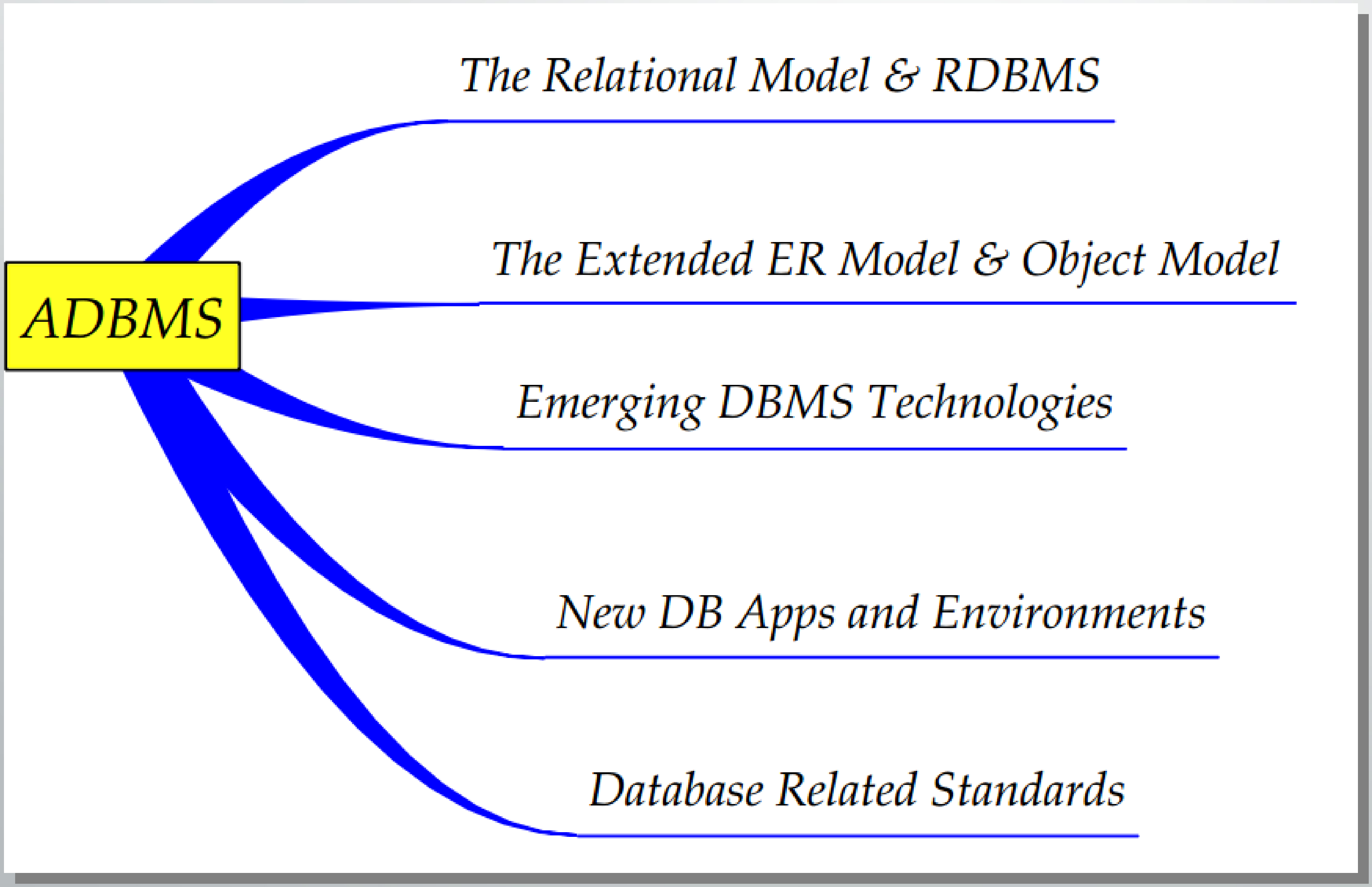
Advanced
Database Management System

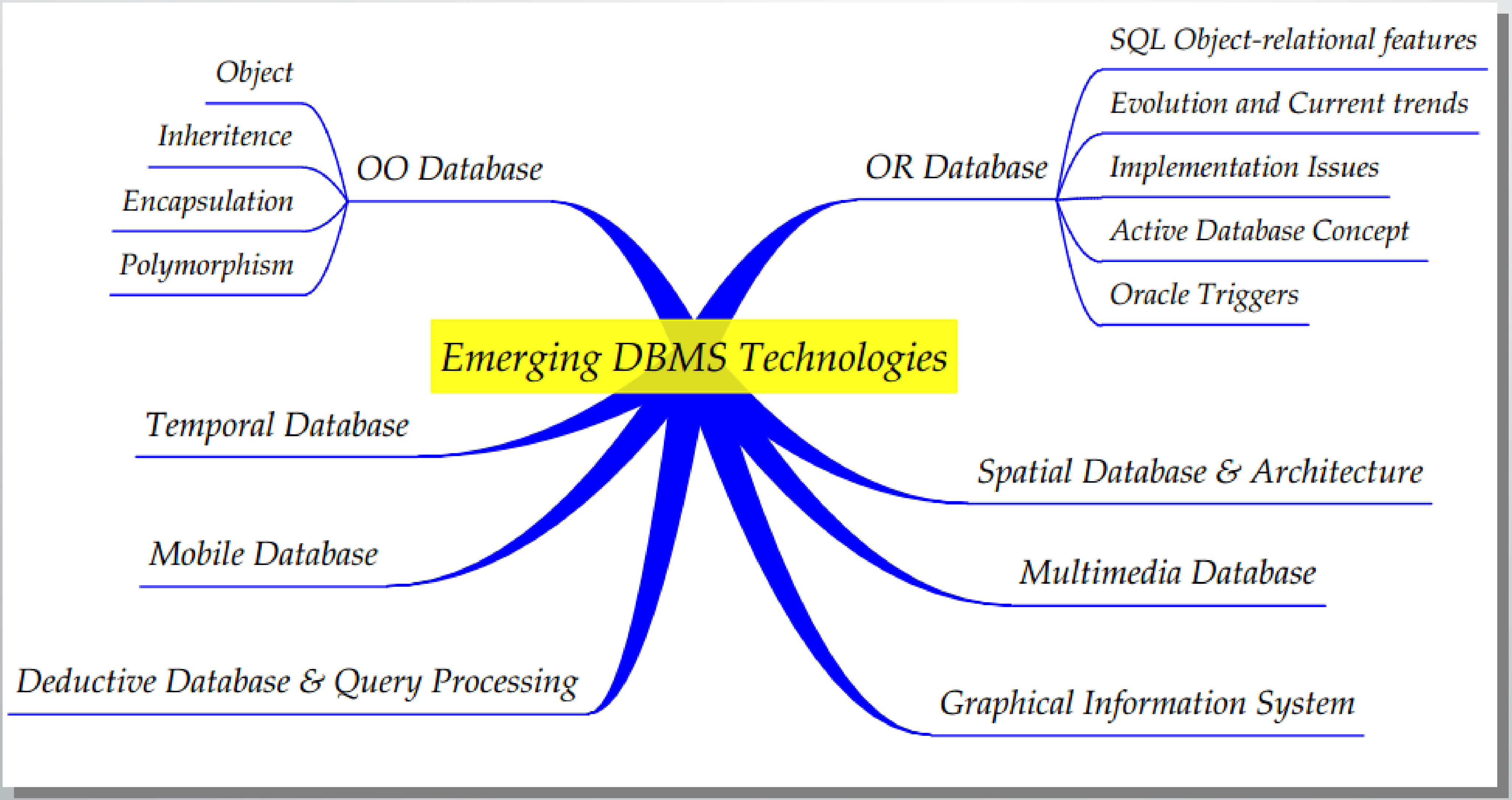
[BSc CSIT-7th Semester]

