SOUTHERN LEYTE STATE UNIVERSITY BONTOC

San Ramon, Bontoc, Southern Leyte, 6604 Philippines
Telefax No.: (053) 382-3121; E-mail: slsubc@yahoo.com
Website: www.slsuonline.edu.ph

ADVANCED DATABASE

LEARNING GUIDE 2020

REXAL S. TOLEDO

SOUTHERN LEYTE
STATE UNIVERSITY-BONTOC

Southern Leyte State University

Vision

A high quality corporate University of Science, Technology and Innovation.

Mission

SLSU will:

a) Develop Science, Technology, and Innovation leaders and professionals;
b) Produce high-impact technologies from research and innovations;
c) Contribute to sustainable development through responsive community engagement programs;
d) Generate revenues to be self-sufficient and financially-viable.

Quality Policy

We at Southern Leyte State University commit enthusiastically to satisfy our stakeholders' needs and expectations by adhering to good governance, relevance and innovations of our instruction, research and development, extension and other support services and to continually improve the effectiveness of our Quality Management System in compliance to ethical standards and applicable statutory, regulatory, industry and stakeholders' requirements.

The management commits to establish, maintain and monitor our quality management system and ensure that adequate resources are available.

COURSE OVERVIEW

Course No.	IT301/IT301L
Course Code	
Descriptive Title	Advanced Database Systems
Credit Units	2 units (Lecture) / 1 unit (Laboratory)
School Year/Term	2020-2021 / 1st semester
Mode of Delivery	
Name of Instructor	Rexal S. Toledo
Course Description	This course covers modern database and information system as well as research issues in the field. It will cover selected topics on NoSQL, object-oriented, active, deductive, spatial, temporal and multimedia databases. The course includes advanced issues of object-oriented database, XML, advanced client server architecture, Information Retrieval and Web Search and distributed database techniques.

Course Outcomes

1. Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.
2. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.
3. Apply software development fundamentals to produce computing-based solutions.
4. Communicate effectively in a variety of professional contexts.

TABLE OF CONTENTS

MODULE

2

Object and Object-Relational

LESSON

1 Object Database Concepts and Conceptual Design

2 Object Database Extensions to SQL

3 The ODMG Object Model and the Object Definition Language ODL

4 The Object Query Language OQL

PRE-TEST

MODULE 2:

1. What are the origins of the object-oriented approach?
2. What primary characteristics should an OID possess?
3. Discuss the various type constructors. How are they used to create complex object structures?
4. Discuss the concept of encapsulation, and tell how it is used to create abstract data types.
5. Explain what the following terms mean in object-oriented database terminology: method, signature, message, collection, extent.
6. What is the relationship between a type and its subtype in a type hierarchy?
7. What is the constraint that is enforced on extents corresponding to types in the type hierarchy?
8. What is the difference between persistent and transient objects? How persistence handled in typical OO database systems?
9. How do regular inheritance, multiple inheritance, and selective inheritance differ?
10. Discuss the concept of polymorphism/operator overloading.
11. Discuss how each of the following features is realized in SQL 2008: object identifier, type inheritance, encapsulation of operations, and complex object structures.
12. In the traditional relational model, creating a table defined both the table type (schema or attributes) and the table itself (extension or set of current tuples). How can these two concepts be separated in SQL 2008?
13. Describe the rules of inheritance in SQL 2008.
14. What are the differences and similarities between objects and literals in the ODMG object model?
15. List the basic operations of the following built-in interfaces of the
16. ODMG object model: Object, Collection, Iterator, Set, List, Bag, Array, and
17. Dictionary.
18. Describe the built-in structured literals of the ODMG object model and the operations of each.
19. What are the differences and similarities of attribute and relationship properties of a user-defined (atomic) class?
20. What the differences and similarities of class inheritance are via extends and interface inheritance via ":" in the ODMG object model?
21. Why are the concepts of extents and keys important in database applications?
22. Describe the following OQL concepts: database entry points, path expressions, iterator variables, named queries (views), aggregate functions, grouping, and quantifiers.
23. What is meant by the type orthogonality of OQL?
24. What are the main differences between designing a relational database and an object database?
25. Describe the steps of the algorithm for object database design by EER-to OO mapping.

LESSON 5

Object Database Concepts and Conceptual Design

LEARNING OUTCOMES

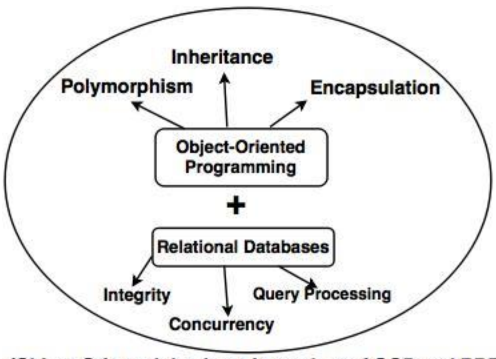
After studying this lesson, you should be able to:

1. Learn how the types for complex-structured objects are specified via type constructors
2. Discuss encapsulation and persistence and presents inheritance concepts
3. Discuss some additional OO concepts, and gives a summary of all the OO concepts and discuss how object database (ODB) design differs from relational database (RDB) design

Introduction to Object based databases

Object oriented database systems are alternative to relational database and other database systems. In object oriented database, information is represented in the form of objects.

