer content in high-performance scenarios while producing threads that are simple to comprehend for the typical user, beyond merely expressive information in a graphical and effective way (such as in the case of Twitter). The simplicity of a key-value store is what makes it so brilliant. Although this has potential drawbacks, particularly when dealing with more complicated issues like financial transactions, it was designed specifically to fill in relational databases' inadequacies. We may create a pipeline that is even more effective by combining relational and non-relational technologies, whether we are working with users or data analysis. Document-store data models are quite popular and regularly used due to their versatility. It helps analytics by making it easy for firms to store multiple sorts of data for later use.

15.5 SOLUTIONS/ANSWERS

Check Your Progress 1

1) NoSQL is a way to build databases that can accommodate many different kinds of information, such as key-value pairs, multimedia files, documents, columnar data, graphs, external files, and more. In order to facilitate the development of cutting-edge applications, NoSQL was designed to work with a variety of different data models and schemas.

2)
   - Schema flexibility
   - Horizontal scaling
   - Quick responses to queries as a result of the data model
   - Ease of use for software developers

3) It is different in the following ways:
   - For the most part, NoSQL databases fall under the category of non-relational or distributed databases, while SQL databases are classified as Relational Database Management Systems (RDBMS).
   - Databases that use the Structured Query Language (SQL) are table-oriented, while NoSQL databases use either document-oriented or key-value pairs or wide-column stores, or graph databases.
   - Unlike NoSQL databases, which have dynamic or flexible schema to manage unstructured data, SQL databases have a strict, preset or static schema.
   - Structured data is stored using SQL, whereas both structured and unstructured data can be stored using NoSQL.
   - SQL databases are thought to be scalable in a vertical direction, whereas NoSQL databases are thought to be scalable in a horizontal direction.

- Increasing the computing capability of your hardware is the first step in the scaling process for SQL databases. In contrast, NoSQL databases scale by distributing the load over multiple servers.
- MySQL, SQLite, Oracle SQL, PostgreSQL, and Microsoft SQL Server are all examples of SQL databases. BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB, and Neo4j are a few examples of NoSQL databases.

Check Your Progress 2

1) A columnar database is a type of database management system (DBMS) that allows data to be stored in columns rather than rows. It is accountable for reducing the amount of time needed to return a certain query. Additionally, it is accountable for the significant enhancement of the disk I/O performance. Both data analytics and data warehousing benefit from it. Additionally, the primary goal of a Columnar Database is to read and write data in an efficient manner. Column-store databases include Casandra, CosmoDB, Bigtable, and HBase, to name a few. Also, refer 15.3.1.

2) Graph Database Examples:

- **Neo4j** is both an open-source and an interestingly developed on Java graph database. It is considered to be one of the best graph databases in the world. In addition to that, it comes with its own language known as Cypher, which is comparable to the declarative SQL language but is designed to work with graphs. In addition to Java, it supports a number of other popular programming languages, including Python, .NET, JavaScript, and a few others. Neo4j excels in applications such as the administration of data centres and the identification of fraudulent activity.
- **RedisGraph** is a graph module that is integrated into Redis, which is a key-value NoSQL database. RedisGraph was developed to have its data saved in RAM for the same reason that Redis itself is constructed on in-memory data structures. As a result, a graph database with excellent speed and quick searching and indexing is created. RedisGraph also makes use of Cypher, which is ideal if you're a programmer or data scientist looking for greater database flexibility. Applications that require blazing-fast performance are the main uses.
- **OrientDB:** It's interesting to note that OrientDB supports graph, document store, key-value store, and object-based data formats. Having stated that, the graph model, which uses direct links between databases, is used to hold all of the relationships.

3) It is generally agreed that document stores, which are a sort of NoSQL database, are the most advanced of the available options. They use JSON as their data storage format, which is different from the more traditional rows and columns layout. Most of the day-to-day activities that we carry out on the internet are supported by NoSQL databases. NoSQL is everywhere nowadays; just look at Twitter and its use of FlockDB or Amazon and their use of DynamoDB. Also, refer 15.3.4.

15.6 FURTHER READINGS

1) Next Generation Databases: NoSQL and Big Data 1st ed. Edition, G. Harrison, Apress, December 26, 2015.
2) Shashank Tiwari, Professional NoSQL, 1st Edition, Wrox, September 2011.
3) https://www.kdnuggets.com/

UNIT 16 EMERGING DATABASE MODELS

Structure **Page no.**

16.0 Introduction
16.1 Objectives
16.2 Distributed Databases
    16.2.1 Data Fragmentation and Replication
    16.2.2 Distributed Query Processing
16.3 Active Databases
16.4 XML for Data Representation
16.5 Blockchain Databases
16.6 Multimedia Database
16.7 Use of Databases in Web Applications
16.8 Summary
16.9 Solutions / Answers

16.0 INTRODUCTION

With the advent of relational database systems in the 1970s, database technology became popular in the industry due to the simplicity of database technologies and the availability of SQL for querying the database. However, just a relational model was not sufficient. Many advanced database technologies have become available. We have already discussed some of these technologies, like object-oriented database management systems, data warehousing and mining and NoSQL databases, in the first three Units of this Block. This Unit discusses several advanced database technologies.

This Unit first introduces you to the distributed database systems needed to address the needs of organisations that have distributed data. A distributed database system allows the distribution of fragments of data over a number of database sites. It also supports query processing, which may involve several sites. These concepts are detailed in this Unit. The Unit also introduces you to the concepts of Active databases, XML data and Blockchain technology. Finally, the Unit defines how a database system can be used as a backend to a web application.