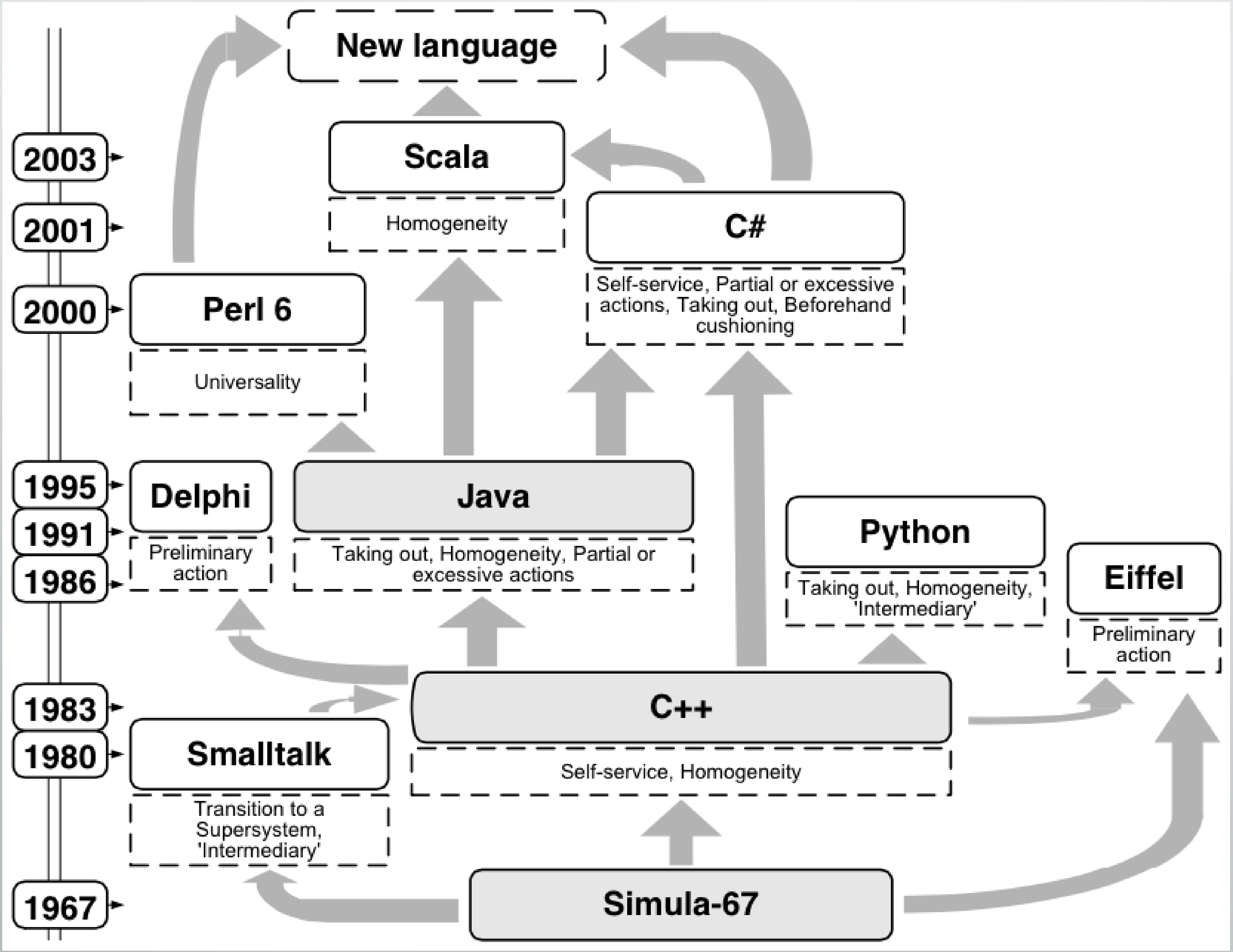
Rishi K. Marseni
{rishimarseni@gmail.com}

Course Content – Abstract View


ADBMS
The Relational Model & RDBMS
The Extended ER Model & Object Model
Emerging DBMS Technologies
New DB Apps and Environments
Database Related Standards

