CS9152 – DATABASE TECHNOLOGY

UNIT – II

OBJECT ORIENTED DATABASES

TEXT BOOK

1. Elisa Bertino, Barbara Catania, Gian Piero Zarri, "Intelligent Database Systems", Addison-Wesley, 2001.

REFERENCES

1. Carlo Zaniolo, Stefano Ceri, Christos Faloustsos, R.T.Snodgrass, V.S.Subrahmanian, "Advanced Database Systems", Morgan Kaufman, 1997.
2. N.Tamer Ozsu, Patrick Valduriez, "Principles of Distributed Database Systems", Prentice Hal International Inc. , 1999.
3. C.S.R Prabhu, "Object-Oriented Database Systems", Prentice Hall Of India, 1998.
4. Abdullah Uz Tansel Et Al, "Temporal Databases: Theory, Design And Principles",Benjamin Cummings Publishers , 1993.
5. Raghu Ramakrishnan, Johannes Gehrke, "Database Management Systems", Mcgraw Hill, Third Edition, 2004.
6. Henry F Korth, Abraham Silberschatz, S. Sudharshan, "Database System Concepts", Fourth Ediion, McGraw Hill , 2002.
7. R. Elmasri, S.B. Navathe, "Fundamentals of Database Systems", Pearson Education, 2004.

Syllabus:

UNIT II OBJECT ORIENTED DATABASES 10

Introduction to Object Oriented Data Bases - Approaches - Modeling and Design – Persistence – Query Languages –Transaction – Concurrency – Multi Version Locks –Recovery.

Table of Contents

Topic – 1: Introduction to Object Oriented Data Bases

Object Databases

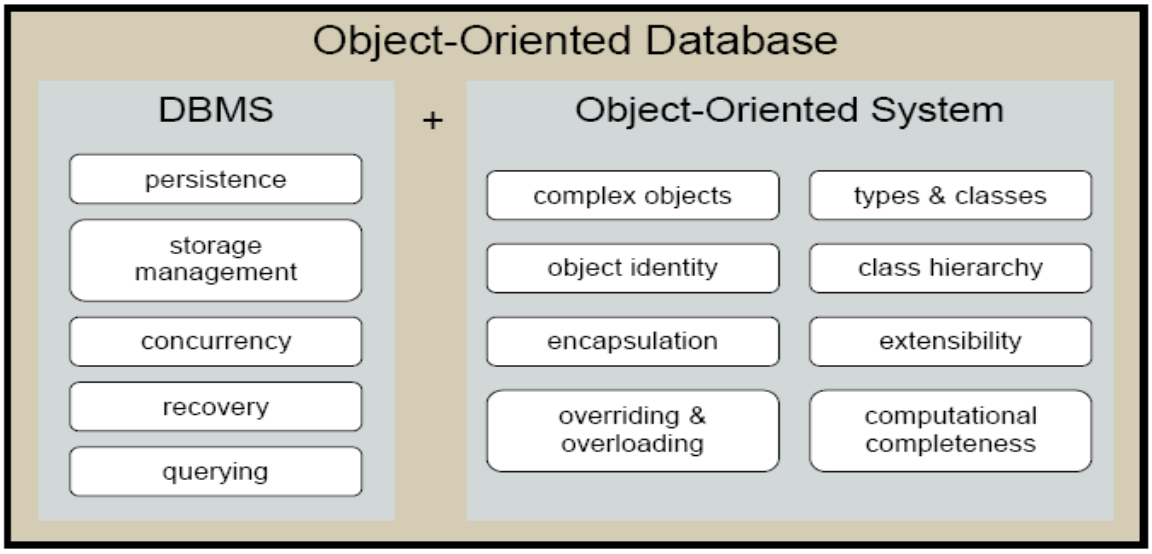
- Became commercially popular in mid 1990's
- You can store the data in the same format as you use it. No paradigm shift.
- Did not reach full potential till the classes they store were decoupled from the database schema.
- Open source implementation available – low cost solution now exists.
- Motivation: to overcome weaknesses of relational approach:
  - Richer data models.
  - Closer integration with programming languages.
- Kinds of object database:
  - Object relational (Oracle, DB2, PostgreSQL).
  - Semantic data model (Jasmine).
  - Programming language centred (Objectivity, FastObjects, Versant, ObjectStore).

OOBD Basics:

- A basic aim of OODB is to "*raise the level of abstraction*". That is, to provide access to data via methods that hide the complexity of low level access.
- Applications suited to OODB tend to be those that require complex SQL queries:
  - Computer-aided design and manufacturing (CAD/CAM)
  - Computer-aided software engineering (CASE)
  - Geographic information systems (GIS)
  - Document storage and retrieval

What is Object Oriented Database? (OODB)

- A database system that incorporates all the important object-oriented concepts
- Some additional features
  - Unique Object identifiers
  - Persistent object handling
  - Is the coupling of Object Oriented (OOP) Programming principles with Database Management System (DBMS) principles
  - Provides access to persisted objects using the same OO-programming language



Advantages of OODBS

- Designer can specify the structure of objects and their behavior (methods)
- Better interaction with object-oriented languages such as Java and C++
- Definition of complex and user-defined types
- Encapsulation of operations and user-defined methods

Object Database Vendors

- ✓ Matisse Software Inc.,
- ✓ Objectivity Inc.,
- ✓ Poet's FastObjects,
- ✓ Computer Associates,
- ✓ eXcelon Corporation
- ✓ Db4o

Database Structures

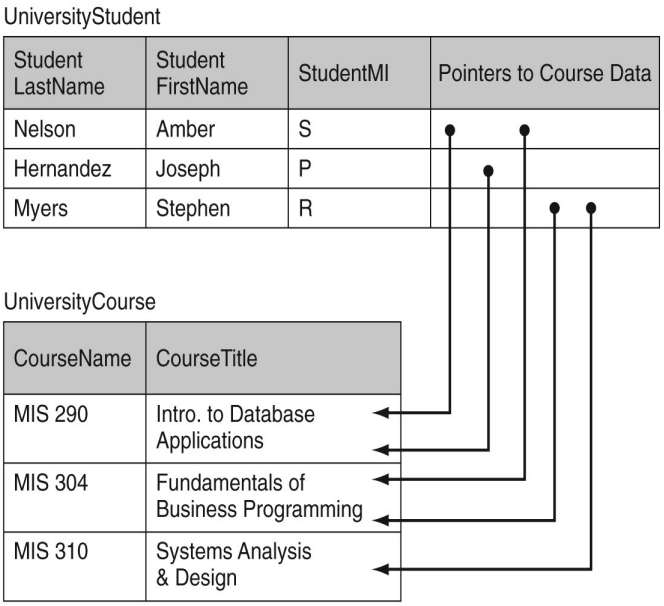
- Three primary database structures
  - Hierarchical databases
  - Relational databases
  - Object-Oriented databases
- Each structure stores data entity instance values and represents relationships differently