Object oriented databases are exactly same as object oriented programming languages. If we can combine the features of relational model (transaction, concurrency, recovery) to object oriented databases, the resultant model is called as object oriented database model.



(Object-Oriented database is product of OOP and RDB)

Object-Oriented database

Object has two components:

1. State (value)
2. Behavior (operations)

Instance variables (attributes) – Hold values that define internal state of object.

Operation is defined in two parts, Signature (interface) and implementation (method).

Inheritance

Permits specification of new types or classes that inherit much of their structure and/or operations from previously defined types or classes.

Operator overloading

- Operation's ability to be applied to different types of objects
- Operation name may refer to several distinct implementations

Object Identity, and Objects versus Literals

Object has unique identity, Implemented via a unique, system-generated object identifier (OID), Immutable.

Most OO database systems allow for the representation of both objects and literals (simple or complex values).

Complex Type Structures for Objects and Literals

Structure of arbitrary complexity – Contain all necessary information that describes object or literal

Nesting type constructors generate complex type from other types.

Type constructors (type generators):
- Atom (basic data type – int, string, etc.)
- Struct (or tuple)
- Collection

Collection types:
- Set
- Bag
- List
- Array
- Dictionary

Object definition language (ODL)

Used to define object types for a particular database application.

Example:
Specifying the object types EMPLOYEE, DATE, and DEPARTMENT using type constructors.

```
define type EMPLOYEE
    tuple (  Fname:          string;
             Minit :         char;
             Lname:          string;
             Ssn:            string;
             Birth_date:     DATE;
             Address:        string;
             Sex:            char;
             Salary:         float;
             Supervisor:     EMPLOYEE;
             Dept:           DEPARTMENT;
define type DATE
    tuple (  Year:           integer;
             Month:          integer;
             Day:            integer; );
define type DEPARTMENT
    tuple (  Dname:          string;
             Dnumber:        integer;
             Mgr:            tuple (  Manager:      EMPLOYEE;
                                      Start_date:   DATE; );
             Locations:      set(string);
             Employees:      set(EMPLOYEE);
             Projects:       set(PROJECT); );
```

Example:
Adding operations to the definitions of EMPLOYEE and DEPARTMENT.

```
define class EMPLOYEE
    type tuple (  Fname:          string;
                  Minit:          char;
                  Lname:          string;
                  Ssn:            string;
                  Birth_date:     DATE;
                  Address:        string;
                  Sex:            char;
                  Salary:         float;
                  Supervisor:     EMPLOYEE;
                  Dept:           DEPARTMENT; );
    operations    age:            integer;
                  create_emp:     EMPLOYEE;
                  destroy_emp:    boolean;
end EMPLOYEE;
define class DEPARTMENT
    type tuple (  Dname:          string;
                  Dnumber:        integer;
                  Mgr:            tuple (  Manager:      EMPLOYEE;
                                           Start_date:   DATE; );
                  Locations:      set (string);
                  Employees:      set (EMPLOYEE);
                  Projects        set(PROJECT); );
    operations    no_of_emps:     integer;
                  create_dept:    DEPARTMENT;
                  destroy_dept:   boolean;
                  assign_emp(e: EMPLOYEE): boolean;
                  (* adds an employee to the department *)
                  remove_emp(e: EMPLOYEE): boolean;
                  (* removes an employee from the department *)
end DEPARTMENT;
```

Encapsulation of Operations

Encapsulation

- Related to abstract data types.
- Define behavior of a class of object based on operations that can be externally applied.
- External users only aware of interface of the operations.
- Can divide structure of object into visible and hidden attributes.

Constructor operation
- Used to create a new object

Destructor operation
- Used to destroy (delete) an object

Modifier operations
- Modify the state of an object

Retrieve operation
- Dot notation to apply operations to object

Persistence of Objects

Transient objects
- Exist in executing program
- Disappear once program terminates

Persistent objects
- Stored in database, persist after program termination
- **Naming mechanism**: object assigned a unique name in object base, user finds object by its name
- **Reachability**: object referenced from other persistent objects, object located through references

Example:
Creating persistent objects by naming and reachability.

```
define class DEPARTMENT_SET
    type set (DEPARTMENT);
    operations add_dept(d: DEPARTMENT):   boolean;
        (* adds a department to the DEPARTMENT_SET object *)
            remove_dept(d: DEPARTMENT):  boolean;
        (* removes a department from the DEPARTMENT_SET object *)
            create_dept_set:        DEPARTMENT_SET;
            destroy_dept_set:       boolean;
end Department_Set;
...
persistent name ALL_DEPARTMENTS: DEPARTMENT_SET;
(* ALL_DEPARTMENTS is a persistent named object of type DEPARTMENT_SET *)
...
d:= create_dept;
(* create a new DEPARTMENT object in the variable d *)
...
b:= ALL_DEPARTMENTS.add_dept(d);
(* make d persistent by adding it to the persistent set ALL_DEPARTMENTS *)
```

Type (Class) Hierarchies and Inheritance

Inheritance – Definition of new types based on other predefined types. Leads to type (or class) hierarchy.