Emerging DBMS Technologies

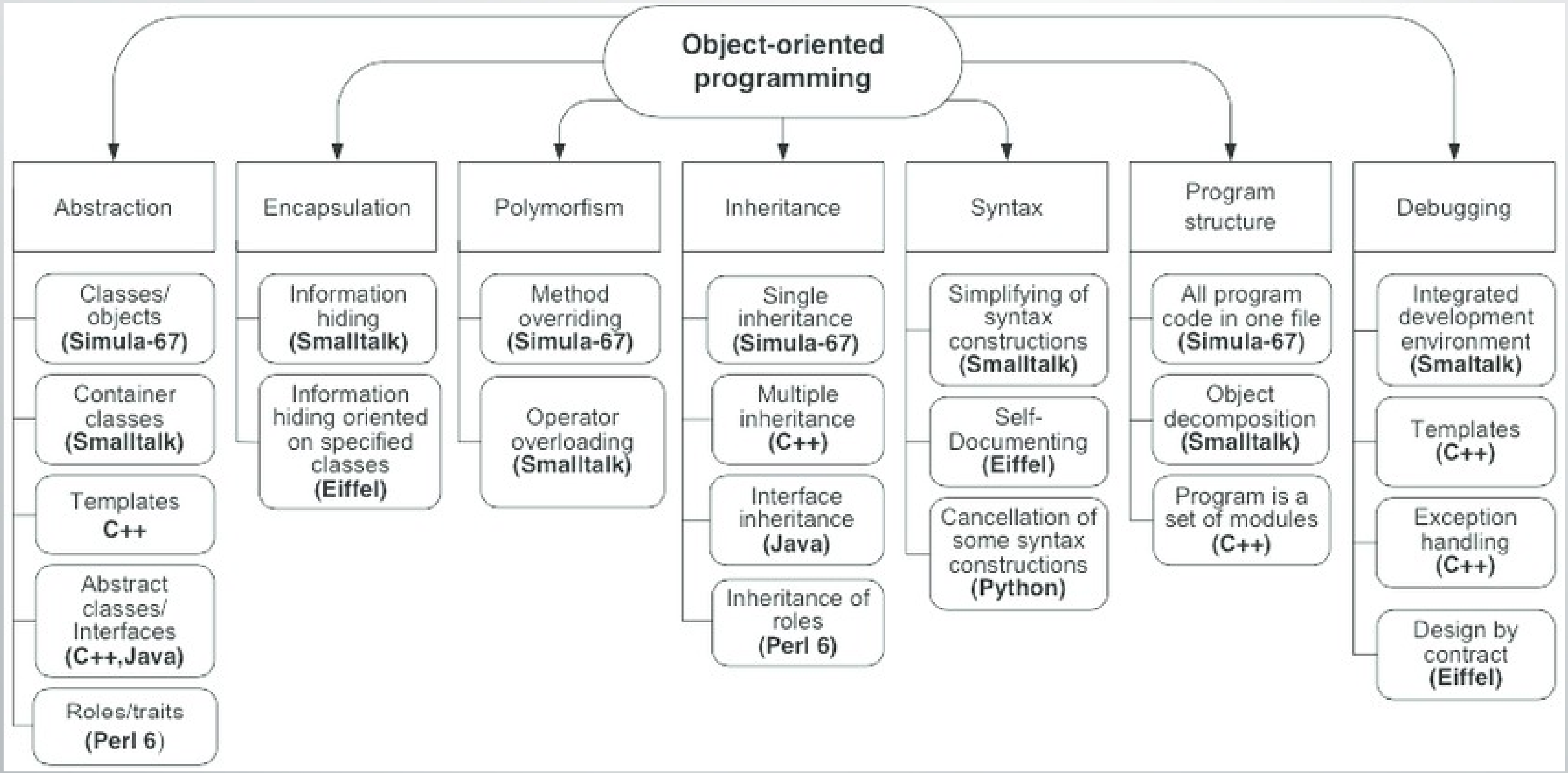

Emerging DBMS Technologies
OO Database
Object
Inheritence
Encapsulation
Polymorphism
OR Database
SQL Object-relational features
Evolution and Current trends
Implementation Issues
Active Database Concept
Oracle Triggers
Temporal Database
Mobile Database
Deductive Database & Query Processing
Spatial Database & Architecture
Multimedia Database
Graphical Information System


New language
2003
Scala
Homogeneity
2001
C#
Self-service, Partial or excessive actions, Taking out, Beforehand cushioning
2000
Perl 6
Universality
1995
Delphi
Java
1991
Preliminary action
Taking out, Homogeneity, Partial or excessive actions
Python
Taking out, Homogeneity, 'Intermediary'
1986
Eiffel
Preliminary action
1983
C++
Self-service, Homogeneity
1980
Smalltalk
Transition to a Supersystem, 'Intermediary'
1967
Simula-67

Need of Object-oriented Database Model

1) Object-oriented databases were proposed to meet the needs of complex applications.
2) ODBMS is the increasing use of object-oriented programming languages in developing software applications which can be easily integrated with OODB.
3) OODBMS is closely coupled with OOPL.

Object Oriented Features


Object-oriented programming
Abstraction
Classes/ objects (Simula-67)
Container classes (Smalltalk)
Templates C++
Abstract classes/ Interfaces (C++,Java)
Roles/traits (Perl 6)
Encapsulation
Information hiding (Smalltalk)
Information hiding oriented on specified classes (Eiffel)
Polymorfism
Method overriding (Simula-67)
Operator overloading (Smalltalk)
Inheritance
Single inheritance (Simula-67)
Multiple inheritance (C++)
Interface inheritance (Java)
Inheritance of roles (Perl 6)
Syntax
Simplifying of syntax constructions (Smalltalk)
Self-Documenting (Eiffel)
Cancellation of some syntax constructions (Python)
Program structure
All program code in one file (Simula-67)
Object decomposition (Smalltalk)
Program is a set of modules (C++)
Debugging
Integrated development environment (Smaltalk)
Templates (C++)
Exception handling (C++)
Design by contract (Eiffel)

OO Database Concepts

1) Object:
2) Object Structure:
3) Object Identity:
4) Type Constructor:
5) Types of Object:
6) Encapsulation of operations:
7) Type hierarchies and inheritance:
8) Extents:
9) Polymorphism and operator overloading:
10) Versioning and Configuration:

OO Database Concepts

Object

- An object or real world thing has two components; state (value) and behavior(operations).
- It is somewhat similar to a program variable in a programming language,
- Except that it will typically have a complex data structure as well as specific operations defined by the programmer.

OO Database Concepts

Object Identity

- An OO database system provides a unique identity to each independent object stored in the database are independent of their attribute values.
- This unique identity is typically implemented via a unique, system-generated object identifier, or OID.
- The value of an OID is not visible to the external user, but it is used internally by the system to identify each object uniquely and to create and manage inter-object references.
- OID should be immutable.
- OID preserves the identity of the real-world object being represented.

OO Database Concepts

Type Constructor:

- The type constructor is defined by Object Definition Language ODL.
- The basic type (atom), tuple type, collection or bulk type (set, list, bag, array).
- **Atom** represents all basic atomic values, such as integers, real numbers, character strings, Booleans, and any other basic data types that the system supports directly. The only case where an actual value appears in the state of an object is of type atom.
- This model of objects allows arbitrary nesting of the set, list, tuple and other constructors. The state of an object that is not of type atom will refer to other objects by their object identifiers.
- **Tuple** is often called a structured type, since it corresponds to the struct construct in the C and C++. Object in tuple consists of sub-objects which can be viewed as single object.
- **Collection Type** has state of the object in the form of a collection of objects that may be unordered (such as a set or a bag) or ordered (such as a list or an array).

OO Database Concepts

Collection Type

- **Set** is unordered collection of objects without duplicates or distinct.
- **Bag** is unordered collection of objects that allow duplicates.
- **List** is ordered collection of objects that allow duplicates having arbitrary no. of elements.
- **Array** is ordered collection of objects without duplicates. Array has a maximum fixed size.
- **Dictionary** is unordered sequence of key-value pairs without duplicate keys.
- **Complex objects** are based on other objects. There are two types of complex objects structured and unstructured.