This Unit gives a brief introduction to these database technologies. You may refer to the further readings for more details on these technologies.

16.1 OBJECTIVES

After going through this Unit, you should be able to:

- define the need for a distributed database system;
- explain the data fragmentation and replication in distributed databases;
- define the distributed query processing;
- define the features of the active database management system;
- explain the document creation using XML;
- explain the characteristics of Blockchain systems;
- list the characteristics of multimedia database
- use the web database in a web application.

16.2 DISTRIBUTED DATABASES

Many commercial organisations use a database system to manage large amounts of transactional data. These database systems have multiple users and run on a high-performance centralised computer system. These systems may allow geographically distributed users to connect to the database using a network. However, with the increase in the volume of database transactions and user interactions, use of a number of database servers, which may process local data, may be more efficient. This led to creation of distributed database management systems. A distributed database management system (DDBMS) manages several database servers, which may be dispersed at various geographical locations but store the data of a single database system. For example, consider a sample student relational schema given in Figure 1; how can this schema be represented in a distributed database? Figure 2 shows an example of a distribution of the student database over a few possible locations.

Student							
Enrollment No.	Name	Father's Name	Highest Qualification	Email	Phone	Regional Centre	Programme

Fee			
Enrollment No.	Semester	Amount Paid	Date of Payment

Subject		
Enrollment No.	Semester	Subject Code

Figure 1: A sample student relation

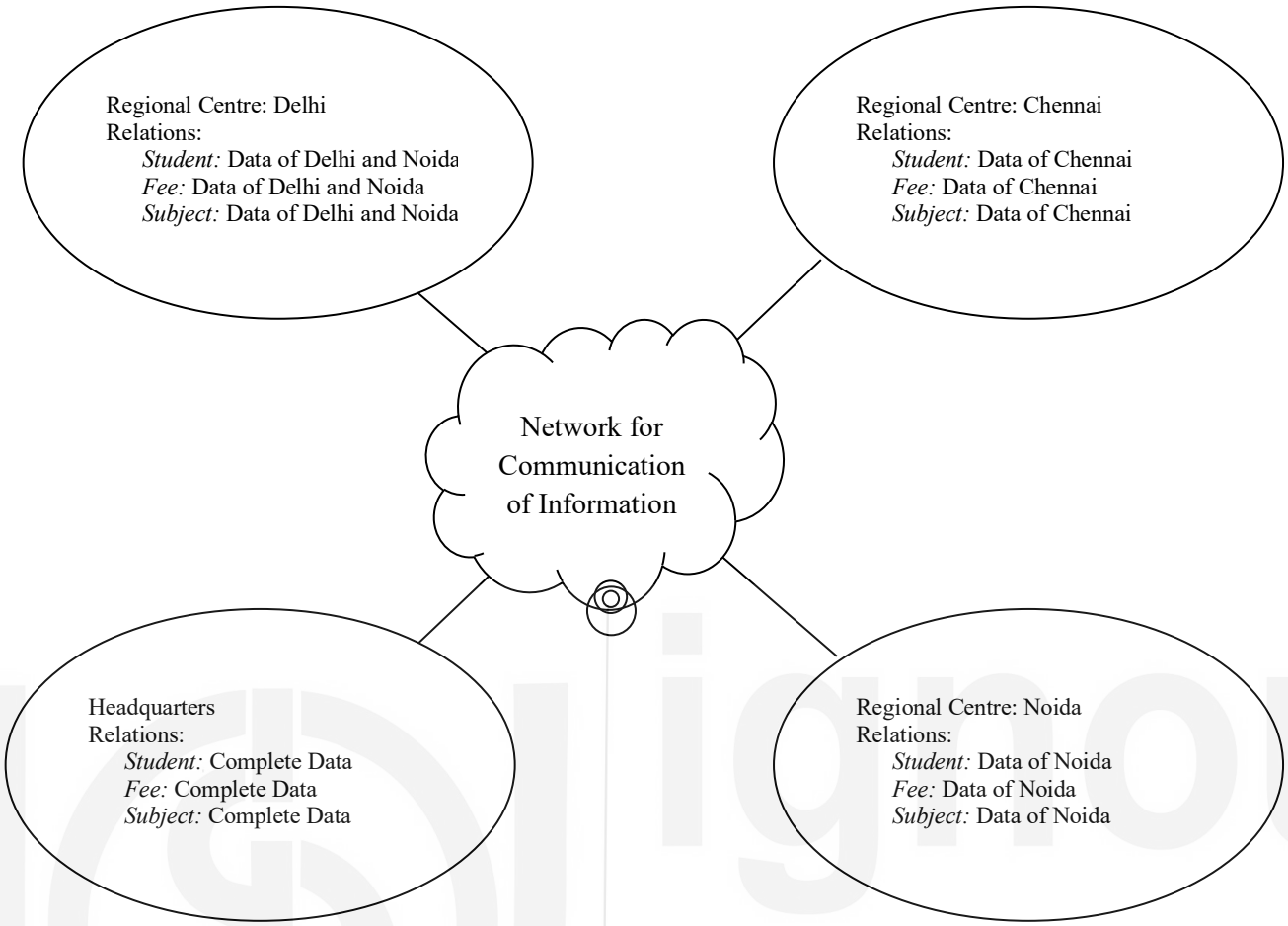

Figure 2: A distributed student relation

The following are some of the important aspects of distributed database systems:

- A distributed database has multiple sites that hold a part of a logically connected database. For example, in Figure 2, the Regional Centre Delhi holds the data of the students who are registered with that regional office of the head office.
- The technical implementation details, such as "The data of Delhi students is available at the Headquarters and Delhi Regional Center", are hidden from the actual users of the database, who can be the students or Regional Centre staff, etc. This is known as transparency.
- The user can issue a command to the DDBMS without worrying about the location of data (called location transparency), where the copies of the data have been kept (called replication transparency), or the fragment of the horizontal fragment of data is kept (called sharding). For example, in Figure 2, a student who is associated with Regional Centre Noida can place his/her query without knowing the fact that his/her data is located at the Headquarters, Regional Centre Delhi and Regional Centre Noida. His/her query will be replied to by any of the locations based on the place of the query and the status of different sites. You may also observe that in Figure 2, the data of students related to Regional Cenre Noida is replicated at three sites – Headquarters, Regional Centre Delhi and Regional Centre Noida. Also, note that horizontal fragments of student

data of Chennai are stored at the Regional Centre Chennai site.

- A DDBMS will be required to manage more failures than a centralised DBMS, as it must also manage network failures.
- A DDBMS is available for a longer duration than that of centralised RDBMS.
- A DDBMS is better scalable than a centralised RDBMS due to data distribution on various sites.

16.2.1 Data Fragmentation and Replication

In a distributed database, as shown in Figure 2, all the data is not stored at all the database sites. In general, the data related to a particular site is stored on that site. For example, in Figure 2, data related to Regional Centre Chennai and Regional Centre Noida is stored at their respective sites. The process of distributing data into different parts is called fragmentation. This kind of distribution of data facilitates faster query processing, as most of the queries at a site can be answered from the local data. In addition to fragmentation, the data is replicated at more than one site. For example, in Figure 2, data of Regional Centre Noida is replicated at Regional Centre Noida and Regional Centre Delhi sites. This data replication helps improve the reliability and availability of the database, as the database would be available to users even if one of the replicated sites fails. The following example explains different kinds of fragmentation.

Example: Consider the slightly modified Student table given in Figure 1 with the following database instance:

Student						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23001	Anil	Mohan	UG	9900100000	Noida	PGDCA
23002	Rahim	Jamil	UG	9900200000	Chennai	PGDCA
23003	Simon	Robert	PG	9900300000	Noida	MCA
23004	Sahil	Sanjay	UG	9900400000	Delhi	MCA
23005	Sanjay	Ajay	PG	9900300000	Noida	PGDCA
23006	Diya	Jeba	UG	9900400000	Chennai	MCA

The following can be the Horizontal Fragments of the table based on the RC.

Student: Horizontal Fragment on RC Noida site						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23001	Anil	Mohan	UG	9900100000	Noida	PGDCA
23003	Simon	Robert	PG	9900300000	Noida	MCA
23005	Sanjay	Ajay	PG	9900300000	Noida	PGDCA

Student: Horizontal Fragment on RC Chennai site						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23002	Rahim	Jamil	UG	9900200000	Chennai	PGDCA
23006	Diya	Jeba	UG	9900400000	Chennai	MCA

Student: Horizontal Fragment on RC Delhi site (Noida or Delhi)						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23001	Anil	Mohan	UG	9900100000	Noida	PGDCA
23003	Simon	Robert	PG	9900300000	Noida	MCA
23004	Sahil	Sanjay	UG	9900400000	Delhi	MCA
23005	Sanjay	Ajay	PG	9900300000	Noida	PGDCA

The other kind of fragmentation is vertical fragmentation. For example, if RC Noida has been assigned the work of contacting all the students telephonically, irrespective of their regional centre, then one possible vertical fragment for Regional Centre Noida would be as follows:

Student: Vertical Fragmentation for Noida			
EnrNo	Name	Phone	Programme
23001	Anil	9900100000	PGDCA
23002	Rahim	9900200000	PGDCA
23003	Simon	9900300000	MCA
23004	Sahil	9900400000	MCA
23005	Sanjay	9900300000	PGDCA
23006	Diya	9900400000	MCA

Further, considering that Regional Centre Noida is assigned PGDCA students only for telephonic contact, the fragment would be a mixed fragment as follows:

Student: Mixed Fragmentation for Noida			
EnrNo	Name	Phone	Programme
23001	Anil	9900100000	PGDCA
23002	Rahim	9900200000	PGDCA
23005	Sanjay	9900300000	PGDCA

As far as replication is concerned, you may observe that the student data of Regional Centre Noida is replicated at Regional Centre Delhi, too. Replication helps in enhancing reliability and availability but results in more overheads in transaction processing.

16.2.2 Distributed Query Processing

A query in a distributed database management system is submitted at a site. This query is then converted to a relational algebraic query and optimised using local and global query optimisation processes. Local query optimisation is the same as that of a centralised DBMS; however, global query optimisation involves the selection of sites for query evaluation, cost of data communication, and cost of query processing. The process of distributed query processing is explained with the help of the following example.

Example: Consider a query submitted at the Headquarters seeking to find the Percentage of fee share of Regional Centre Noida in the financial year 2022-23. This query would require computing the fee collected by RC Noida to the total fee collected between the dates 01st April 2022 and 31st March 2023. This query may consist of two subqueries:

(a) Finding the total fee collected for the financial year 2022-23.
(b) Finding the total fee collected at RC Noida in the financial year 2022-23.

In addition, you may assume that the financial records are ordered chronologically.