Type: type name and list of visible (public) functions (attributes or operations).

Format: `TYPE_NAME: function, function, ..., function`

Subtype

- Useful when creating a new type that is similar but not identical to an already defined type.
- Subtype inherits functions.
- Additional (local or specific) functions in subtype.

Example:

```
EMPLOYEE subtype-of PERSON: Salary, Hire_date, Seniority
STUDENT subtype-of PERSON: Major, Gpa
```

Extent

- A named persistent object to hold collection of all persistent objects for a class.

Persistent collection

- Stored permanently in the database

Transient collection

- Exists temporarily during the execution of a program (e.g. **query result**).

Other Object-Oriented Concepts

Polymorphism of operations

- Also known as operator overloading
- Allows same operator name or symbol to be bound to two or more different implementations
- Type of objects determines which operator is applied

Multiple inheritance

Subtype inherits functions (attributes and operations) of more than one supertype. Summary of Object Database Concepts or the main concepts used in ODBs and object-relational systems:

- Object identity
- Type constructors (type generators)
- Encapsulation of operations
- Programming language compatibility
- Type (class) hierarchies and inheritance
- Extents
- Polymorphism and operator overloading

Object Database Conceptual Design

Differences between Conceptual Design of ODB and RDB

One of the main differences between ODB and RDB design is how relationships are handled.

RDBMS stands for Relational Database Management System. It is a database management system based on the relational model i.e. the data and relationships are represented by a collection of inter-related tables. It is a DBMS that enables the user to create, update, administer and interact with a relational database. RDBMS is the basis for SQL, and for all modern database systems like: **MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.**

OODBMS stands for Object-Oriented Database Management System. It is a DBMS where data is represented in the form of objects, as used in object-oriented programming. OODB implements object-oriented concepts such as classes of objects, object identity, polymorphism, encapsulation, and inheritance. An object-oriented database stores complex data as compared to relational database.

Some examples of OODBMS are: **Versant Object Database, Objectivity/DB, ObjectStore, Caché and ZODB.**

BASIS	RDBMS	OODBMS
Long Form	Stands for Relational Database Management System.	Stands for Object Orientedl Database Management System.
Way of storing data	Stores data in Entities, defined as tables hold specific information.	Stores data as Objects.
Data Complexity	Handles comparitively simpler data.	Handles larger and complex data than RDBMS.
Grouping	Entity type refers to the collection of entity that share a common definition.	Class describes a group of objects that have common relationships, behaviors, and also have similar properties.
Data Handeling	RDBMS stores only data.	Stores data as well as methods to use it.
Main Objective	Data Independece from application program.	Data Encapsulation.

BASIS	RDBMS	OODBMS
Key	A Primary key distinctively identifies an object in a table..	An object identifier (OID) is an unambiguous, long-term name for any type of object or entity.

Mapping an EER Schema to an ODB Schema

- Create ODL class for each EER entity type
- Add relationship properties for each binary relationship
- Include appropriate operations for each class
- ODL class that corresponds to a subclass in the EER schema
    - Inherits type and methods of its superclass in ODL schema
- Weak entity types
    - Mapped same as regular entity types
- Categories (union types)
    - Difficult to map to ODL
- An n-ary relationship with degree n > 2
    - Map into a separate class, with appropriate references to each participating class

The Object Query Language OQL

- Query language proposed for ODMG object model.
- Simple OQL queries, database entry points, and iterator variables.
  Syntax:

```
select ... from ... where ... structure
```

Entry point: named persistent object
Iterator variable: define whenever a collection is referenced in an OQL query

Query Results and Path Expressions

Result of a query

Any type that can be expressed in ODMG object model

OQL orthogonal with respect to specifying path expressions

- Attributes, relationships, and operation names (methods) can be used interchangeably within the path expressions

Other Features of OQL

Named query

- Specify identifier of named query

OQL query will return collection as its result

- If user requires that a query only return a single element use element operator

Aggregate operators
Membership and quantification over a collection
Special operations for ordered collections
Group by clause in OQL

- Similar to the corresponding clause in SQL
- Provides explicit reference to the collection of objects within each group or partition

Having clause

- Used to filter partitioned sets

LESSON 6

Object Database Extensions to SQL

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Learn SQL Object-Relational Features.

SQL Object-Relational Features

The object-relational extensions to SQL include row types, collection types, user defined types (UDT), typed tables and reference types. The object-relational data model does not support all the features of the object-oriented data model. Instead it represents an evolutionary step from the relational data model to the object-oriented data model. A row type allows a table to have attributes that are not atomic. This violates the first normal form constraint (1NF) and in general the object-relational data model does not require tables to be in 1NF. The following table includes a row attribute called name and a row attribute called address. Name contains fields for the first name and last name. Address contains fields for street, city, state and zip.

```
Create table Faculty (fid varchar (10) primary key,
name row (first varchar (10), last varchar (30)), address row
(street varchar (30), city varchar (30), state
char (2), zip char(10))
```

The fields of a row type can be accessed by path expressions. For example if f is an alias for the Faculty table in a query, the path expression f.name.first could be used to refer to the first name of a Faculty member.