Hierarchical Databases

- Entities have parent-to-child relationship
- Uses pointers to create relationship between associated data items

- Pointer
  - Unique address value that defines the physical location of where data is stored on a storage device
  - Links data values of the parent entity instance with multiple child entity instances

FIGURE 1-4 Hierarchical database structure



- **Problems with Hierarchical Databases**
  - Difficult to move to new storage medium
    - Data is physically dependent on its location on the storage media
  - Changes to database structure require rewriting of programs
    - Time-consuming
    - Expensive

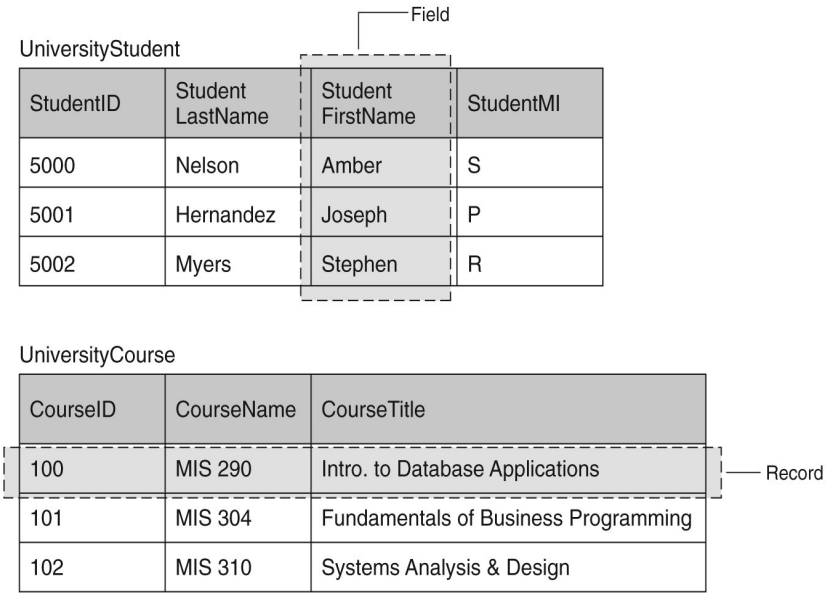
Relational Databases

Basic Concepts of RDB

- Tables
  - Stores the data
- Records (like a row in a table)
  - Contains data about an individual entity instance
- Fields (table columns)
  - Attributes that are associated with individual data values
- Key fields

- Create relationship among records in different tables

FIGURE 1-5 Examples of relational database tables

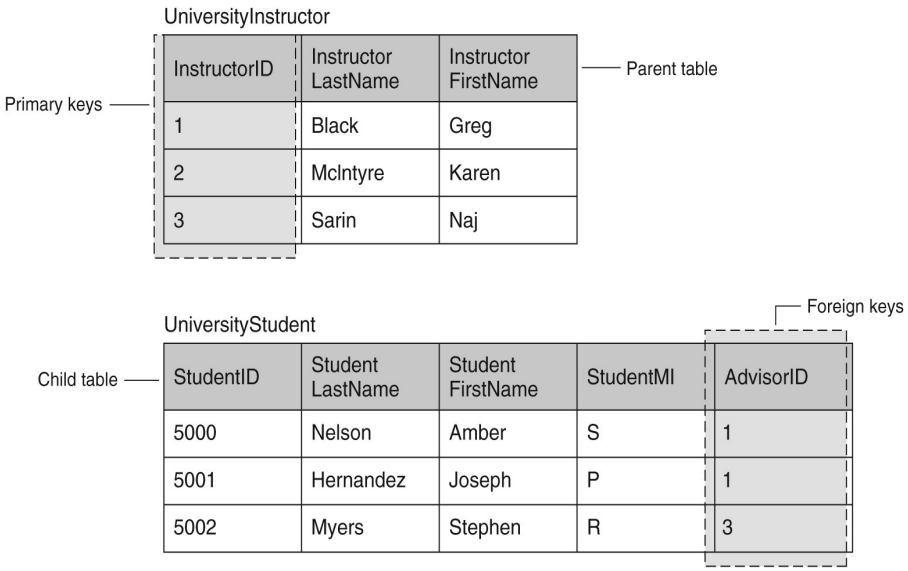


Relational Databases – Key Fields

- Primary key
  - Table field that uniquely identifies a record
  - Cannot be NULL
  - Mandatory for each table
- Surrogate key (SUK)
  - Fields that did not appear automatically (VIN or SIN) but were created by designers to be a designated primary key (e.g., student #, customer ID)
  - Unique, can be generated automatically for new records
- Foreign keys
  - Create a relationship between two tables
  - A primary key field in one table (parent) and acts as a foreign key in another table (child)
  - Enforces referential integrity
    - When a table is created with a foreign key, all foreign key values must exist in the parent table
    - **Populate the table with primary keys first!!!**
- Composite keys
  - A unique primary key created by combining multiple key fields in a table

- Made up of foreign keys fields that are primary keys in other tables
- E.g., CustomerID and OrderID
- Always optional
- You will use them in all projects

FIGURE 1-6 Creating relationships using foreign keys



Types of relationships

- One-to-one
  - Each student has only one student #, each student # belongs to only one student
- One-to-many
  - A customer may place many orders, but each order is associated with only one customer
- Many-to-many
  - Several profs may teach the same course, the same course may be taught by several different profs (in case of multiple sections)

FIGURE 1-7 Composite key

CourseEnrollment

CourseID	StudentID	CourseGrade
100	5000	A
100	5001	C
101	5000	B
101	5002	B
102	5002	A

Composite key

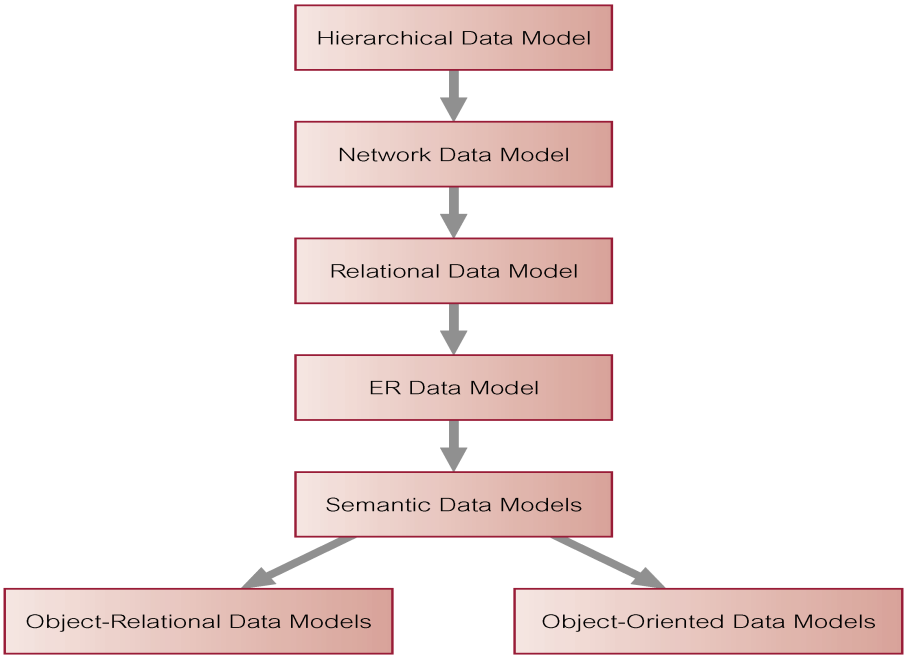
UniversityStudent

StudentID	Student LastName	Student FirstName	StudentMI
5000	Nelson	Amber	S
5001	Hernandez	Joseph	P
5002	Myers	Stephen	R