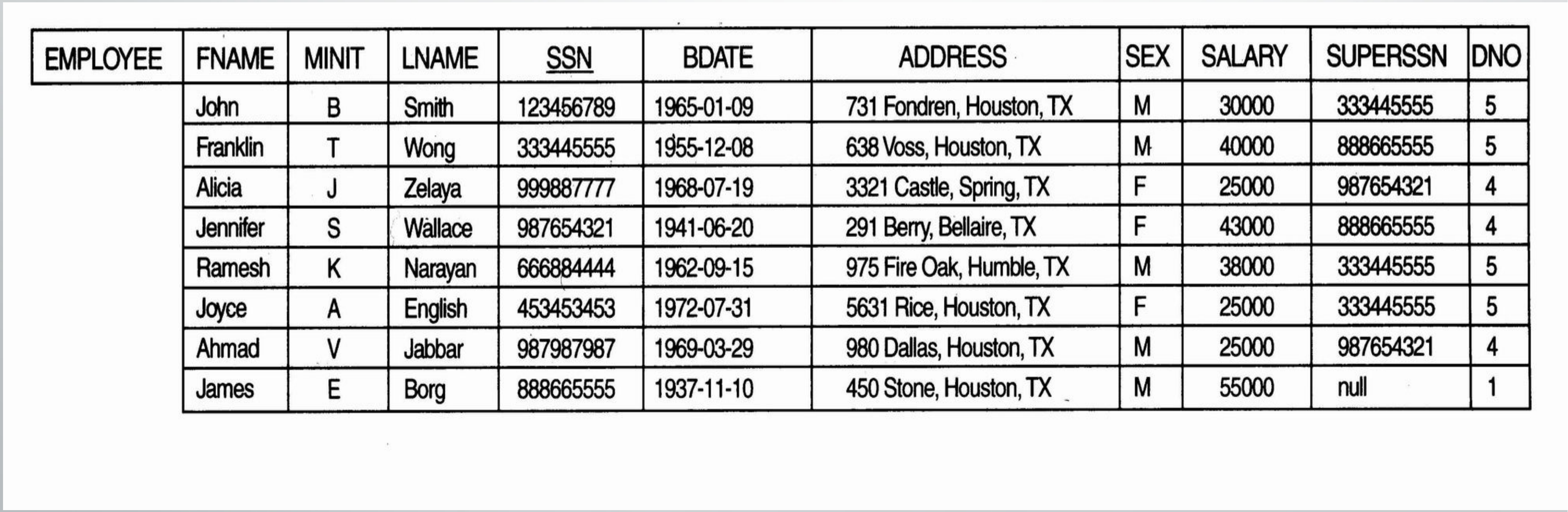
OO Database Concepts

Object Structure

- In OODB, the state(current value) of a complex object may be constructed from other objects by using certain type constructors.
- Each object can be viewed as a triple(i,c,v):
  → i = unique object identifier(OID)
  → c = type constructor(indicate how the object value is constructed)
  → v = the object state(current value)

Object Identity, Structure & Type Constructors

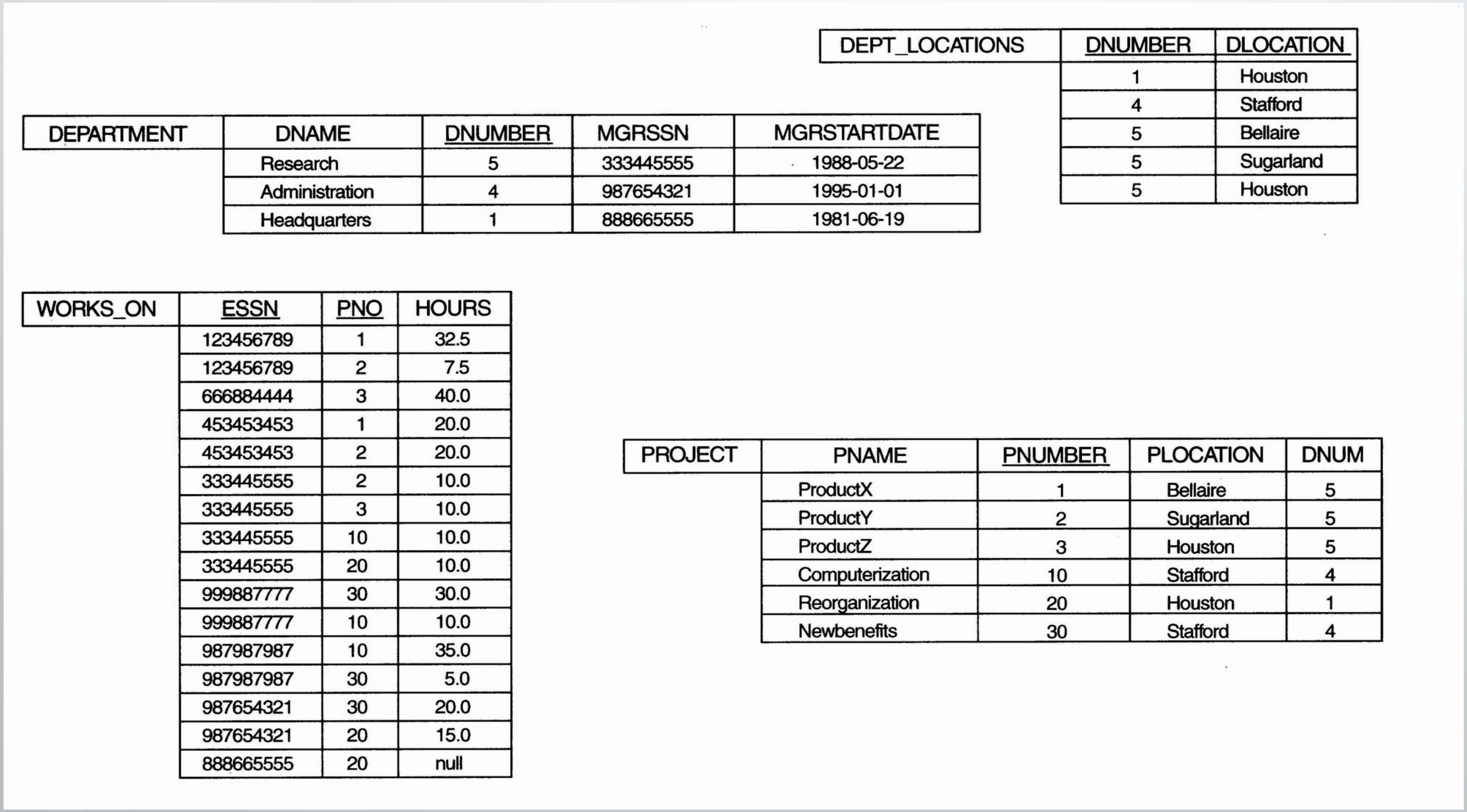
- **Example 1**, one possible relational database state corresponding to COMPANY schema