Object-Relational Features: Object DB Extensions to SQL

- **Type constructors (generators)**
  Specify complex types using UDT
- **Mechanism for specifying object identity**
- **Encapsulation of operations**

Provided through user-defined types (UDTs)

- **Inheritance mechanisms**
  Provided using keyword UNDER

User-Defined Types (UDTs) and Complex Structures for Objects

- **UDT syntax:**
  - CREATE TYPE <type name> AS (<component declarations>);
  - Can be used to create a complex type for an attribute (similar to *struct* – no operations)
  - Or: can be used to create a type as a basis for a table of objects (similar to *class* – can have operations)
- Array type – to specify collections
  - Reference array elements using []
- **CARDINALITY** function
  - Return the current number of elements in an array
- Early SQL had only array for collections
  - Later versions of SQL added other collection types (set, list, bag, array, etc.)

- **Reference type**
  - Create unique object identifiers (OIDs)
  - Can specify system-generated object identifiers
  - Alternatively can use primary key as OID as in traditional relational model
  - Examples:
    - REF IS SYSTEM GENERATED
    - REF IS <OID_ATTRIBUTE><VALUE_GENERATION_METHOD> ;

Creating Tables Based on the UDTs

- **INSTANTIABLE**
  - Specify that UDT is instantiable
  - The user can then create one or more tables based on the UDT
  - If keyword INSTANTIABLE is left out, can use UDT only as attribute data type – not as a basis for a table of objects

Encapsulation of Operations

- User-defined type
  - Specify methods (or operations) in addition to the attributes
  - **Format**:

```
CREATE TYPE <TYPE-NAME> (
<LIST OF COMPONENT ATTRIBUTES AND THEIR TYPES>
<DECLARATION OF FUNCTIONS (METHODS)>
);
```

Illustrating some of the object features of SQL. Using UDTs as types for attributes such as Address and Phone.

```
(a) CREATE TYPE STREET_ADDR_TYPE AS (
        NUMBER          VARCHAR (5),
        STREET          NAME VARCHAR (25),
        APT_NO          VARCHAR (5),
        SUITE_NO        VARCHAR (5)
    );
    CREATE TYPE USA_ADDR_TYPE AS (
        STREET_ADDR     STREET_ADDR_TYPE,
        CITY            VARCHAR (25),
        ZIP             VARCHAR (10)
    );
    CREATE TYPE USA_PHONE_TYPE AS (
        PHONE_TYPE      VARCHAR (5),
        AREA_CODE       CHAR (3),
        PHONE_NUM       CHAR (7)
    );
```

Illustrating some of the object features of SQL. Specifying UDT for PERSON_TYPE.

```
(b) CREATE TYPE PERSON_TYPE AS (
        NAME            VARCHAR (35),
        SEX             CHAR,
        BIRTH_DATE      DATE,
        PHONES          USA_PHONE_TYPE ARRAY [4],
        ADDR            USA_ADDR_TYPE
    INSTANTIABLE
    NOT FINAL
    REF IS SYSTEM GENERATED
    INSTANCE METHOD AGE() RETURNS INTEGER;
    CREATE INSTANCE METHOD AGE() RETURNS INTEGER
        FOR PERSON_TYPE
        BEGIN
            RETURN /* CODE TO CALCULATE A PERSON'S AGE FROM
                      TODAY'S DATE AND SELF.BIRTH_DATE */
        END;
    );
```

Specifying Type Inheritance

- **NOT FINAL:**
    - The keyword NOT FINAL indicates that subtypes can be created for that type
- **UNDER**
    - The keyword UNDER is used to create a subtype

Illustrating some of the object features of SQL. Specifying UDTs for STUDENT_TYPE and EMPLOYEE_TYPE as two subtypes of PERSON_TYPE.

```
(c) CREATE TYPE GRADE_TYPE AS (
        COURSENO      CHAR (8),
        SEMESTER      VARCHAR (8),
        YEAR          CHAR (4),
        GRADE         CHAR
    );
    CREATE TYPE STUDENT_TYPE UNDER PERSON_TYPE AS (
        MAJOR_CODE    CHAR (4),
        STUDENT_ID    CHAR (12),
        DEGREE        VARCHAR (5),
        TRANSCRIPT    GRADE_TYPE ARRAY [100]


INSTANTIABLE
NOT FINAL
INSTANCE METHOD GPA( ) RETURNS FLOAT;
CREATE INSTANCE METHOD GPA( ) RETURNS FLOAT
    FOR STUDENT_TYPE
    BEGIN
        RETURN /* CODE TO CALCULATE A STUDENT'S GPA FROM
                  SELF.TRANSCRIPT */
    END;
);
CREATE TYPE EMPLOYEE_TYPE UNDER PERSON_TYPE AS (
    JOB_CODE      CHAR (4),
    SALARY        FLOAT,
    SSN           CHAR (11)
INSTANTIABLE
NOT FINAL
);
CREATE TYPE MANAGER_TYPE UNDER EMPLOYEE_TYPE AS (
     DEPT_MANAGED CHAR (20)
INSTANTIABLE
);
```