UniversityCourse

CourseID	CourseName	CourseTitle
100	MIS 290	Intro. to Database Applications
101	MIS 304	Fundamentals of Business Programming
102	MIS 310	Systems Analysis & Design

The Hierarchies of Data Models



Object-Oriented Databases

OBJECT ORIENTED DATABASES

- **Object**
  - Similar to an entity
  - Stores data as well as methods
    - Methods - the programs that interact with the data
  - Two components:
    - state (value) and behavior (operations)
  - Similar to program variable in programming language, except that it will typically have a complex data structure as well as specific operations defined by the programmer

- **Object instance**
  - Similar to a record
  - Refers to the single unique object
- **Object class**
  - Collection of similar objects
  - State - specifies its attribute values and the relationships of all object instances within the class
  - Behavior - represents the actions of its instances within the database application

- **Object identity**:

  Objects have unique identities that are independent of their attribute values.

  • An object retains its identity even if some or all of the values of variables or definitions of methods change over time.

  • Object identity is a stronger notion of identity than in programming languages or data models not based on object orientation.

  – Value – data value; e.g. primary key value used in relational systems.

  – Name – supplied by user; used for variables in procedures.

  – Built-in – identity built into data model or programming language.

• no user-supplied identifier is required.

• Is the form of identity used in object-oriented systems

Object Structure

- The state (current value) of a complex object may be constructed from other objects (or other values) by using certain type constructors

- Can be represented by (i,c,v)
  - i is an unique id
  - c is a type constructor

- v is the object state

- Constructors
  - Basic types: atom, tuple and set
  - Collection type: list, bag and array

- **Complex objects:**
  - the value of each object can be an object or set of objects referred to by OIDs
  - eg: attribute of a person may be children, which is a set of OIDs of other persons

- **Encapsulation**
  - abstraction that forces a separation between the external interface and the internal implementation
  - hides unnecessary details of implementation
  - allows code and data to be packaged together
  - each object has methods (allowed functions/procedures)
  - methods provide a visible (public) interface and a hidden (private) implementation
  -
- **Abstract Data Types**
  - Class definition, provides extension to complex attribute types
- **Encapsulation**
  - Implementation of operations and object structure hidden
- **Inheritance**
  - Sharing of data within hierarchy scope, supports code reusability
  - classes can be a subclass of another class e.g., car of vehicle, the subclass inherits the attributes and methods of its superclass.

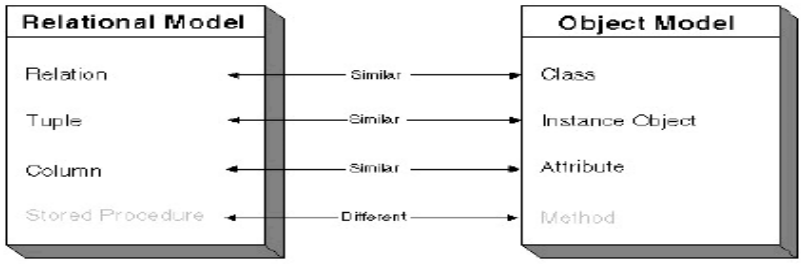
- **Polymorphism**
  - An operation's ability to be applied to different types of objects; in such a situation, an operation name may refer to several distinct implementations, depending on the type of objects it is applied to is called Operator overloading

- **OODBMS - Manage data objects**
  - Define the object classes, their associated attributes, and methods
  - Write commands to create individual object instances for each data item
- Each object has specific attribute values and relationships with other objects

- Advantages
  - Easy to use and maintain
  - Easy to store and manage sound and video clips
- Disadvantages
  - Expensive and time-consuming to migrate data to object classes
  - Poor performance in processing high transaction volumes

- Reasons for creation of Object Oriented Databases
  - Need for more complex applications
  - Need for additional data modeling features
  - Increased use of object-oriented programming languages

Relational/Object - Comparison



Class Hierarchy

class *person*{
string *name;*
string *address:*
};
class *customer* **isa** *person* {
int *credit-rating;*
};
class *employee* **isa** *person* {
date *start-date;*
int *salary;*
};
class *officer* **isa** *employee* {
int *office-number,*
int *expense-account-number,*
};

Some OODBMS's

Commercial	Open Source
Fast Objects (formerly Poet)	Ozone
Gemstone	XL2
Versant	FramerD
Ontos	Zope
Objectivity/DB	Academic - ObjectStore

Advantages and disadvantages of OODBMSs

Advantages

- Enriched Modeling Capabilities.
- Extensibility
- Removal of Impedance Mismatch
- More Expressive Query language.
- Support for schema evaluation.
- Support for long duration Ts.
- Applicability to advanced Database Apps.
- Improved performance.

Disadvantages

- Lack of Universal Data Model.
- Lack of experience.
- Lack of standards.
- Query Optimization compromises Encapsulation.
- Object Level locking may impact performance.
- Complexity
- Lack of support for views.
- Lack of support for Security.

Differences between OODB and Relational DB

OODB	RDB
Uses OO data model - Data is a collection of objects whose behaviour, state and relationships are stored as a physical entity.	Uses record-oriented model - Data is a collection of record types (relations), each having collection of records or tuples stored in a file.
Language dependence (OO-Language specific.	Language independence (via SQL)

No impedance mismatch in application using OODB	Impedance mismatch in application. Mapping must be performed.