One of the possible ways of processing the queries would be to process both the sub-queries at the Headquarters. If both these sub-queries can be processed during the same query processing cycle, then it may be a good choice. However, if both the sub-queries are processed separately, then other options may be explored. What if subquery (a) is processed at Headquarters and subquery (b) is processed at the RC Noida site? This will require a transfer of results from subquery (b) from the RC Noida site to the headquarters site, where the result will be displayed to the user who made the query.

A detailed discussion on distributed database management systems is beyond the scope of this Unit. You may refer to the further readings for more details on this topic.

16.3 ACTIVE DATABASES

Active databases, as the name suggests, comprise dynamic actions on the occurrence of certain events. Such actions were part of the SQL 99 standard and are called triggers. Let us define the model that can be used for an active database:

Active Database Model

Consider the Student and Result relations given in Figure 3.

Student				
Enrollment No.	Name	Programme	Cumulative Grade Point Average	Status

Result		
Enrollment No.	CourseCode	Grade

Figure 3: An example

Assuming that the Cumulative Grade Point Average is to be updated for each student when related data is created in the Result relation. The following events will cause a database action to be activated:

Action Triggering Event: In the database of Figure 3, on updating the Result table, the Cumulative Grade Point Average (CGPA) of a Student needs to be updated, as well. Assuming that once a Record is entered in the Result table, then it cannot be deleted, and only the Grade attribute can be modified in the Result relation, the following may be the events that may trigger the action of an update on CGPA for the Student relation:

- Addition of a tuple in the Result relation
- Modification of a Grade in the Result Relation.

Once the trigger event occurs, the next step is to check the condition, if any. In the case of the Student database given in Figure 3, the referential integrity constraints on the Enrollment number and CourseCode (each student will register for a set of courses) will make sure that only the valid entry is made in the Result table. In addition, the check constraint on Grade ensures that a valid Grade letter is entered in the Result table. However, an interesting observation here is that the state of a student must be "Active" in case his/her CGPA is to be updated. This can fit as a condition for activating the trigger.

Issues relating to Active databases: Some of the issues relating to active databases relate primarily to the process of creating and maintaining the triggers. Such databases must include a set of rules or commands that should be able to deactivate or drop the old or redundant rules, as there can be a very large number of rules which may be changed during the lifetime of the database. Next, since triggers can be activated either before, during or after a condition, a huge number of events may be scheduled, causing the database to act slowly as a response to normal record addition and deletion processes. Interestingly, it adds another issue, considering that addition of results by a teacher in the student's database triggered the update of about 50 students' GPA. However, the teacher realised that s/he had used an incorrect file to update student records and deleted his/her result. This will lead to the firing of a very large number of triggers that would be required to undo the changes in the GPA of the students.

A detailed discussion on active databases is beyond the scope of this Unit. You may refer to further readings for more details.

☞ Check Your Progress 1

1) How is DDBMS different from RDBMS?

……

……

2) Suggest the data fragmentation of the student table if all RCs need only the enrolment number and name of all the PGDCA students.

……

……

……

3) What is an active database? What is a triggering event?

……

……

……

16.4 XML AND DATA REPRESENTATION

The eXtensible Markup Language (XML) is one of the popular data representation languages. It uses user-defined tags to represent a document. A typical XML document relating to the student's table, as shown in Figure 3, is shown below:

```
<school>
        <class>
                <class_no>XII</class_no>
                <class_teacher>John</class_teacher>
                <student>
                        <enrolmentNo>2301002297</enrolmentNo>
                        <name>Ritesh Jain</name>
                        <programme>PGDCA</programme>
                        <cgpa>7.5</cgpa>
                        <status>Pass</status>
                        <result>
                                <coursecode>BCS011</coursecode>
                                <grade>A</grade>
                                <coursecode>BCS013</coursecode>
                                <grade>B+</grade>
                        </result>

                </student>
                <student>
                        <enrolmentNo>2301002301</enrolmentNo>
                        <name>Amitesh</name>
                        <programme>PGDCA</programme>
                        <cgpa>8.5</cgpa>
                        <status>Distinction</status>
                        <result>
                                <coursecode>BCS011</coursecode>
                                <grade>A+</grade>
                                <coursecode>BCS013</coursecode>
```

```
                    <grade>A</grade>
                </result>
            </student>
        </class>
</school>
```

Figure 4: A sample XML document

You may please observe that instead of using a separate table, in the XML document, the results of the students are merged along with the student information. Thus, XML representation has the potential to store all the information about an entity in one place. Such a representation, though it may be useful for searching from a point of view as no join operation is required, may lead to redundancy of information. In addition to the use of tags, XML allows users to store attributes along with a tag. For example, the enrolment number can be stored as an attribute of the student as:

```
<student enrolmentNo = "2301002301">
        <name> …
</student>
…
```

The attribute may be useful for searching for information on the related field.

Just like the database management system has a different schema, XML also can be used to validate the structure of the XML data. Figure 5 is a possible document type definition (DTD) that would validate the document given in Figure 4.

```
<!DOCTYPE  school [
<!ELEMENT school (class+)>
<!ELEMENT class (class_no, class_teacher, student+)>
<!ELEMENT student (enrolmentNo, name, programme, cgpa, status, result *)>
<!ELEMENT result ( (coursecode, grade)+)>
<!ELEMENT class_no( #PCDATA )>
<!ELEMENT class_teacher( #PCDATA )>
<!ELEMENT enrolmentno( #PCDATA )>
<!ELEMENT name ( #PCDATA )>
<!ELEMENT programme ( #PCDATA )>
<!ELEMENT cgpa ( #PCDATA )>
<!ELEMENT status ( #PCDATA )>
<!ELEMENT coursecode ( #PCDATA )>
<!ELEMENT grade ( #PCDATA )>
]>
```

Figure 5: Document Type Definition for XML of Figure 4

Please note the following points in the DTD, as shown in Figure 5:

- A school consists of data from one or more classes.
- Each class stores the class number and class teacher's name.
- A class have one or more students.