Specifying Type Inheritance

- **Type inheritance rules:**
    - All attributes/operations are inherited
    - Order of supertypes in UNDER clause determines inheritance hierarchy
    - Instance (object) of a subtype can be used in every context in which a supertype instance used
    - Subtype can redefine any function defined in supertype

Creating Tables based on UDT

- **UDT must be INSTANTIABLE**
- **One or more tables can be created**
- **Table inheritance:**
    - UNDER keyword can also be used to specify supertable/subtable inheritance

- o Objects in subtable must be a subset of the objects in the supertable

Illustrating some of the object features of SQL. Creating tables based on some of the UDTs, and illustrating table inheritance.

```
(d) CREATE TABLE PERSON OF PERSON_TYPE
        REF IS PERSON_ID SYSTEM GENERATED;
    CREATE TABLE EMPLOYEE OF EMPLOYEE_TYPE
        UNDER PERSON;
    CREATE TABLE MANAGER OF MANAGER_TYPE
        UNDER EMPLOYEE;
    CREATE TABLE STUDENT OF STUDENT_TYPE
        UNDER PERSON;
```

Specifying Relationships via Reference

- **Component attribute of one tuple may be a reference to a tuple of another table**
  - o Specified using keyword REF
- **Keyword SCOPE**
  - o Specify name of table whose tuples referenced
- **Dot notation**
  - o Build path expressions
- **–>**
  - o Used for dereferencing

Illustrating some of the object features of SQL. Specifying relationships using REF and SCOPE.

```
(e) CREATE TYPE COMPANY_TYPE AS (
        COMP_NAME      VARCHAR (20),
        LOCATION       VARCHAR (20));
    CREATE TYPE EMPLOYMENT_TYPE AS (
        Employee REF (EMPLOYEE_TYPE) SCOPE (EMPLOYEE),
        Company REF (COMPANY_TYPE) SCOPE (COMPANY) );
    CREATE TABLE COMPANY OF COMPANY_TYPE (
        REF IS COMP_ID SYSTEM GENERATED,
        PRIMARY KEY (COMP_NAME) );
    CREATE TABLE EMPLOYMENT OF EMPLOYMENT_TYPE;
```

LESSON 7

The ODMG Object Model and the Object Definition Language ODL

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Discuss how to design an ODB from an EER conceptual schema.
2. Learn how to use ODL, OQL.
3. Discuss how object types for atomic objects can be constructed.

ODMG Object Model and Object Definition Language ODL

ODMG object model

- Data model for object definition language (ODL) and object query language (OQL)

Objects and Literals

- Basic building blocks of the object model. The main difference between the two is that an object has both an object identifier and a state (or current value), whereas a literal has a value (state) but no object identifier.

Object has five aspects:

- **Object identifier –** unique system-wide identifier (or Object_id).
- **name –** this name can be used to locate the object, and the system should return the object given that name.
- **lifetime –** The lifetime of an object specifies whether it is a persistent object (that is, a database object) or transient object (that is, an object in an executing program that disappears after the program terminates).
- **structure –** specifies how the object is constructed by using the type constructors. The structure specifies whether an object is atomic or not.
- **creation -** refers to the manner in which an object can be created.

Literal - a value that does not have an object identifier.

There are three types of literals:

1. **Atomic literals**
2. **Structured literals**
3. **Collection literals**

The notation of ODMG uses three concepts: interface, literal, and class. Behavior refers to operations. State refers to properties (attributes).

Interface – Specifies only behavior of an object type. Typically noninstantiable.
Class – specifies both state (attributes) and behavior (operations) of an object type and is instantiable.

Inheritance in the Object Model of ODMG

Behavior inheritance – Also known as ISA or interface inheritance. Specified by the colon (:) notation.

EXTENDS inheritance

- Specified by keyword extends
- Inherit both state and behavior strictly among classes
- Multiple inheritance via extends not permitted

Built-in Interfaces and Classes in the Object Model

Collection objects

- Inherit the basic Collection interface

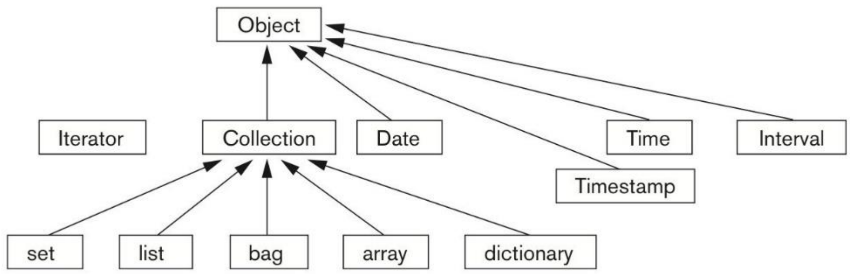
```
I = O.create_iterator()
```

- Creates an iterator object for the collection
- To loop over each object in a collection

Collection objects further specialized into:

- Set
- List
- bag
- array
- dictionary

Inheritance hierarchy for the built-in interfaces of the object model.