Topic – 2: Approaches

The main four OODB approaches are:

i. Extended Relational Model Approach
ii. The Semantic Database Approach
iii. The object oriented database programming language extension approach
iv. The DBMS Generator Approach

i .Extended Relational Model Approach

☐ Several attempts were made towards achieving greater object orientation within the boundaries of the Relational Model.
☐ This model is the first model in this direction
☐ POSTGRES was the first largely successful attempt in this direction with a real implementation.
☐ STARBURST is a parallel approach with much greater robustness and scope but is yet to result in a commercial product.
☐ INGRESS and SQL DBII (extension R*) were the predecessors of POSTGRES and STARBURST

ii The Semantic Database Approach

☐ Semantic Database Models and Systems, evolved independently
as database extensions, offer semantic richness and object orientation
☐ Semantic models began with E-R model and became more sophisticated with advanced models as SDM, SAM and IFO.
☐ Commercial implementations as SIM established a clear place for semantic systems in the market

iii Object Oriented Database Programming language extension approach

☐ This approach is based on the object oriented programming paradigm evolved originally from languages such as C++ and Smalltalk offering an object oriented programming environment.
☐ An application program is developed in an environment which is basically an extension of object oriented programming language to database environment
☐ The implementation of such object oriented programming language in terms of a compiler, pre-processor and program execution environment are usually extended and enhanced to make provision for incorporating a data management facility and the essential and fundamental database system features.

☐ Systems available in the market are O2, Objectstore, Gbase, ONTOS, Gemstone, ITASCA and ORION

The DBMS Generator Approach

☐ This approach is analogous to the system generation approach
☐ Application system will be generated from standard modules of toolkits available in the module library.
☐ Features are specified by DBI (Database Implementer)

Categories of Semantic database models

Semantic database models
Relational Model
Extension Functional Models
Entity association models
Formal models
Object based models

Relational model extensions

☐ RM/T is a record based database model that incorporates a set of semantic constructs in relational database model
☐ A type or a class is represented by a type relation that contains a symbolic unique identifier for every tuple member or entity.
☐ Attributes of type members are represented in a separate n-ary relation that relates every unique identifier of a member with a set of values for its properties.

Functional database models

☐ A functional database model views objects, properties of objects, object classification and inter object relationship types uniformly and defines them as functions
☐ Eg. Daplex, BINARY model

Entity association models

☐ A database is defined as a collection of entities and in terms of the relationships among these types
☐ Eg.
NASEMOD
E-R Model
Event Model
SAM*

Object Based Model

☐ Employ the concepts of objects, inter object relationships and object relationships and object associations

☐ Eg.

Sembase

OSAM*

Other OODB Approaches

- **OOPS Extension**
  DB support is added to a programming language. e.g.:
  - ONTOS (C++)
  - ObjectStore (C++)
  - GemStone (Smalltalk, Opal, C++)
- **RDBMS Extension**
  OO support is added to a relational database, ie. Object-Relational DBMS. e.g.:
  - POSTGRES (University INGRES, QUEL)
  - Starburst (IBM research)
  - IRIS (OSQL, SQL)
- **New OODBMS**
  Fully OODBMS. Only stores data in the form of objects, so does not support tuples. Performance is optimized for fast access of objects from the server and transparent access of objects from the client side. e.g.:
  - ORION (Common LISP)
  - O2 (C, CO2)

Topic – 3: Modeling and Design

Object-Oriented Data Model

No one agreed object data model. One definition:

Object-Oriented Data Model (OODM)

– Data model that captures semantics of objects supported in object-oriented programming.

Object-Oriented Database (OODB)

– Persistent and sharable collection of objects defined by an ODM.

Object-Oriented DBMS (OODBMS)

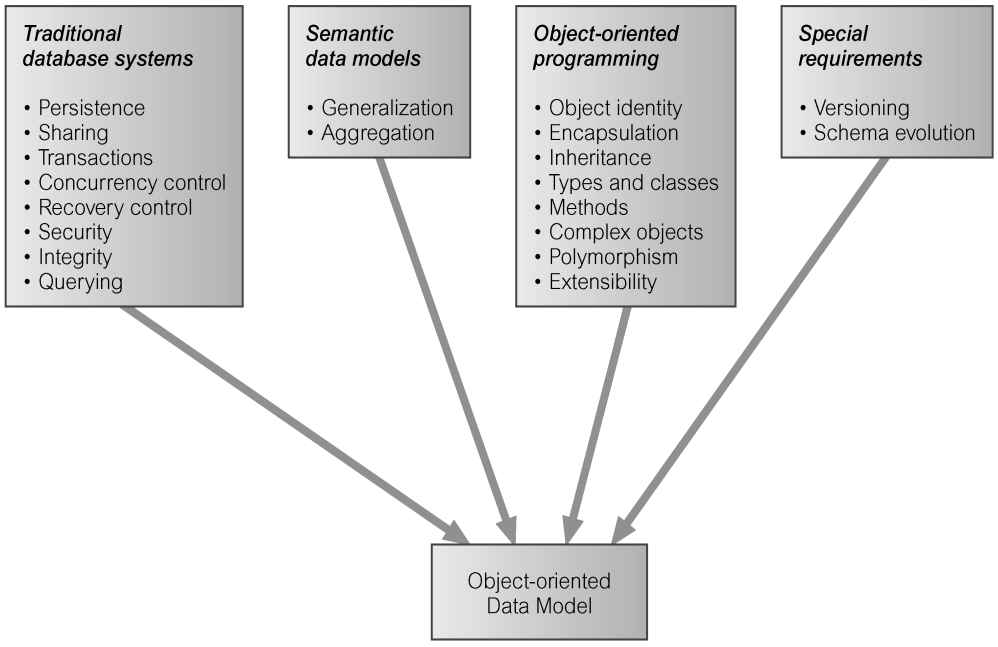
– Manager of an ODB.

An OODBMS must, at a minimum, satisfy:

– It must provide database functionality.
– It must support object identity.
– It must provide encapsulation.
– It must support objects with complex state.

- OODBMS viewed as:
  – OO = ADTs + Inheritance + Object identity
  – OODBMS = OO + Database capabilities.
- Parsaye *et al*. gives:
  – High-level query language with query optimization.
  – Support for persistence, atomic transactions: concurrency and recovery control.
  – Support for complex object storage, indexes, and access methods.
  – OODBMS = OO system + (1), (2), and (3).

Origins of the Object-Oriented Data Model



Alternative Strategies for Developing an OODBMS

- Extend existing object-oriented programming language.
  - GemStone extended Smalltalk.
- Provide extensible OODBMS library.
  - Approach taken by Ontos, Versant, and ObjectStore.
- Embed OODB language constructs in a conventional host language.
  - Approach taken by O2,which has extensions for C**.**

Object-Oriented Database Design

Table 25.3 Comparison of OODM and CDM.

OODM	CDM	Difference
Object	Entity	Object includes behavior
Attribute	Attribute	None
Association	Relationship	Associations are the same but inheritance in OODM includes both state and behavior
Message		No corresponding concept in CDM
Class	Entity type/Supertype	None
Instance	Entity	None
Encapsulation		No corresponding concept in CDM

Relationships

- **Relationships represented using reference attributes, typically implemented using OIDs.**
- **Consider how to represent following binary relationships according to their cardinality:**
  - **1:1**
  - **1:***
  - ***:*.**

1:1 Relationship Between Objects A and B

Add reference attribute to A and, to maintain referential integrity, reference attribute to B.