- For each student, you store the enrolment number, name, programme, his/her CGPA and present status (Distinction, Merit, Pass, Unsuccessful).
- In addition, the result of each student is stored. This result can be in zero or more subjects.
- The filed type #PCDATA means parsed character data, which stores the text parsed by the parser into the fields.

XML has become a popular format for data exchange and storage. Several tools have been developed to verify, display as tree nodes and query the XML documents. A detailed discussion of these tools is beyond the scope of this Unit. However, we present a few basic features of XQuery, which is a standard query language for XML documents.

A XQuery expression uses five basic keywords for querying. These are *for*, *let, where*, *order by* and *return*. The *for* clause selects a sequence of nodes in a document, *let* is used to bind a variable name to a sequence, *where* is used for the selection of nodes, *order by* is used for giving an order to the sequence, and *return* is used to specify what values are to be returned.

For example, the query:

```
for $x in /school/class/student
where $x/cgpa > 8
return </name>
```

This query will return the name of the student with a CGPA of 8 or more. You may refer to the further readings for more details on XQuery.

16.5 BLOCKCHAIN DATABASES

Conceptually, blockchain is a paradigm of storage of data in a distributed manner, which may also protect data from fraudulent transactions and updates. Blockchain technology was used to store distributed ledgers and bitcoins. However, this technology is not limited to only these applications. In this section, we will present some of the basic features of blockchain with the help of an example. However, to understand the principles of blockchain, you should study the paper "Bitcoin: A Peer-to-Peer Electronic Cash System" by Satoshi Nakamoto in 2008.

Let us discuss the components of the blockchain technology. A Blockchain will have the following components:

1. Distributed set of Authorised Nodes: The role of a node is to keep the ledger of data. A ledger consists of data logs, which are timestamped. A *blockchain* is a sequence of *blocks* of data, which is maintained at each of these authorised nodes.
2. A *block* contains:
    a. Block Number: It is a unique sequence number given to every

block on the blockchain.

b. Nonce: It is a random number that is used only once in a block.
c. Transaction logs: A sequence of transaction logs that are to be recorded in a block.
d. Computed Hash value: A cryptographic hash function is applied to the content of a block to compute a hash value of the block. This hash value is also stored in the block.
e. Hash value of Previous Block.

3. A blockchain is constructed by linking the blocks. In addition, each block contains the hash value of previous block, which is used to ensure data security.

A hypothetical blockchain is shown in Figure 6.

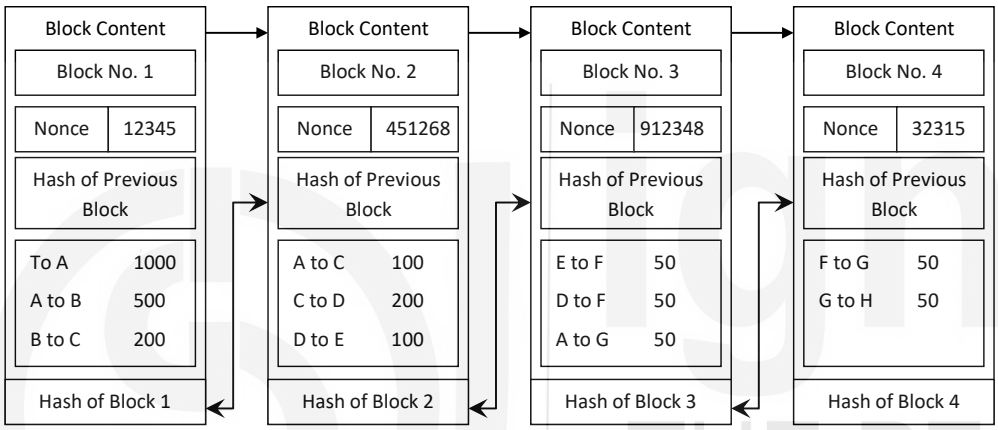

Figure 6: A Hypothetical Blockchain Data

The blockchain processes involve the following:

1. A network consensus protocol, which ensures security, trust, and concurrence amongst the network of nodes of the blockchain.

2. The hashing process ensures the integrity of the transaction ledgers; in other words, it ensures that the transaction ledgers are not tampered with. In Figure 6, each transfer of money may represent one entry of the transaction ledger.

3. Uses a digital signature to certify consent for the transaction.

Let us discuss some of these concepts in more detail.

Cryptographic Hash Function: At the most elementary level, a blockchain consists of a cryptographic Hash function. This hash function is a kind of secure fingerprint of a Block of data of blockchain. One of the popular hash algorithms is Secure Hash Function 256 (SHA256), which produces a 256-bit long hash value for the content of a block irrespective of the size of the block (which can vary from one character to millions of characters). The hash function has the following properties:

- It maps the block contents of any size to a hash value called Hash of a fixed size; for example, SHA 256 maps block content of any size to 256 bits Hash.
- For a given content, the hash function will always produce the same Hash. In addition, it is expected that if the contents of two blocks are different even by a single character, then the hash function should produce different Hash values for the blocks.
- In case you change the content of a block slightly, then the value of the Hash function changes significantly. In addition, if you know a part of the hash value, you cannot predict the rest of the hash value.
- The cryptographic hash function is also called one-way encryption, as you can use the block contents to create the Hash value, but you cannot use the Hash value to find the block contents. Interestingly, many password-based authentication systems store the password using one-way encryption. This is why your system administrator simply provides you a link to change the password, but not the password itself.