Atomic (User-Defined) Objects

- Specified using the keyword class in ODL.

In the object model, any user-defined object that is not a collection object is called an atomic object.

An **attribute** is a property that describes some aspect of an object. Attributes have values (which are typically literals having a simple or complex structure) that are stored within the object. However, attribute values can also be Object_ids of other objects. Attribute values can even be specified via methods that are used to calculate the attribute value.

A **relationship** is a property that specifies that two objects in the database are related. In the object model of ODMG, only binary relationships are explicitly represented, and each binary relationship is represented by a pair of inverse references specified via the keyword relationship.

Each object type can have a number of **operation signatures**, which specify the operation name, its argument types, and its returned value, if applicable. Operation names are unique within each object type, but they can be overloaded by having the same operation name appear in distinct object types. The operation signature can also specify the names of exceptions that can occur during operation execution.

The attributes, relationships, and operations in a class definition.

```
class EMPLOYEE
(     extent            ALL_EMPLOYEES
      key               Ssn   )
{
      attribute         string              Name;
      attribute         string              Ssn;
      attribute         date Birth_date;
      attribute         enum Gender{M, F}   Sex;
      attribute         short               Age;
      relationship      DEPARTMENT          Works_for
                            inverse DEPARTMENT::Has_emps;
      void              reassign_emp(in string New_dname)
                            raises(dname_not_valid);
};
```

```
class DEPARTMENT
(   extent          ALL_DEPARTMENTS
    key             Dname, Dnumber )
{
    attribute       string              Dname;
    attribute       short               Dnumber;
    attribute       struct Dept_mgr {EMPLOYEE Manager, date Start_date}
                        Mgr;
    attribute       set<string>         Locations;
    attribute       struct Projs {string Proj_name, time Weekly_hours)
                        Projs;
    relationship    set<EMPLOYEE>       Has_emps inverse EMPLOYEE::Works_for;
    void            add_emp(in string New_ename) raises(ename_not_valid);
    void            change_manager(in string New_mgr_name; in date
                        Start_date);
};
```

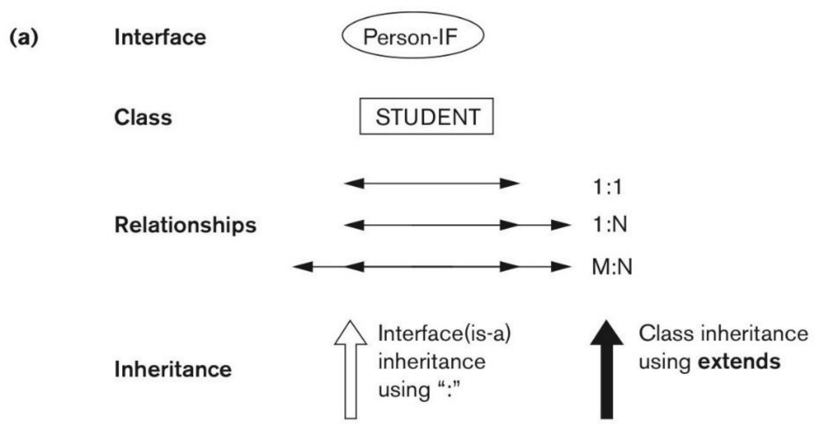
Extents, Keys, and Factory Objects

Extent – A persistent named collection object that contains all persistent objects of class.
Key – One or more properties whose values are unique for each object in extent of a class.
Factory object – Used to generate or create individual objects via its operations.