1:* Relationship Between Objects A and B

- Add reference attribute to B and attribute containing set of references to A.

***:* Relationship Between Objects A and B**

- Add attribute containing set of references to each object.

- For relational database design, would decompose *:N into two 1:* relationships linked by intermediate entity. Can also represent this model in an ODBMS.

Referential Integrity

Several techniques to handle referential integrity:

- Do not allow user to explicitly delete objects.
  - System is responsible for "garbage collection".
- Allow user to delete objects when they are no longer required.
  - System may detect invalid references automatically and set reference to NULL or disallow the deletion.

- Allow user to modify and delete objects and relationships when they are no longer required.
  - System automatically maintains the integrity of objects.
  - Inverse attributes can be used to maintain referential integrity.

Behavioral Design

- EER approach must be supported with technique that identifies behavior of each class.
- Involves identifying:
  - public methods: visible to all users
  - private methods: internal to class.
- Three types of methods:
  - constructors and destructors
  - access
  - transform

Methods:

- Constructor - creates new instance of class.
- Destructor - deletes class instance no longer required.
- Access - returns value of one or more attributes (Get).
- Transform - changes state of class instance (Put).

.

Identifying Methods

- Several methodologies for identifying methods, typically combine following approaches:
  - Identify classes and determine methods that may be usefully provided for each class.

– Decompose application in top-down fashion and determine methods required to provide required functionality.

OOPL (Object Oriented Programming Languages)

- Basically, an OODBMS is an object database that provides DBMS capabilities to objects that have been created using an object-oriented programming language (OOPL).
- The basic principle is to add persistence to objects and to make objects persistent.

- Consequently application programmers who use OODBMSs typically write programs in a native OOPL such as Java, C++ or Smalltalk, and the language has some kind of Persistent class, Database class, Database Interface, or Database API that provides DBMS functionality as, effectively, an extension of the OOPL.

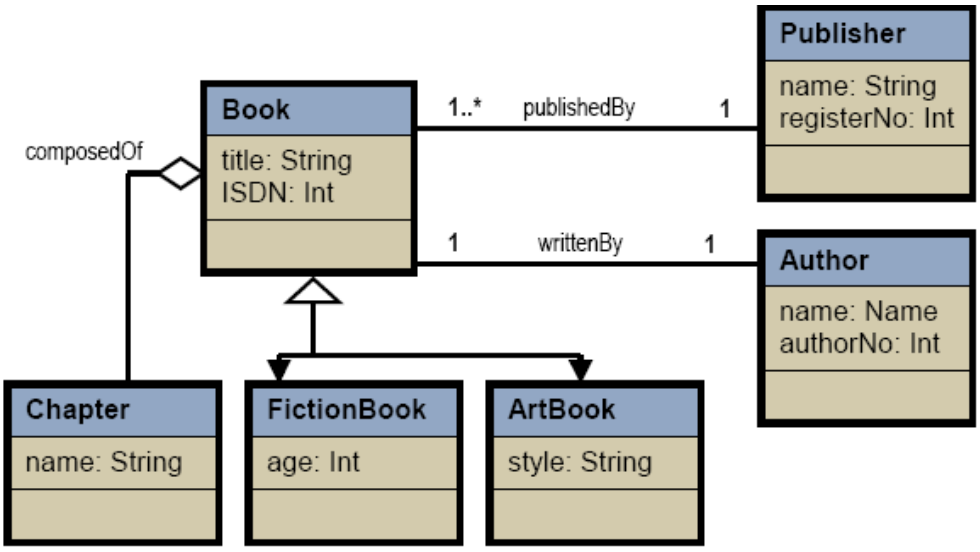
- Object-oriented DBMSs, however, go much beyond simply adding persistence to any one object-oriented programming language.

- This is because, historically, many object-oriented DBMSs were built to serve the market for computer-aided design/computer-aided manufacturing (CAD/CAM) applications in which features like fast navigational access,
- versions, and long transactions are extremely important.

- Object-oriented DBMSs, therefore, support advanced object-oriented database applications with features like support for persistent objects from more than one programming language, distribution of data, advanced transaction models, versions, schema evolution, and dynamic generation of new types.

Object data modeling

An object consists of three parts: structure (attribute, and relationship to other objects like aggregation, and association), behavior (a set of operations) and characteristic of types (generalization/serialization). An object is similar to an entity in ER model; therefore we begin with an example to demonstrate the structure and relationship.



The structure of an object `Book` is defined as following:

```
class Book {
      title: String;
      ISDN: Int;
      publishedBy: Publisher inverse publish;
      writtenBy: Author inverse write;
      chapterSet: Set<Chapter>;
}

class Author {
      name: String;
      authorNo: Int;
      write: Book inverse writtenBy;
}
```

Attributes are like the fields in a relational model. However in the Book example we have,for attributes publishedBy and writtenBy, complex types Publisher and Author,which are also objects. Attributes with complex objects, in RDNS, are usually other tableslinked by keys to the employee table.

Relationships: publish and writtenBy are associations with I:N and 1:1 relationship; composed_of is an aggregation (a Book is composed of chapters). The 1:N relationship is usually realized as attributes through complex types and at the behavioral level. For example,

```
class Publisher {
      …
      publish: Set<Book> inverse publishedBy;
      …
Method insert(Book book){
      publish.add(book);
}
```

Generalization/Serialization is the is_a relationship, which is supported in OODB through class hierarchy. An ArtBook is a Book, therefore the ArtBook class is a subclass of Book class. A subclass inherits all the attribute and method of its superclass.

```
class ArtBook extends Book {
      style: String;
}
```

Message: means by which objects communicate, and it is a request from one object to another to execute one of its methods. For example:
Publisher_object.insert (”Rose”, 123,…) i.e. request to execute the insert method on a Publisher object **)**

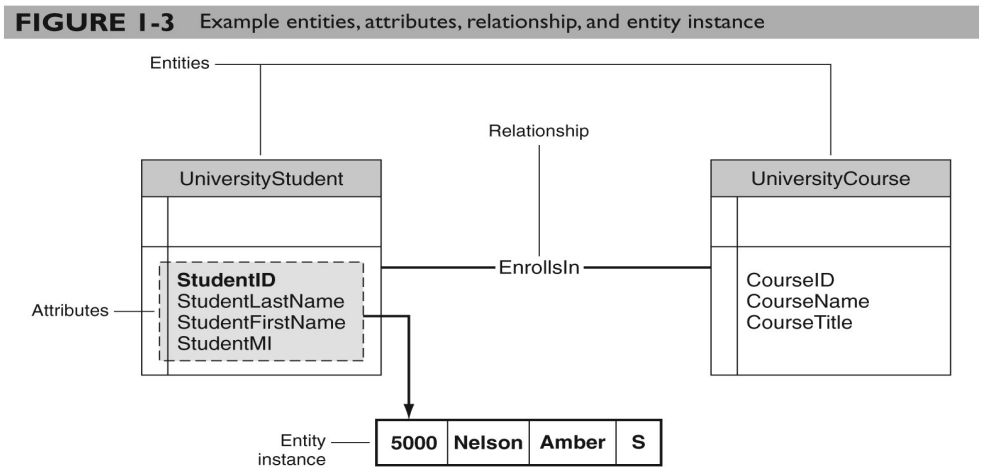
Method: defines the behavior of an object. Methods can be used
. to change state by modifying its attribute values . to query the value of selected attributes The method that responds to the message example is the method insert defied in the Publisher class.