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

Object Identity, Structure & Type Constructors

❑ **Example 1 (cont.):**



DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

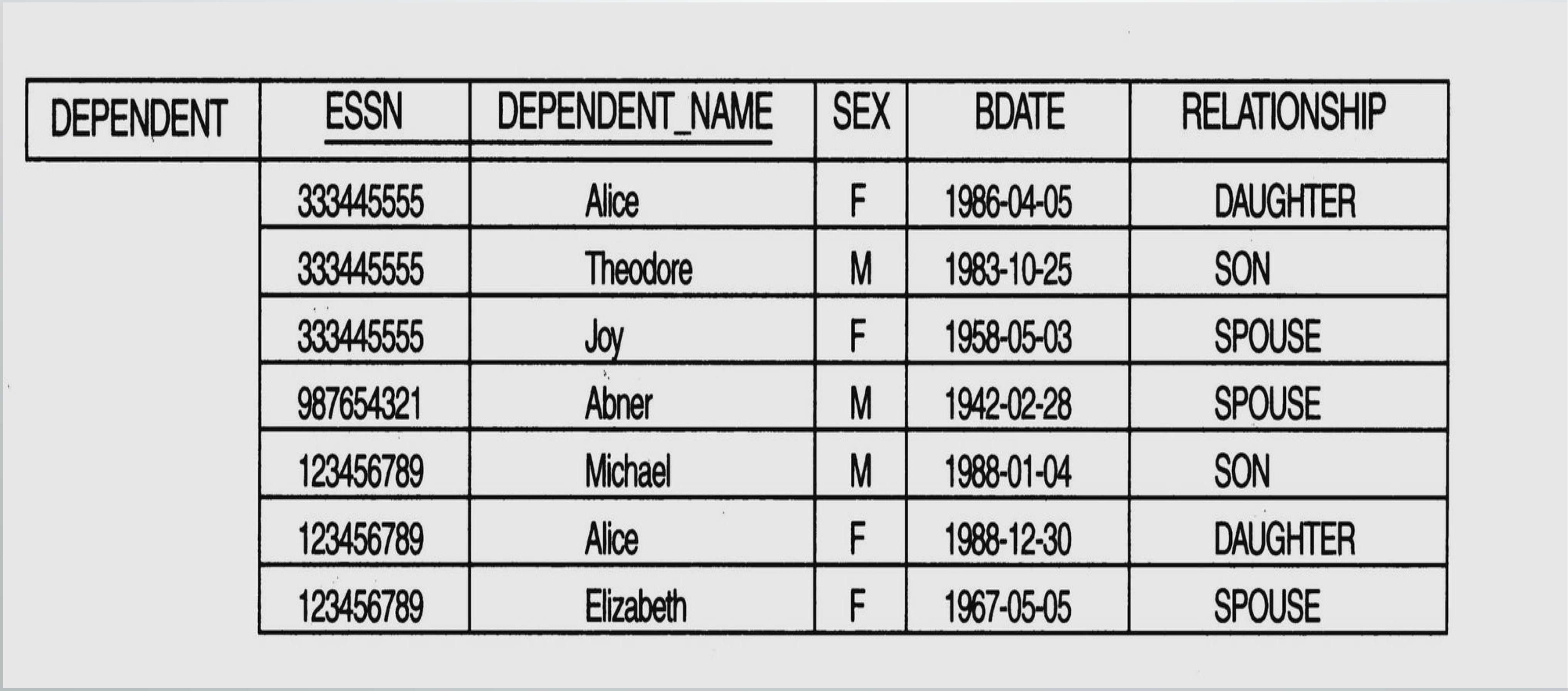
DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

Object Identity, Structure & Type Constructors

- **Example 1 (cont.)**



DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
333445555	Alice	F	1986-04-05	DAUGHTER
333445555	Theodore	M	1983-10-25	SON
333445555	Joy	F	1958-05-03	SPOUSE
987654321	Abner	M	1942-02-28	SPOUSE
123456789	Michael	M	1988-01-04	SON
123456789	Alice	F	1988-12-30	DAUGHTER
123456789	Elizabeth	F	1967-05-05	SPOUSE

Object Identity, Structure & Type Constructors

- **Example 1 (cont.)**

  We use i_1, i_2, i_3, . . . to stand for unique system-generated object identifiers. Consider the following objects:

  o_1 = (i_1, atom, ‘Houston’)

  o_2 = (i_2, atom, ‘Bellaire’)

  o_3 = (i_3, atom, ‘Sugarland’)

  o_4 = (i_4, atom, 5)

  o_5 = (i_5, atom, ‘Research’)

  o_6 = (i_6, atom, ‘1988-05-22’)

  o_7 = (i_7, set, {i_1, i_2, i_3})

Object Identity, Structure & Type Constructors

- **Example 1(cont.)**

  o_8 = (i_8, tuple, <dname:i_5, dnumber:i_4, mgr:i_9, locations:i_7, employees:i_{10}, projects:i_{11}>)

  o_9 = (i_9, tuple, <manager:i_{12}, manager_start_date:i_6>)

  o_{10} = (i_{10}, set, {i_{12}, i_{13}, i_{14}})

  o_{11} = (i_{11}, set {i_{15}, i_{16}, i_{17}})

  o_{12} = (i_{12}, tuple, <fname:i_{18}, minit:i_{19}, lname:i_{20}, ssn:i_{21}, . . ., salary:i_{26}, supervisor:i_{27}, dept:i_8>)

  . . .

Object Identity, Structure & Type Constructors

Example 1 (cont.)

- The first six objects listed in this example represent atomic values.
- Object seven is a set-valued object that represents the set of locations for department 5; the set refers to the atomic objects with values {‘Houston’, ‘Bellaire’, ‘Sugarland’}.
- Object 8 is a tuple-valued object that represents department 5 itself, and has the attributes DNAME, DNUMBER, MGR, LOCATIONS, and so on.

Object Identity, Structure & Type Constructors

Example 2:

This example illustrates the difference between the two definitions for comparing object states for equality.

o_1 = (i_1, tuple, <a_1:i_4, a_2:i_6>)

o_2 = (i_2, tuple, <a_1:i_5, a_2:i_6>)

o_3 = (i_3, tuple, <a_1:i_4, a_2:i_6>)

o_4 = (i_4, atom, 10)

o_5 = (i_5, atom, 10)

o_6 = (i_6, atom, 20)