Please also note that in case there is a change in the content of a block, its hash value will also change. This feature of the hash value may be very useful in finding the tampering of data of a block, which is discussed next.

How to identify data tampering or corruption in a Blockchain?

As discussed earlier, any minor change of any value in a block at a particular node will result in change in its computed hash value. This will cause change in the stored *hash value of previous block* in all the subsequent blocks. For example, consider in Block No. 2 of Figure 6, the transaction log of A to C is modified to 200 from the present value of 100. This will result in change in the computed hash value of Block no. 2. Further, this will result in change in *hash value of previous block* of Block 3, which will result in change in the computed hash values of Block no 3. A similar change will occur in Block 4, too. Thus, the previously stored hash values of Block 2, Block 3 and Block 4 will not match the computed new hash values of these blocks.

But how will it be recognised that the blocks have different hash values? Here, the consensus protocol will play its role. Since each blockchain block is stored on all the blockchain network nodes, changes in one site can easily be recognised, as the other nodes with the stored copy of the blockchain will not agree with this blockchain, thus identifying the tampering.

Use of Nonce: Nonce is part of the block content and is a short form of random Number used once. This number is mined using an algorithm such that the Hash value generated for the block is unique.

Validity of Transactions: You may observe that in Figure 6, most of the transactions show the transfer of money from one account to another. For example, in Block 1, there is a transaction - A to C 500. But does A have that much amount? Well, that can be established from the first transaction in this block, which says "To A 1000". Thus establishing that A has sufficient funds for

the transfer. Such transactions are sometimes called Coinbase transactions and can be used to ascertain the validity of transactions.

Use of Digital Signature: The digital signatures are used to ensure that the transactions are performed by the authorised person. For example, consider a new transaction, "A to E 50", to be added to Block 4; how will it be ensured that this transaction has been performed by A? For this, A needs to have a pair of private and public keys; let us call them PrivateKeyA and PublicKeyB. Both these keys can be very long in length. A must maintain the PrivateKeyA as a secret key, whereas he can share the PublicKeyA with other stakeholders. One key characteristic of this private-public key pair is that if you know the PublicKeyA, you cannot derive the PrivatekeyA. A can create this transaction "A to E 50" by using the PublickeyA and PublickeyE as:

"PublickeyA toPublickeyE 50"

Next, A needs to sign this message using the PrivatekeyA.

This transaction can be verified by using the PublickeyA, which ensures that the transaction has been created by an authorised person only. Please note here that in Figure 6, we have shown transactions using alphabets like A, B, C, etc., but in the actual blockchain, they will be public keys.

You may refer to the further readings for more details on Blockchain technology.

☞ **Check Your Progress 2**

1) Create an XML document consisting of Marks of two students in at least one subject. Also, make the DTD for validating this XML document.

……

……

2) What is the role of nonce in a blockchain?

……

……

3) Consider that in Figure 6, in Block 3, the E to F transaction is modified to 100. What changes would it cause in the blockchain?

……

……

16.6 MULTIMEDIA DATABASE

Multimedia data is an integration of textual, graphical, audio, video, and animation data. In general, multimedia data may include lengthy textual documents, pictures, drawings, digital audio clips, movies, and animations. A multimedia database should be able to store large multimedia data efficiently and provide the feature of querying the multimedia data.

Querying is a very interesting domain in the context of multimedia data, as most searches in such data require retrieval of data based on some content. For example, you may be interested in all the videos related to "Database Integrity and Normalization" from a multimedia database or videos of a particular presenter. Please note that such queries would require indexing on the objects and related contents.

How can you create these indexes? One way to create indexes would be to create a large amount of metadata for each object. This metadata should use standardised keywords, title, credentials of creators, summary information, etc. However, in many situations, the metadata would not be available, leading to the generation and verification of the metadata. The second type of indexes are those that can be created using automatic analysis.

Characteristics of Multimedia Data:

Let us now discuss some of the basic characteristics of different types of multimedia data.

Textual Data: In general, the textual data consists of articles, which include paragraphs and headings. This data is stored using ASCII or Unicode standards. Further, you may define certain keywords on these articles that can be used for searching.

Image Data: An image is stored in digital form using picture elements called *pixels*. The size of the picture depends on the number of colours used. For example, a black and white picture would just require 1 bit for every pixel, whereas a true coloured picture may require 24 bits (8 bits each for three basic colours) to store one element. Further, the resolution of a picture is represented using pixels per inch. Therefore, the size of a good-quality picture is quite substantial. You can use different kinds of compression standards to reduce the size of a picture. Some of the common image files include formats like GIF, PNG, JPEG, etc. A search on images may be to find the images related to the same objects. To answer such a query, every image can be divided into segments that have similar characteristics. Further, this segment characteristics information can be grouped to identify certain basic characteristics of an image. Various images are compared based on the similarity of characteristics of those images. In addition, a number of labels, indexes, etc., can also be linked to the segments of images, helping in finding meaningful information about the images. You may have noticed that present image recognition software are able to link many images together based on their characteristics.

Video and Animation Data: Video and animation data may be considered as a large sequence of frames that are displayed in real-time to form a live movie. In general, video data may be divided into video segments, which may be related to a group of objects or activities. For example, an eLearning video may be divided into segments, with each segment involving a sequence of learning concepts. Some of the popular compression formats used for video are MPEG, AVI, etc. A query of video data may be related to identifying the portion of a video related to a specific learning objective.