Object Definition Language ODL

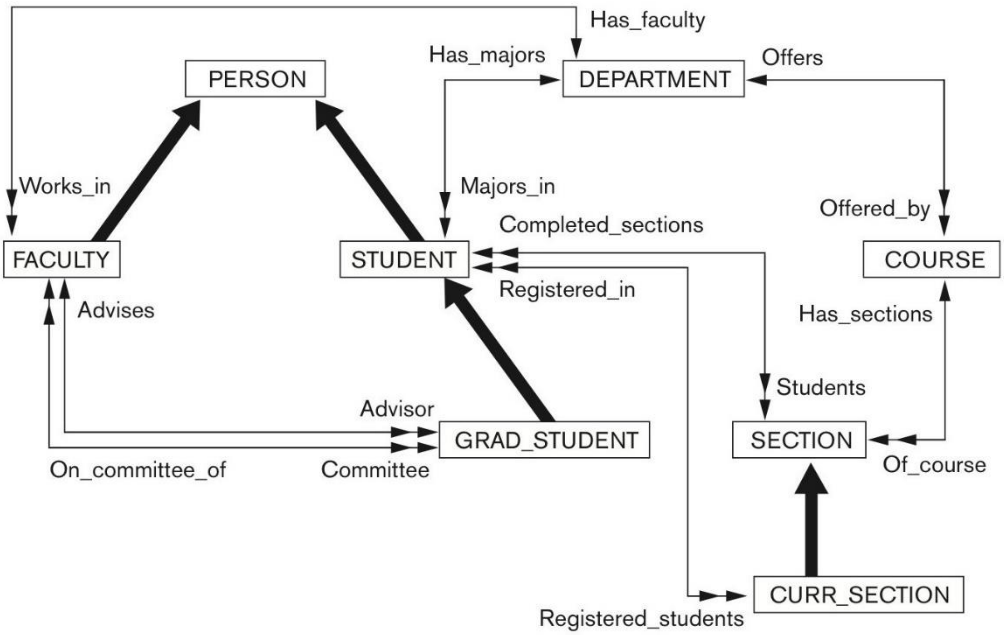
The **ODL** is designed to support the semantic constructs of the ODMG object model and is independent of any particular programming language. Its main use is to create object specifications—that is, classes and interfaces. Hence, ODL is not a programming language. A user can specify a database schema in ODL independently of any programming language, and then use the specific language bindings to specify how ODL constructs can be mapped to constructs in specific programming languages, such as C++, Smalltalk, and Java.

An example of a database schema. Graphical notation for representing ODL schemas.



An example of a database schema. A graphical object database schema for part of the UNIVERSITY database (GRADE and DEGREE classes are not shown).

(b)



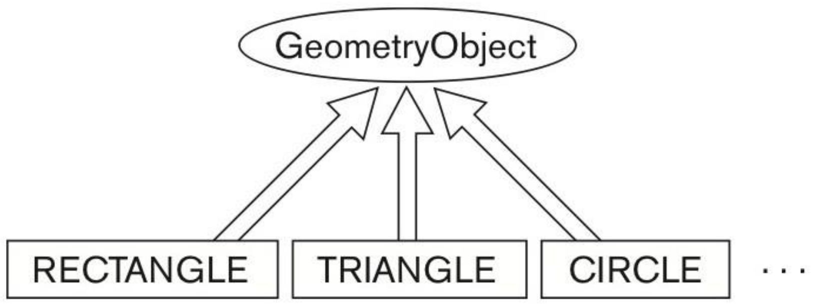
Possible ODL schema for the UNIVERSITY database.

```
class PERSON
(   extent          PERSONS
    key             Ssn )
{   attribute       struct Pname {   string   Fname,
                                     string   Mname,
                                     string   Lname }        Name;
    attribute       string                                   Ssn;
    attribute       date                                     Birth_date;
    attribute       enum Gender{M, F}                        Sex;
    attribute       struct Address {  short    No,
                                      string   Street,
                                      short    Apt_no,
                                      string   City,
                                      string   State,
                                      short    Zip }         Address;
    short           Age();   };
class FACULTY extends PERSON
(   extent          FACULTY )
{   attribute       string          Rank;
    attribute       float           Salary;
    attribute       string          Office;
    attribute       string          Phone;
    relationship    DEPARTMENT      Works_in inverse DEPARTMENT::Has faculty;
    relationship    set<GRAD_STUDENT> Advises inverse GRAD_STUDENT::Advisor;
    relationship    set<GRAD_STUDENT> On_committee_of inverse GRAD_STUDENT::Committee;
    void            give_raise(in float raise);
    void            promote(in string new rank); };
class GRADE
(   extent GRADES )
{
    attribute       enum GradeValues{A,B,C,D,F,I, P} Grade;
    relationship    SECTION Section inverse SECTION::Students;
    relationship STUDENT Student inverse STUDENT::Completed_sections; };
class STUDENT extends PERSON
(   extent          STUDENTS )
{   attribute       string          Class;
    attribute       Department      Minors_in;
    relationship    Department Majors_in inverse DEPARTMENT::Has_majors;
    relationship    set<GRADE> Completed_sections inverse GRADE::Student;
    relationship    set<CURR_SECTION> Registered_in INVERSE CURR_SECTION::Registered_students;
    void            change_major(in string dname) raises(dname_not_valid);
    float           gpa();
    void            register(in short secno) raises(section_not_valid);
    void            assign_grade(in short secno; IN GradeValue grade)
                        raises(section_not_valid,grade_not_valid); };
```

```
class DEGREE
{   attribute     string          College;
    attribute     string          Degree;
    attribute     string          Year; };
class GRAD_STUDENT extends STUDENT
(   extent        GRAD_STUDENTS )
{   attribute     set<Degree>     Degrees;
    relationship  Faculty advisor inverse FACULTY::Advises;
    relationship  set<FACULTY>    Committee inverse FACULTY::On_committee_of;
    void          assign_advisor(in string Lname; in string Fname)
                      raises(faculty_not_valid);
    void          assign_committee_member(in string Lname; in string Fname)
                      raises(faculty_not_valid); };
class DEPARTMENT
(   extent        DEPARTMENTS
    key           Dname )
{   attribute     string          Dname;
    attribute     string          Dphone;
    attribute     string          Doffice;
    attribute     string          College;
    attribute     FACULTY         Chair;
    relationship  set<FACULTY> Has_faculty inverse FACULTY::Works_in;
    relationship  set<STUDENT> Has_majors inverse STUDENT::Majors_in;
    relationship  set<COURSE> Offers inverse COURSE::Offered_by; };
class COURSE
(   extent        COURSES
    key           Cno )
{   attribute     string          Cname;
    attribute     string          Cno;
    attribute     string          Description;
    relationship  set<SECTION> Has_sections inverse SECTION::Of_course;
    relationship  <DEPARTMENT> Offered_by inverse DEPARTMENT::Offers; };
class SECTION
(   extent        SECTIONS )
{   attribute     short           Sec_no;
    attribute     string          Year;
    attribute     enum Quarter{Fall, Winter, Spring, Summer}
                      Qtr;
    relationship  set<Grade> Students inverse Grade::Section;
    relationship  COURSE Of_course inverse COURSE::Has_sections; };
class CURR_SECTION extends SECTION
(   extent        CURRENT_SECTIONS )
{   relationship  set<STUDENT> Registered_students
                      inverse STUDENT::Registered_in
    void          register_student(in string Ssn)
                      raises(student_not_valid, section_full); };
```