Object Identity, Structure & Type Constructors

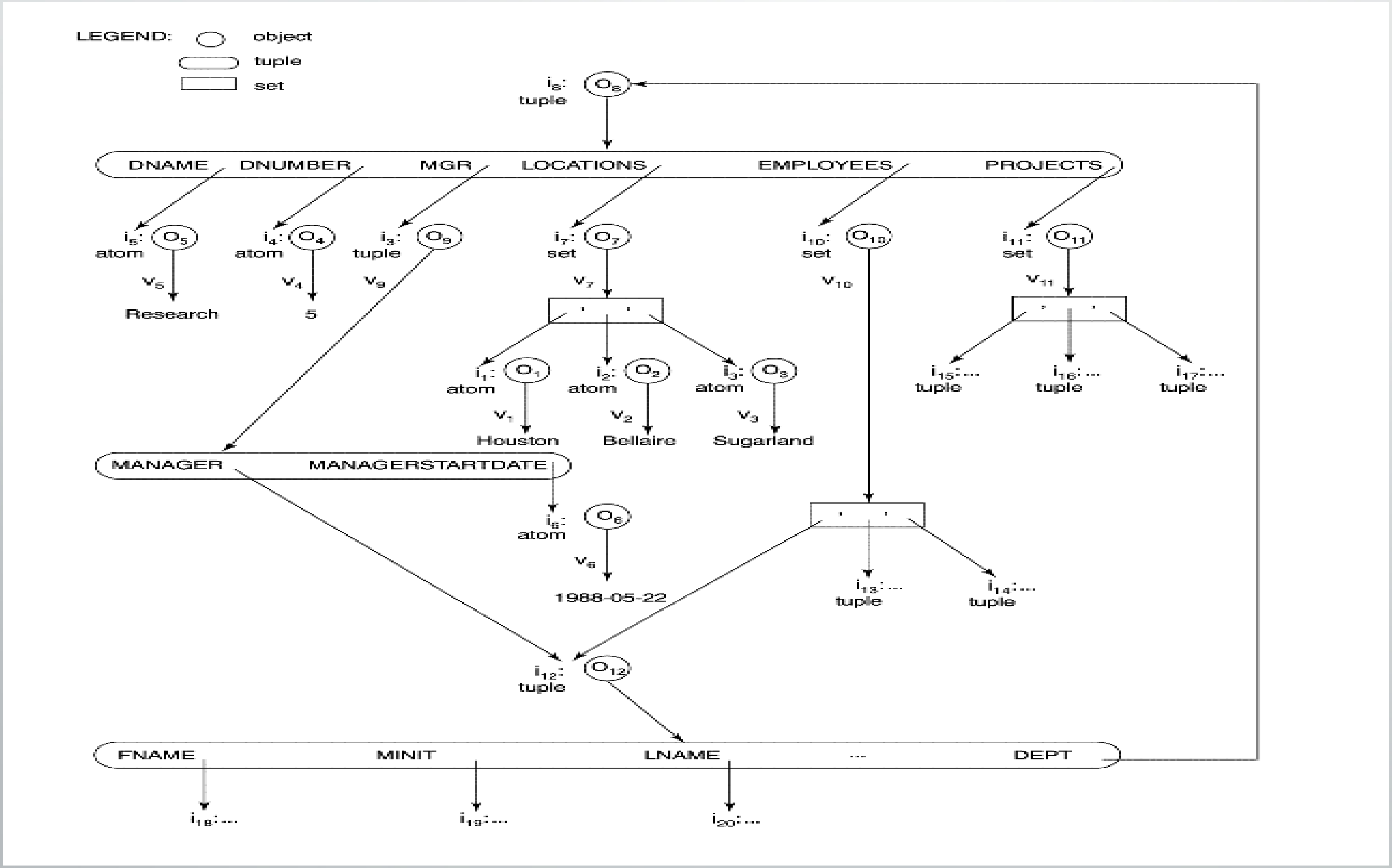
Example 2 (cont.):

In this example, The objects o_1 and o_2 have ***equal*** states, since their states at the atomic level are the same but the values are reached through distinct objects o_4 and o_5.

However, the states of objects o_1 and o_3 are ***identical***, even though the objects themselves are not because they have distinct OIDs. Similarly, although the states of o_4 and o_5 are identical, the actual objects o_4 and o_5 are equal but not identical, because they have distinct OIDs.

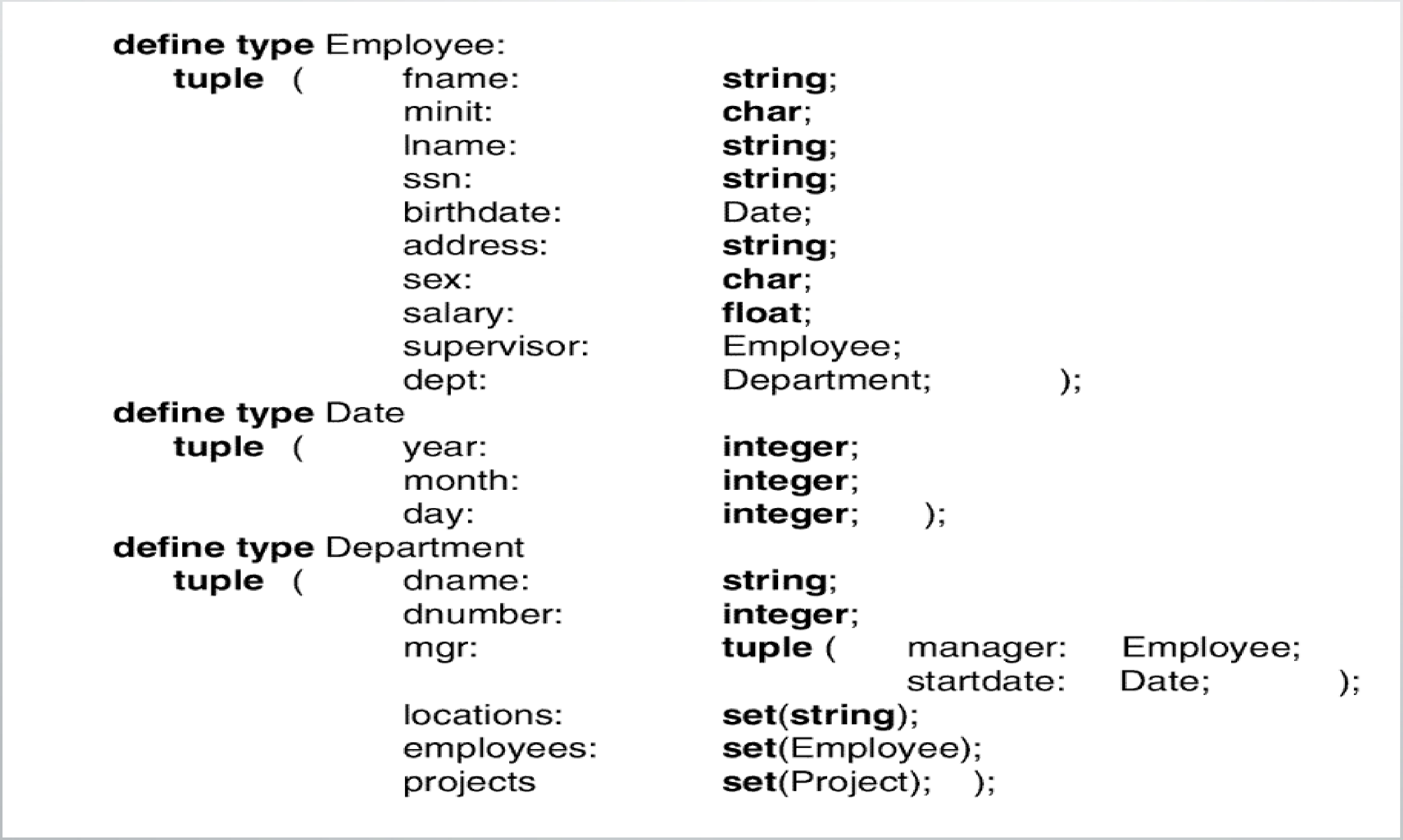
Object Identity, Structure & Type Constructors

Figure: Representation of a DEPARTMENT complex object as a graph



Object Identity, Structure & Type Constructors

Figure 20.2 Specifying the object types Employee, date, and Department using type constructors



```
define type Employee:
    tuple  (       fname:              string;
                   minit:              char;
                   lname:              string;
                   ssn:                string;
                   birthdate:          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;
                                                  startdate:  Date;         );
                   locations:          set(string);
                   employees:          set(Employee);
                   projects            set(Project);   );
```

OO Database Concepts

Types of Object:

Transient and Persistent Objects: Objects in an OOPL exist only during program execution and are hence called transient objects.

Objects that persist beyond program termination and can be retrieved later and shared by programs and applications are called persistent objects.