The main differences between relational database design and object oriented database design include:

• Many-to-many relationships must be removed before entities can be translated into relations. Many-to-many relationships can be implemented directly in an object-oriented database.

• Operations are not represented in the relational data model. Operations are one of the main components in an object-oriented database.

• In the relational data model relationships are implemented by primary and foreign keys. In the object model objects communicate through their *interfaces*. The interface describes the data (attributes) and operations (methods) that are visible to other objects.



Relational Design Vs O-O Design

Relational Design	O-O Design
1. Identify entities/attributes	1. Identify objects/attributes
2. Resolve many-to-many relationships	Identify operations on objects
3. Translate entities into relations	Establish interface for each object.
4. Create primary/foreign key relationships.	4. Implement objects.
5. Implement realtions	

Topic – 4: Persistence

Persistent Object

The longest duration that an object supports the throughout the execution of applications is called persistency of that object.

Persistent Programming Languages

• Persistent Programming languages allow objects to be created and stored in a database, and useddirectly from a programming language
– allow data to be manipulated directly from the programming language
• No need to go through SQL.
– No need for explicit format (type) changes
• format changes are carried out transparently by system

• Without a persistent programming language, format changes becomes a burden on the programmer
– More code to be written
– More chance of bugs
- allow objects to be manipulated in-memory
• no need to explicitly load from or store to the database
– Saved code, and saved overhead of loading/storing large amounts of data

- **Drawbacks of persistent programming languages**
  - Due to power of most programming languages, it is easy to make programming errors that damage the database.
  - Complexity of languages makes automatic high-level optimization more difficult.
  - Do not support declarative querying as well as relational databases

Persistence of Objects
• Approaches to make transient objects persistent include establishing
– Persistence by Class – declare all objects of a class to be persistent; simple but inflexible.
– Persistence by Creation – extend the syntax for creating objects to specify that that an object is persistent.
  – Persistence by Marking – an object that is to persist beyond program execution is marked as persistent before program termination.

– Persistence by Reachability - declare (root) persistent objects; objects are persistent if they are referred to (directly or indirectly) from a root object.
  • Easier for programmer, but more overhead for database system
  • Similar to garbage collection used e.g. in Java, which also performs reachability tests

Storage and Access of Persistent Objects
How to find objects in the database:
• Name objects (as you would name files)
– Cannot scale to large number of objects.
– Typically given only to class extents and other collections of objects, but not objects.
• Expose object identifiers or persistent pointers to the objects
– Can be stored externally.
– All objects have object identifiers

Store collections of objects, and allow programs to iterate over the collections to find required objects

– Model collections of objects as **collection types**
– **Class extent** - the collection of all objects belonging to the class; usually maintained for all classes that can have persistent objects.

Persistent C++ Systems
☐ C++ language allows support for persistence to be added without changing the language
☐ Declare a class called Persistent_Object with attributes and methods to support persistence
☐ **Overloading** – ability to redefine standard function names and operators (i.e., +, –, the pointer deference operator –>) when applied to new types
☐ **Template classes** help to build a type-safe type system supporting collections and persistent types.
☐ Providing persistence without extending the C++ language is ☐ relatively easy to implement but more difficult to use
☐ Persistent C++ systems that add features to the C++ language have been built, as also systems that avoid changing the language.

Topic – 5: Query Languages

- Declarative query language
  - Not computationally complete
- Syntax based on SQL (select, from, where)
- Additional flexibility (queries with user defined operators and types)

Object Query Language (OQL)

A special type of Query structure is required to handle objects and its manipulation in OODB and is called OQL.

Example of OQL query

The following is the simple query:

What are the names of the black product?
Select distinct p.name From products p where p.color=”black”

- Valid in both SQL and OQL, but results are different.

Result of the query (SQL)

Original Table

Product no	Name	Color
P3	Mercedes SLK	Black
P2	Toyota Celica	Green
P1	Ford Mustang	Black

- The statement queries a relational database.
Ford Mustang => Returns a table with rows.
Result

Name
Mercedes SLK

SQL Query Vs OQL

Comparison

☐ Queries look very similar in SQL and OQL, sometimes they are the same.
☐ In fact, the results they give are very different
Query returns:

OQL	**SQL**
Object	Tuple
Collection of objects	Table

OOSQL:

SQL3 "Object-oriented SQL"
• Foundation for several OO database management systems – ORACLE8, DB2, etc
• New features – "relational" & "Object oriented"
• Relational Features – new data types, new predicates, enhanced semantics, additional security and an active database
• Object Oriented Features – support for functions and procedures

Basic concepts in O-O SQL:

User defined Data Types –

Creating a "row type"

Example:

```
create row type AddressType(
street char(50),
city char(20));
create row type StarType(
name char(30),
address AddressType);
```

Creating "Table"

```
create table Address of type AddressType;
create table MovieStar of type StarType;
```

Instances of Row types are tuples in tables

Sample Query

Find the names and street addresses of those MovieStars
who stay in the city "Columbus":

```
select MovieStar.name,
MovieStar.address.street
from MovieStar
where MovieStar.address.city = "Columbus";
```

Complex Data and Queries in O-O Query

A Water Resource Management example

- A database of state wide water projects
- Includes a library of picture slides
- Indexing according to predefined concepts – prohibitively expensive
- Type of queries
  - Geographic locations
  - Reservoir levels during droughts
  - Recent flood conditions, etc
- Addressing these queries
  - Linking this database to landmarks on a topographic map
  - Examining the captions for each slide
  - Implementing image-understanding programs

- Inspecting images and ascertaining attributes
- These type of queries necessitate dedicated "methods"

Creating Functions

```
create function one() returns int4
as 'select 1 as RESULT'
language 'sql';
        select one() as answer;
```

Answer
1

Creating "tables" with "methods"

Implementation

```
create table slides (
id int,
date date,
caption document,
picture CD_image,
method containsName
(name varchar)
returns boolean
as external name 'matching'
language 'C' );

create table landmarks(
name varchar (30),
location point);
```

Sample query – find a picture of a reservoir with low water level which is in "Sacramento"

```
select P.id
from slides P, landmarks L
where IsLowWaterLevel (P.picture) and
P.containsName (L.name) and L.name =
"Sacramento";
```

Recovery and the ACID properties

Atomicity: All actions in the transaction happen, or none happen.