Audio data: Audio data is recorded audio messages. These messages may be identified for similarity of voice. A query on audio data may try to group the audio of a person based on recognition of voice.

Issues and Characteristics of Analysis of Multimedia Data

- Multimedia data must be stored, labelled, and indexed so you can define some similarity measures on that data.
- Statistics can be used to define the characteristics of image data using the values of colour, texture, etc.
- Further, the object's shape can be one feature of object recognition. This may

include the identification of facial features.

- In the present time, the recognition of an object in an image, video or animation is a major challenge. The key barriers to object recognition are:
    - The angle from which an image has been taken may vary the shape of an object
    - The scale and size of the picture may affect the recognition of objects.
    - Rotated, transformed, and occluded images are difficult to recognise, as it is difficult to keep a database of all transformations of an image.
- An important development in the area of object recognition is the development of a scale-invariant feature transform (SIFT) by David Lowe. SIFT extracts features from images such that they are not affected by rotation and scaling. You may refer to this algorithm in further readings.
- Another important concept relating to the recognition of images is the tagging of images. Tagging is useful in searching for images. You must have observed on certain social media that you may be tagged in some images that include you. At present, the tagging algorithms are being enhanced for better accuracy. In general, these algorithms use machine learning and statistics to analyse the image content with already tagged image libraries.
- Digital audio is difficult to index and retrieve, as it does not have any typical features or characteristics that can be used to identify the content specifically.
- A multimedia database manages different data formats and, thus, requires many storage and Input/Output technologies. These technologies may include technologies for text and images, like scanners, digital cameras, printers, etc.; technologies for audio, video, and animation, like microphones, video cameras, DVDs, Musical Instrument Digital Interface (MIDI), good quality displays and speakers etc.

Multimedia data is part of many digital systems, such as Patient health monitoring data, geographical data, students' data, etc.; therefore, effective multimedia data management systems are required to handle such vast data.

16.7 USE OF DATABASES IN WEB APPLICATIONS

A database system is a persistent collection of an organisation's data, which is shared and integrated among various applications. Database technology supports non-redundant storage of data, which allows the following features:

- Structured storage of data in the form of tables
- Secure data insertion, modification, and deletion.
- Support for concurrent database transactions.
- Easy but controlled access to data.

These features are very useful for any web application, too. Therefore, many web applications use database management systems (DBMS) to store data and access it securely for display on the web. These DBMSs are normally managed on a separate server, called a database server, and communicate with the web server to store or retrieve data. The web server then communicates this information through the relevant web pages to communicate with the web clients.

For example, when you register on an eCommerce website, the information you fill in the registration form is stored in a registration database. This registration information is accessed when you log in again on that website. On the eCommerce website, you may

also search for product information, put some items in the shopping cart, order the items from the cart, and so on. All these activities are supported by several databases. However, these activities are obscure, as you just interact with the eCommerce website through a browser interface. This section focuses on how the database can be accessed by a web server; however, our approach will be to discuss the process rather than the use of a scripting language for coding.

Some of the key characteristics that should be supported by the web server–database server interconnection are:

- The information exchange between the two servers must be secure.
- The two servers should have pools of connection for transfer of information.
- Concurrent read and write operations should be supported by the database server.
- The response time for an operation should be as low as possible.

In general, these servers may either be supporting a 2-tier or a 3-tier client-server architecture.

Steps for Creating a Web Database Application

The first requirement for creating a web application is that your web server has the required drivers to connect to the DBMS of interest. You are required to perform the following steps to create a web application that uses a database as a backend.

Connecting Web Server to a Database Server

On receiving a request from a web application client, a web application may request data from a database server. In general, a web server is required to establish a connection with the database server if it has not already done so. You require the following parameters for a connection:

- A database driver for the DBMS of interest.
- Access credentials on the DBMS and the database (e.g. username and password).
- The URL of the database server and its port number.

For example, assume that you are developing a web application using Java Server Pages (JSP) and you want to connect to a MySQL database. Further, your username is *student* and password *dbms*. In addition, the MySQL host is a local host at the URL and port number - *jdbc:mysql://localhost* and 3306, respectively. Further, assume that the database which is to be accessed is named: *class_schedule* and the table of the database is *class (teacherName, dateofclass, timeofclass, classroom),* which have been created by you using SQL command using the username *student.* The following command would be needed by you to create an active connection:

```
Connection connection = DriverManager.getConnection
        ("jdbc:mysql://localhost:3306/class_schedule", "student", "dbms");
```

You may have to perform several other commands, too. You may refer to further readings for more details.

Creating a form for data input and inserting data into the database

You may need to create a form using HTML or any scripting language and use GET or POST methods to transfer the data to the web server, which may execute the commands for inserting data into the database (only after establishing a connection with the database).

For example, the following commands will help you insert the data into the database:

```
String sqlinsert = "insert into class_schedule values (?, ?, ?, ?)";
PreparedStatement preparedstat = connection.prepareStatement (sqlinsert);
   preparedstat.setString(1, teacherName);
   preparedstat.setString(2, dateofclass);
   preparedstat.setString(3, timeofclass;
   preparedstat.setString(4, classroom);
```

```
prep.executeUpdate();
        preparedstat.close();
```

The commands shown above are just to demonstrate the programming. Please note that the variable name *connection* is the same as the variable name used while establishing the connection. The insert statement consists of four '?', filled up by the data obtained from the HTML form using the setString functions.