In OODBMS not all objects are persistent some are transient. In an ER Model all objects are persistent.

OO Database Concepts

Encapsulation of operations:

Both the object structure and the operations that can be applied to objects are included in the object class definitions.

To encourage encapsulation, an operation is defined in two parts:

1. Signature or interface of the operations, specifies the operation name and arguments.

2. Method or body, specifies the implementation of the operation.

Operations can be invoked by passing a message to an object, which includes the operation name and the parameters. The object then executes the method for that operation.

This encapsulation permits modification of the internal structure of an object, as well as the implementation of its operations, without the need to disturb the external programs that invoke these operations.

OO Database Concepts

Type hierarchies and inheritance:

Object types can be specified by using a type hierarchy, which allows the inheritance of both attributes and methods of previously defined types.

Multiple inheritances are allowed in some models.

Multiple inheritance in a type hierarchy occurs when a certain subtype T is a subtype of two (or more) types and hence inherits the functions (attributes and methods) of both super types.

For example, we may create a subtype ENGINEERING_MANAGER that is a subtype of both MANAGER and ENGINEER. This leads to the creation of a type lattice rather than a type hierarchy.

Issues in Multiple Inheritance

One problem: if a subtype inherits two distinct methods with the same name from two different super-types

A solution: check for ambiguity when the subtype is created and let the user choose the function

Another solution: use some system default

A third solution: disallow multiple inheritance if ambiguity occurs

Selective Inheritance:

- → When a subtype inherits only a few methods
- → This mechanism is not usually provided by OODBMS

OO Database Concepts

Extents:

In most OO databases, the collection of objects in an extent has the same type or class.

However, since the majority of OO databases support types, we assume that extents are collections of objects of the same type.

All persistent objects of a particular type can be stored in an extent. Extents corresponding to a type hierarchy have set/subset constraints enforced on them.

A persistent collection holds a collection of objects that is stored permanently in the database and hence can be accessed and shared by multiple programs whereas transient collection exists temporarily during the execution of a program but is not kept when the program terminates.

OO Database Concepts

Polymorphism and operator overloading:

Operations and method names can be overloaded to apply to different object types with different implementations.

Operator polymorphism: It refers to an operation’s ability to be applied to different types of objects; in such a situation, an operation name may refer to several distinct implementations, depending on the type of objects it is applied to. This feature is also called operator overloading.

Versions

- Ability to maintain several versions of an object
- Commonly found in many software engineering and concurrent engineering environments
- Merging and reconciliation of various versions is left to the application program
- Some systems maintain a version graph

Configuration

- A configuration is a collection compatible versions of modules of a software system (a version per module)

OO Database Concepts

Versions and Configuration:

Some OO systems provide support for maintaining several versions of the same object. Some OO systems provide capabilities for dealing with multiple versions of the same object (a feature that is essential in design and engineering applications).

For example, an old version of an object that represents a tested and verified design should be retained until the new version is tested and verified, very crucial for designs in manufacturing process control, architecture, software systems etc.

Object Oriented Database Concepts

- The relational database model has been successfully implemented in wide variety of application areas.
- However it doesn’t easily support the distribution of one database across a number of servers, which is the natural node for the internet.
- Due to this reason, object oriented database management system (OODBMS) was developed. In this database, the user can define their own data access methods, the representation of data and the method of manipulating it.
- An object oriented database stores and maintains objects.
- An object is an item that can contain both data and the procedures that manipulate the data. With an object oriented database, a system can store and access unstructured data such as pictures, video clips, sounds etc more efficiently than relational database.

Object Oriented Database Concepts

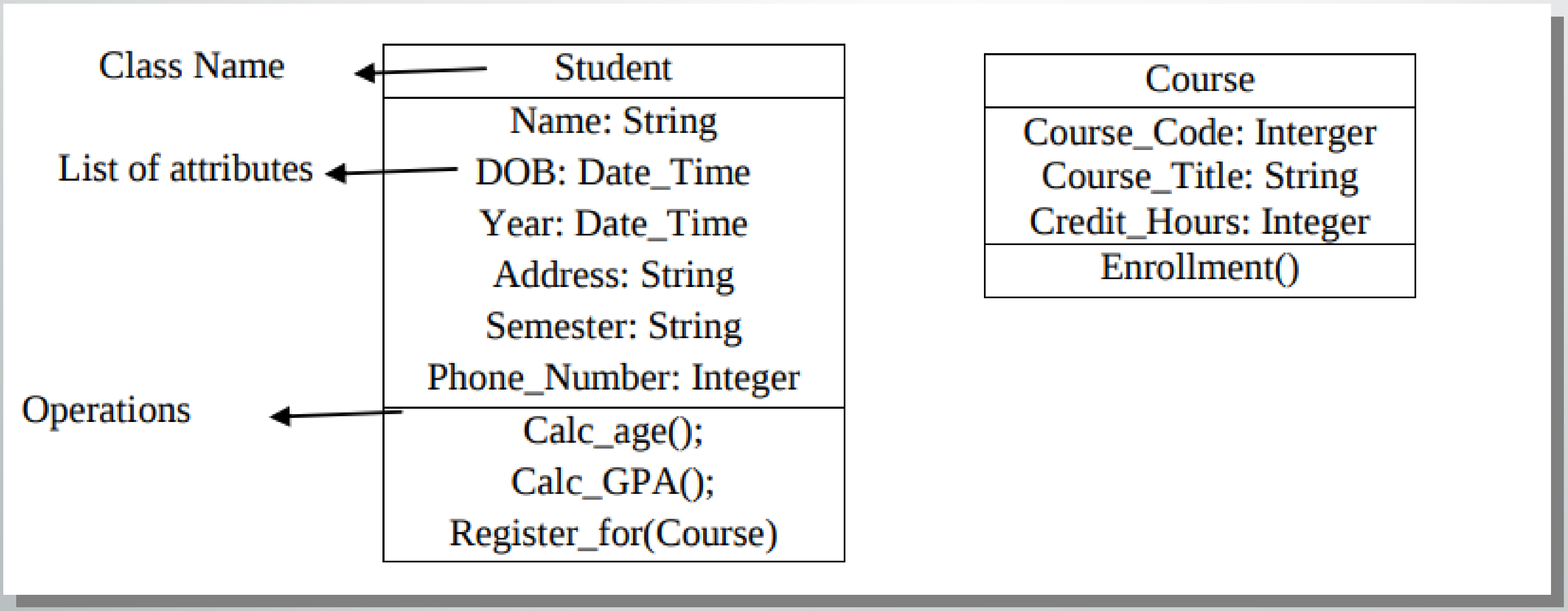
- An object database (also object-oriented database management system) is a database management system in which information is represented in the form of objects as used in object-oriented programming.
- Object databases are different from relational databases which are table-oriented.
- Object-relational databases are a hybrid of both approaches.
- Object databases have been considered since the early 1980s.
- Object databases based on persistent programming acquired a niche in application areas such as engineering and spatial databases, telecommunications, and scientific areas such as high energy physics and molecular biology.
- Another group of object databases focuses on embedded use in devices, packaged software, and real-time systems.