Consistency: If each transaction is consistent, and the DB starts consistent, it ends up consistent.

Isolation: Execution of one transaction is isolated from that of other transactions.

Durability: If a transaction commits, its effects persist.

- The **Recovery Manager** is responsible for ensuring two important properties of transactions: ***Atomicity*** and ***Durability***.
- Atomicity is guaranteed by undoing the actions of the transactions that did not commit.
- Durability is guaranteed by making sure that all actions of committed transactions survive crashes and failures.

Topic – 6: Transaction

Basic Concepts

***Data Items*—**collection of objects representing a database

• ***Granularity*—**size of a data item

• ***Concurrency***—multiple users accessing a database instance at the same time

• ***Transaction***—*a* logical unit of database processing that includes one or more database access operations

– *Insert, Delete, Modify, Retrieve operations*

• ***Serializability***—Interleaving execution of a set of concurrent transactions without "giving up any correctness"

Provide Transactions that have ACID properties:

– ***A***tomicity—a transaction is an atomic unit of work; it's performed in its entirety or not at all

– ***C***onsistency preservation—a transaction takes database from one consistent state to another

– ***I***solation—a transaction should appear as though it is being executed in isolation from other transactions

- ***D***urability or permanency—the changes applied to the database by a committed transaction must persist in the database. The changes must not be lost because of any failure

Concurrency and Transaction Management:
OODBM's vs. RDBM's
• Both DBMS's must deal with Concurrency and Transaction Management issues
• Many concurrency protocols can be applied to both DBMS's
– Optimistic and Pessimistic protocols are relevant to both
• However, semantically different:
– Example: *Data Item Granularity*
• In traditional RDBMS, fine granularity data item would be a record field value of a record
• In an OODBMS, fine granularity data item may be an Object or data member (field) of an Object

Many OODB's…Varying Frameworks
• There are many OODBM's existing and emerging in both commercial and open source areas
• Implementations vary differently
– Distributed database model
– Centralized database model
– "hybrid" implementation, such as *Object-Relational Databases*
• Use Relational DBMS Engine
• Use Structured Query Language (SQL) or an extension of it for providing access to "Objects"
• Currently, there is no *consensus* or clear specification for an OODMS as there was for a Relational DBMS (such as Codd's original specification for a relational data model and query language)

Topic – 7: Concurrency

Concurrency Control Protocols—set of rules for
defining the execution of concurrent transactions
(ultimately to ensure **serializability**)
– Optimistic Concurrency Control—validation or certification of a transaction AFTER it executes
• If interference is detected, the transaction is aborted and restarted at a later time
Pessimistic Concurrency Control—Locks are used
to prevent conflicting transactions
• 2-Phase Locking Protocol (2PL): Best known locking
protocol for guaranteeing serializability
– Phase 1: Expanding/Growing. New locks can be acquired but none can be released
– Phase 2: Shrinking. Existing locks can be released but no new locks can be acquired

» Strict 2PL—a transaction does not release any of its exclusive (write) locks until after it commits or aborts

Database Management Systems Should…

• Provide *Concurrency Control*
– The DBMS should allow more than one user to access/manipulate data concurrently
– When there is concurrency, T*ransaction Management* **must** be addressed
• **The Lost Update Problem**—two transactions have their operations interleaved in such a way that some database item is incorrect—inconsistent state!
• **The Temporary Update (Dirty Read) Problem**—One transaction updates a database item and then the transaction fails; the updated item is access by another transaction before it is changed back to its original value

Topic – 8: Multi Version Locks

Multiversion concurrency control (abbreviated **MCC** or **MVCC**), in the database field of computer science, is a concurrency control method commonly used by database management systems to provide concurrent access to the database and in programming languages to implement transactional memory[1].

For instance, a database will implement updates not by deleting an old piece of data and overwriting it with a new one, but instead by marking the old data as obsolete and adding the newer "version."

Thus there are multiple versions stored, but only one is the latest. This allows the database to avoid overhead of filling in holes in memory or disk structures but requires (generally) the system to periodically sweep through and delete the old, obsolete data objects.

For a document-oriented database such as CouchDB, Riak or MarkLogic Server it also allows the system to optimize documents by writing entire documents onto contiguous sections of disk—when updated, the entire document can be re-written rather than bits and pieces cut out or maintained in a linked, non-contiguous database structure.

MVCC also provides potential "point in time" consistent views. In fact read transactions under MVCC typically use a timestamp or transaction ID to determine what state of the DB to read, and read these "versions" of the data.

This avoids managing locks for read transactions because writes can be isolated by virtue of the old versions being maintained, rather than through a process of locks or mutexes. Writes affect future "version" but at the transaction ID that the read is working at, everything is guaranteed to be consistent because the writes are occurring at a later transaction ID.

In other words, MVCC provides each user connected to the database with a "snapshot" of the database for that person to work with. Any changes made will not be seen by other users of the database until the transaction has been committed MVCC uses timestamps or increasing transaction IDs to achieve transactional consistency.

MVCC ensures a transaction never has to wait for a database object by maintaining several versions of an object. Each version would have a write timestamp and it would let a transaction (T_i) read the most recent version of an object which precedes the transaction timestamp ($TS(T_i)$).

If a transaction (T_i) wants to write to an object, and if there is another transaction (T_k), the timestamp of T_i must precede the timestamp of T_k (i.e., $TS(T_i) < TS(T_k)$) for the object write operation to succeed. Which is to say a write cannot complete if there are outstanding transactions with an earlier timestamp.

Every object would also have a read timestamp, and if a transaction T_i wanted to write to object P, and the timestamp of that transaction is earlier than the object's read timestamp ($TS(T_i) < RTS(P)$), the transaction T_i is aborted and restarted.

Otherwise, T_i creates a new version of P and sets the read/write timestamps of P to the timestamp of the transaction $TS(T_i)$.
The obvious drawback to this system is the cost of storing multiple versions of objects in the database. On the other hand reads are never blocked, which can be important for workloads mostly involving reading values from the database.

MVCC is particularly adept at implementing true snapshot isolation, something which other methods of concurrency control frequently do either incompletely or with high performance costs.

At t1 the state of a DB could be

Time	Object 1	Object 2
t1	"Hello"	"Bar"
t0	"Foo"	"Bar"

This indicates that the current set of this database (perhaps a key-value store database) is Object1="Hello", Object2="Bar". Previously, Object1 was "Foo" but

that value has been superseded. It is not deleted because the database holds "multiple versions" but will be deleted later.

If a long running transaction starts a read operation, it will operate at transaction "t1" and see this state.

If there is a concurrent update (during that long-running read transaction) which deletes Object 2 and adds Object 3 = "foo-bar" the database state will look like:

Time	Object 1	Object 2	Object 3
t2	"Hello"	(deleted)	"Foo-Bar"
t1	"Hello"	"Bar"	
t0	"Foo"	"Bar"	

Now there is a new version as of transaction ID t2. Note, critically, that the long-running read transaction *still has access to a coherent snapshot of the system at t1* even though the write transaction added data as of t2, so the read transaction is able to run in isolation from the update transaction that created the t2 values.