Accessing data from the database and displaying it as a webpage
For this purpose, you may create a query to access the database (after establishing a connection). The data obtained from the query may be put in a variable, which can be accessed record by record or complete data at a time to create the required webpage.

For example, you may display the content of all the classes using the following set of commands. This command may result in a number of rows of information being created by the output System.out.println function call.

```
String sqlselect = "select * from class_schedule ";
Statement statement = connection.createStatement();
ResultSet results=statement.executeQuery (sqlselect);
while(results.next())
        System.out.println(results.getString(1)+"  "+rs.getString(2)+"  "+
                rs.getString(3)+"  "+rs.getString(4));
```

Many more statements are required to implement a web database. You may refer to the documentation of the web tools, languages and databases you use to implement a web application. While using the connection, the term jdbc was used. Let us discuss such terms in more detail.

Open Database Connectivity (ODBC) and JAVA Database Connectivity (JDBC)

There are many commercial DBMSs like DB2 (IBM), Oracle and MySQL (Oracle), SQL Server, MS-Access (Microsoft), PostgreSQL (Open Source) etc. Each DBMS has its own interface for programming (called Application Programming Interface or, in short, API) and data storage and indexing mechanisms. ODBC is a standard interface (API) that allows you to connect to a database of any DBMS and manipulate data using SQL commands. If you are using JAVA to write the interface functions to a DBMS, then you require JDBC. These interfaces provide a set of drivers that allow you to connect and access data from the database.

Closing of database connection: In general, it is recommended to close the connection after the required database operation has been performed.

☞ Check Your Progress 3

1) What are the characteristics of multimedia data?

……

……

2) List four application areas of multimedia databases.

……

……

……

3) What are the steps required to access a database from a web application?

……

……

……

4. Why is there a need for ODBC? Why do you need JDBC?

……..……..……

……..……..……

……

16.6 SUMMARY

This Unit introduced you to some of the advanced technologies. The unit discusses the concepts relating to the distributed database management systems (DDBMS). A DDBMS replicates the fragmented data on various database sites. This unit explains the concepts of horizontal and vertical fragmentation of data. A query in distributed database may be submitted at any site of the database. This unit introduces how a distributed query will be processed. The unit also explains the triggering events and dynamic actions in the context of active database with the help of an example. XML is one of the popular ways of representing semi-structure data in a very simple format. This unit explained the data representation and validation in XML. The unit also presented a brief introduction to blockchain technology. The focus in the unit was to introduce you to basic concepts with the help of an example. Finally, the unit presents a brief introduction to multimedia database and web databases.

16.7 SOLUTIONS / ANSWERS

Check Your Progress 1

1) A DDBMS allows transparency to a user about the location of the data, which may be distributed among various sites of the database. DDBMS makes sure that data is made available to the user (if she has the access rights to use the data) even if one site is down. This is possible because DDBMS keeps track of the data replications. In addition, DDBMS is responsible for concurrent transaction management in distributed settings. The RDBMS, on the other hand, does not distribute data and, therefore, is less complex than DDBMS.

2) In such a case, you may only require the following fragment:

```
SELECT EnrNo, Name
FROM Student
WHERE Programme = 'PGDCA'
```

3) An active database is a database that has a set of active actions that are to be performed upon the occurrence of an event. Some of these triggering events can be the addition or deletion of a record or modification of a record.

Check Your Progress 2

1) The following is the XML document:

```
<studentMarks>
    <student>
        <rollno>12345</rollno>
        <name>Aman</name>
        <marksinsubject>
            <subject>Mathematics</subject>
            <marks>75</marks>
            <subject>Physics</subject>
            <marks>85</marks>
        </marksinsubject>
    </student>
    <student>
        <rollno>54321</rollno>
        <name>Arvin</name>
        <marksinsubject>
            <subject>Mathematics</subject>
            <marks>65</marks>
        </marksinsubject>
    </student>
</studentMarks>
```

The DTD would be as follows:

```
<!DOCTYPE  studentMarks [
<!ELEMENT (student (rollno, name, marksinsubject+))+>
<!ELEMENT marksinsubject ( (subject, marks)+)>
<!ELEMENT rollno ( #PCDATA )>
<!ELEMENT name ( #PCDATA )>
<!ELEMENT subject ( #PCDATA )>
<!ELEMENT marks ( #PCDATA )>
]>
```

2) The nonce means the number used once. It is a random number appended to the blockchain content to ensure that no two blocks have the same hash value.

3) The change in the Block 3 E to F transaction to 100 will change the hash value of Block 3; this will no longer match the hash value of the previous block stored in Block 4. Further, the consensus protocol will not accept values from this block for any purpose.

Check Your Progress 3

1) The following are the basic characteristics of multimedia data:
Textual Data: includes long strings such as textual articles, can include multilingual UNICODE formats, and requires a keyword-based search.
Image Data: large data requiring compression, use formats like GIF, PNG, JPEG, etc. images can be segmented and tagged for the purpose of similarity
Video and Animation Data: has a large sequence of frames, video segments may be created, and popular compression formats are MPEG, AVI, etc.
Audio data: may be identified for similarity of voice.

2) Medical databases
Bioinformatics

3) The following steps are required to create a web database:

- You should have a valid database with proper records on a database server.
- Create a connection string using the URL and port number of the database server. This sting should also include a valid username and password.
- Open the connection, query the database using the parameters submitted by the user, and generate the results.
- Format and display the results on the client, which may be a browser window.

4. ODBC helps in accessing the contents of the database using SQL commands, simplifying the programming of the databases. JDBC is an application programming interface that allows Java programmers to access any database.