Object Oriented Database Concepts

- OO model provides all the facilities associated with object oriented paradigm.
- It enables us to create classes, organize objects, structure an inheritance hierarchy and call methods of other classes.
- Besides these, it also provides the facilities associated with standard database systems.
- However, object oriented database systems have not yet replaced the RDBMS. WWW is example of object oriented database.
- The static structure of object oriented model is represented by class diagram. Below example shows class diagram for two classes.

Object Oriented Database Concepts

- The static structure of object oriented model is represented by class diagram.
- Below example shows class diagram for two classes.



Comparison with RDBMSs

- An object database stores complex data and relationships between data directly, without mapping to relational rows and columns, and this makes them suitable for applications dealing with very complex data.
- Objects have a many to many relationship and are accessed by the use of pointers. Pointers are linked to objects to establish relationships.
- Another benefit of an OODBMS is that it can be programmed with small procedural differences without affecting the entire system.
- This is most helpful for those organizations that have data relationships that are not entirely clear or need to change these relations to satisfy the new business requirements.

PROMISES OF OBJECT ORIENTED SYSTEMS

- **Reduced maintenance:**

  The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviors may be reused and incorporated into new behavior

- **Real-world modeling:**

  Object-oriented systems tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and, objects are associated with behaviors The model is based on objects rather than on data and processing.

- **Improved reliability:**

  Object-oriented systems promise to be far more reliable than traditional systems, primarily because new behaviors can be built from existing objects.

- **High code re-usability:**

  When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviors from all super classes in which it participates.

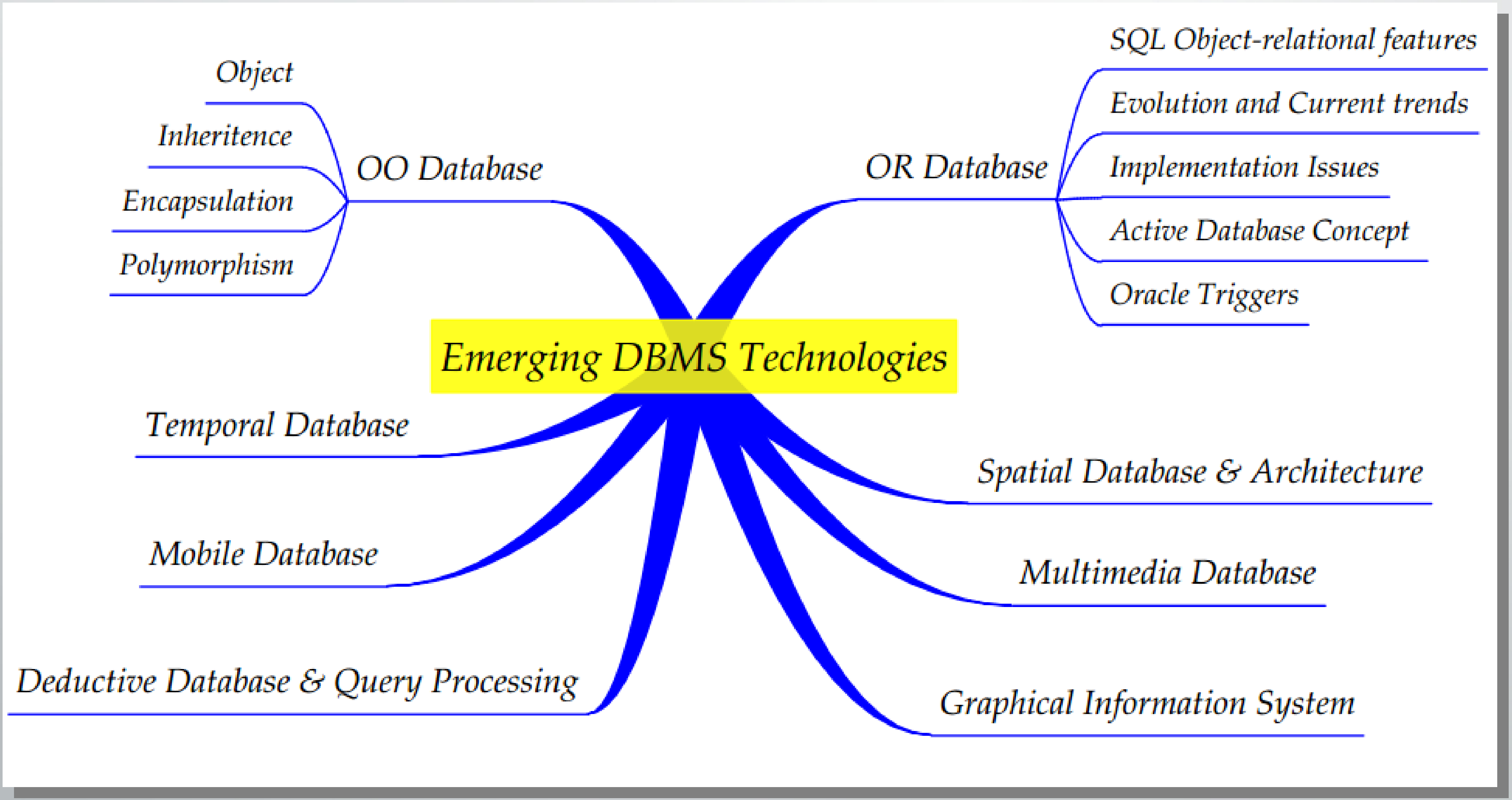
Advantages of OODBMS

- The object-oriented data model allows the 'real world' to be modeled more closely.
- Reusability
- Easier Design-Reflect application.
- Unlike traditional databases (such as hierarchical, network or relational), the object oriented database are capable of storing different types of data, for example, pictures, audio, including text, numbers and so on.
- OODBMSs allow new data types to be built from existing types.
- Multiple Inheritances.
- Improved performance than traditional relational database model.
- Encapsulation of operations and user defined method.

Disadvantages of OODBMS

- There is no universally agreed data model for an OODBMS, and most models lack a theoretical foundation.
- In comparison to RDBMSs the use of OODBMS is still relatively limited. This means that we do not yet have the level of experience that we have with traditional systems.
- There is a general lack of standards of OODBMSs. There is no standard object-oriented query language.
- Many OODBMSs use locking as the basis for concurrency control protocol. However, if locking is applied at the object level, locking of an inheritance hierarchy may be problematic, as well as impacting performance.
- The increased functionality provided by the OODBMS makes the system more complex than that of traditional DBMSs. This complexity leads to products that are more expensive and more difficult to use.
- Currently, OODBMSs do not provide adequate security mechanisms. The user cannot grant access rights on individual objects or classes

Emerging DBMS Technologies


Emerging DBMS Technologies
OO Database
Object
Inheritence
Encapsulation
Polymorphism
OR Database
SQL Object-relational features
Evolution and Current trends
Implementation Issues
Active Database Concept
Oracle Triggers
Temporal Database
Mobile Database
Deductive Database & Query Processing
Spatial Database & Architecture
Multimedia Database
Graphical Information System