An illustration of interface inheritance via ":". (a) Graphical schema representation, (b) Corresponding interface and class definitions in ODL.

(a)



(b) interface GeometryObject

```
{   attribute   enum      Shape{RECTANGLE, TRIANGLE, CIRCLE, … }
                                 Shape;
    attribute   struct    Point {short x, short y} Reference_point;
    float       perimeter();
    float       area();
    void        translate(in short x_translation; in short y_translation);
    void        rotate(in float angle_of_rotation); };
class RECTANGLE : GeometryObject
(   extent      RECTANGLES )
{   attribute   struct    Point {short x, short y} Reference_point;
    attribute   short     Length;
    attribute   short     Height;
    attribute   float     Orientation_angle; };
class TRIANGLE : GeometryObject
(   extent      TRIANGLES )
{   attribute   struct    Point {short x, short y} Reference_point;
    attribute   short     Side_1;
    attribute   short     Side_2;
    attribute   float     Side1_side2_angle;
    attribute   float     Side1_orientation_angle; };
class CIRCLE : GeometryObject
(   extent      CIRCLES )
{   attribute   struct    Point {short x, short y} Reference_point;
    attribute   short     Radius; };
…
```

POST-TEST

MODULE 2:

1. What are the origins of the object-oriented approach?
2. What primary characteristics should an OID possess?
3. Discuss the various type constructors. How are they used to create complex object structures?
4. Discuss the concept of encapsulation, and tell how it is used to create abstract data types.
5. Explain what the following terms mean in object-oriented database terminology: method, signature, message, collection, extent.
6. What is the relationship between a type and its subtype in a type hierarchy?
7. What is the constraint that is enforced on extents corresponding to types in the type hierarchy?
8. What is the difference between persistent and transient objects? How persistence handled in typical OO database systems?
9. How do regular inheritance, multiple inheritance, and selective inheritance differ?
10. Discuss the concept of polymorphism/operator overloading.
11. Discuss how each of the following features is realized in SQL 2008: object identifier, type inheritance, encapsulation of operations, and complex object structures.
12. In the traditional relational model, creating a table defined both the table type (schema or attributes) and the table itself (extension or set of current tuples). How can these two concepts be separated in SQL 2008?
13. Describe the rules of inheritance in SQL 2008.
14. What are the differences and similarities between objects and literals in the ODMG object model?
15. List the basic operations of the following built-in interfaces of the
16. ODMG object model: Object, Collection, Iterator, Set, List, Bag, Array, and
17. Dictionary.
18. Describe the built-in structured literals of the ODMG object model and the operations of each.
19. What are the differences and similarities of attribute and relationship properties of a user-defined (atomic) class?
20. What the differences and similarities of class inheritance are via extends and interface inheritance via ":" in the ODMG object model?
21. Why are the concepts of extents and keys important in database applications?
22. Describe the following OQL concepts: database entry points, path expressions, iterator variables, named queries (views), aggregate functions, grouping, and quantifiers.
23. What is meant by the type orthogonality of OQL?
24. What are the main differences between designing a relational database and an object database?
25. Describe the steps of the algorithm for object database design by EER-to OO mapping.

REFERE NCES

1. Database Systems: Design, Implementation, and Management
2. 4th Edition, Peter Rob & Carlos Coronel
3. Database Systems: Design, Implementation, and Management, Fifth Edition, Rob and Coronel
4. Database System Concepts Seventh Edition, Avi Silberschatz ET. Al.
5. Fundamentals of Database Systems Seventh Edition, by Ramez Elmasri and Shamkant B. Navathe.

Internet:

6. https://www.geeksforgeeks.org/