This is how MVCC allows isolated, ACID, reads without any locks (the write transaction does need to use locks).

Topic – 9:Recovery

To guarantee Atomicity and Durability, Abort/Rollbacks, System Crashes etc..

Reasons for crashes

Transaction failures: logical errors, deadlocks
System crash: power failures, operating system bugs etc
Disk failure: head crashes
We will assume STABLE STORAGE for now Data is not lost
Typically ensured through redundancy (e.g. RAID)

STEAL:

The buffer manager can steal a memory page for replacement purposes
The page might contain dirty writes

FORCE:

Before committing a transaction, force its updates to disk
Easiest option: NO STEAL, FORCE

NO STEAL, so atomicity easier to guarantee
No serious durability issues because of FORCE Issues:
How to force all updates to disk atomically ? Can use shadow copying.
A page might contain updates of two transactions ? Can use page level locking etc.

Desired option: STEAL, NO FORCE
STEAL:
Dirty data might be written on disk
Need to use UNDO logs so we can rollback that action
The UNDO log records must be on disk before the page can be written (Write-Ahead Logging)
NO FORCE:
Data from committed transaction might not make it to disk
Use REDO logs
The REDO log records must make it disk before the transaction is "committed"

Simple Log-based Recovery:

Each action generates a log record (before/after copies)
Write Ahead Logging: Log records make it to disk before corresponding data page
Strict Two-Phase Locking
Locks held till the end
Once a lock is released, not possible to undo

Normal Processing: UNDO (rollback)
Go backwards in the log, and restore the updates
Locks are already there, so not a problem

Normal Processing: Checkpoints
Halt the processing
Dump dirty pages to disk
Log: (checkpoint list-of-active-transactions)

Database Recovery and Security

Motivation and Assumptions

- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical database Security

Recovery and ACID properties

Atomicity: All actions in the transaction happen, or none happen.

Consistency: If each transaction is consistent, and the DB starts consistent, it ends up consistent.

Isolation: Execution of one transaction is isolated from that of other transactions.

Durability: IF a transaction commits, its effects persist.
The recovery manager is responsible for ensuring two important properties of transactions: Atomicity and Durability
Atomicity is guaranteed by making sure that all actions of committed transactions survive crashes and failures.

C & I: Concurrency control; A & D: Recovery mechanisms.

Types of Failures

- **System crashes**: due to hardware or software errors, resulting in loss of main memory
- **Media failures**: due to problems with disk head or unreadable media, resulting in loss of parts of secondary storage.
- **Application errors**: due to logical errors in the program which may cause transactions to fail.
- **Natural disasters**: physical loss of media (fire, flood, earthquakes, terrorism, etc.)
- **Sabotage**: intentional corruption or destruction of data
- **Carelessness**: unintentional destruction of data by user or operator.

Sample Questions

Topic – 1:

1. What is Object Oriented Database? (OODB) (2M)
2. What are the advantages of OODBS? (2M)
3. List any four object Database Vendors. (2M)
4. What are Database Structures? (2M)
5. Explain in detail about Hierarchical Databases. Give example. (8M)
6. Explain in detail about Relational Databases. Give example. (8M)
7. Explain in detail about Object-Oriented Databases Give example.(8M)
8. What is an Object ? (2M)
9. Define Object instance (2M)
10. What is Object class ? (2M)
11. Explain briefly on Object identity (2M)
12. What are Object Structures? (2M)
13. Explain the following O-O concepts briefly: (16M)
    Abstract Data Types
    Encapsulation
    Inheritance
    Polymorphism
14. Explain briefly on Class Hierarchy. (8M)
15. List Some of the OODBMS's. (2M)
16. What are the Advantages and disadvantages of OODBMSs? (8M)
17. Differences between OODB and Relational DB. (8M)

Topic – 2:

1. What is the need of OODB approaches? (2M)
2. Explain in detail on main four approaches of OODB. (16M)

Topic – 3:

1. Explain briefly on Object-Oriented Data Model. (8M)
2. Discuss about the Origins of the Object-Oriented Data Model. (8M)
3. List the Alternative Strategies for Developing an OODBMS (2M)
4. Explain in detail about Object-Oriented Database Design. (16M)
5. What are Relationships? (2M)
6. What is Referential Integrity? Explain briefly with an example. (8M)
7. What are the Several techniques to handle referential integrity (4M)
8. How a Behavioral Design for OODB is achieved. ? Explain concepts associated with it. (8M)
9. Explain briefly on Object data modeling. (8M)
10. Differentiate between Relational Design and O-O Design (4M)

Topic – 4:

1. What are Persistent Object ? How they are useful in OODB Modeling and design? (8M)
2. Describe briefly on Persistent Programming Languages. (8M)
3. List out the drawbacks of persistent programming languages. (2M)
4. How Storage and Access of Persistent Objects is useful for OODB design? Explain. (8M)
5. How Persistent C++ Systems are helpful in OODB design and implementations.? Explain (8M)

Topic – 5:

1. What are Object Query Language (OQL)? (2M)
2. What are the major differences between SQL Query and OQL? (2M)
3. Explain briefly on OOSQL and its basic concepts. (8M)
4. Explain *Complex Data and Queries in O-O Query (8M)*

Topic – 6:

1. What is Transaction ? Explain with an example. (2M)
2. Explain that Provide Transactions that have ACID properties. (8M)
3. What are the basic concepts in Transactions. (4M)
4. Explain briefly on Concurrency and Transaction Management (4M)
5. Compare and contrast between OODBM's and RDBM's. (4M)
6. Discuss how Many OODB's are having Varying Frameworks. (8M)

Topic – 7:

1. What is concurrency? (2M)
2. Explain briefly on Concurrency Control Protocols (4M)
3. What are the mechanisms needed Database Management Systems with respect to concurrency? (8M)

Topic – 8:

1. What are multivesrion locks? (2M)
2. Explain in details on Multiversion Locks. Give examples. (16M)

Topic – 9:

1. What is recovery? How it differs from failure? (2M)
2. List the different types of Failures. (2M).
3. List the reasons for crashes. (2M)
4. Explain in detail on Simple Log-based Recovery. (8M)
5. Explain on Recovery and ACID properties related to Recovery. (8M)

University Questions

1. Explain the need for object oriented database. (2M)
2. What are the various approaches of persistence of objects? (2M)
3. How does the concept of an object in the object oriented model differ from the concept of an entity in the entity-relationship model? (8M)
4. Explain the concept of multi-version locks in object oriented databases. (8M)
5. Explain recovery techniques based on immediate update (8M).
6. Explain in detail ACID properties of transactions. (16M)

********** End of Unit – II **********