Topics
- Database Design
- Design for Object Databases
  - Design for Object-Relational Databases
  - Design for Relational Databases

Persistent database layer
- Relational
  - Oracle, Sybase, DB2, SQL Server
- Object-relational
  - Oracle8, UniSQL
- Object
  - ObjectStore, Versant
- Older models
  - Hierarchical (IMS)
  - Network (IDMS)
  - Inverted or similar (Total, Adabas)
- Flat file
**Data models**

- Abstraction that presents the database structures in more understandable terms than as raw bits and bytes
- **External (conceptual) data model**
  - a high-level conceptual data model required by a single application
- **Logical (global conceptual) data model**
  - a global integrated model to support any current and expected applications that need access to the information stored in the database
- **Physical data model**
  - defines how data is actually stored on persistent storage devices

**Application model & persistent model**

**Mapping objects to database**

- **Mapping difficulties**
  - Storage structures of the database may have little to do with the object-oriented paradigm
    - Requires the conversion of classes in the Entity Package to **non object-oriented structures**
  - Database is almost never designed for a single application
    - Requires an optimal database **design for all applications**, not just the one under consideration
ODB model

- ODMG standards
- The ODMG 3.0 standard (January 2000) – mapping objects to relational and other databases
  - Object Storage API that
    - works with any persistent data source
    - implements the Database Package
- ODBMS
  - May not provide a separate DML language (such as SQL) within a PL environment – PL extended with database objects, including navigational queries.
  - However, Object SQL (OSQL) to access DB interactively

ODB modeling primitives

- Object has an OID
- Literal has no OID – its value is like its identifier
- Distinction between class (implementation) and type (specification)
  - Type can have multiple classes
- Class has properties and operations
  - Property can be an attribute or a relationship

ODB literal types

- Atomic (simple)
  - Numeric
  - Alphanumeric and special characters
- Structured
  - Date
  - Time
- Collection (template) – parameterized type
  - Set<t>
  - List<t>
- Null
  - Specifies that a NULL value is allowed in an atomic, structured or collection literal
**ODB object types**

- **Atomic object**
  - e.g. DeptName

- **Structured object**
  - Contains component object and/or literal types
  - e.g. Dept

- **Collection**
  - The same set of possibilities as for the collection literal but with parameters taking on object values
  - e.g. set<Dept>

**Type declarations in ODB**

```plaintext
<<ODB>>
EmpLOYEE

emp_id : string
date_of_birth : date
gender : enum{M,F}
phone_num : array<string>
salary : float
```

**emp_name** is structured object if the class PersonName consists of attributes such as
family_name, first_name, middle_initial

Other attributes in Employee are literal types

**Relationships and inverses in ODB**

- **ODB model supports**
  - **Association**
    - Implemented with collection object types, in particular Set<> and List<>.
    - The inverse keyword, explicit in the schema definition, enforces the referential integrity and eliminates the possibility of dangling pointers.
  - **Generalization**
  - **Aggregation** is only supported by constraining an association.
  - **Keys** — unique identifying values for objects of the class.
    - However, key is not the principal way of object identification (the OID values serve this purpose).
    - Key can be simple or compound.
Associations in ODB

```
class Student {
  attribute string name;
  attribute string stud_id;
  relationship Set<CourseOffering> crs_off;
  inverse CourseOffering::std;
};

class CourseOffering {
  attribute string crs_name;
  attribute string semester;
  relationship List<Student> std;
  inverse Student::crs_off;
};
```

ISA and EXTENDS inheritance

- ISA relationship = interface inheritance
- EXTENDS relationship = implementation inheritance

The ODMG standard uses:
- Keyword `interface` when defining an abstract class
- Keyword `class` is used only for the definition of concrete classes that can be instantiated

Inheritance in ODB
In UML modeling:
- Attributes in classes are defined on atomic data types and on few built-in structured data types (Data, Currency).
- Association roles imply the collections (templates).

But what about questions like:
- "What if an employee has many phone numbers? How should I model this during analysis? Do I really need to have a separate class of phone numbers?"
- "Can I model an employee name as a single attribute but with the internal structure recognizing that the name consists of the family name, first name and middle initial? Do I really need to have a separate class of employee names?"

To ODB:
```
Employee
<<key>>
employee_id : string
employee_name : PersonLongName

PersonLongName
middle_initial : char

Contact
<<key>>
contact_id : unsigned short
contact_name : PersonShortName
phone : set<string>
fax : set<string>
email : set<string>

Not needed, just to visualize the issue
```

From UML associations...
From UML aggregations ...

... To ODB
From UML generalizations …

… To ODB

ORDB Model

- ANSI and ISO standards
- SQL:1999 standard
  - Many ORDB issues unresolved
  - Expected to be revised roughly every three years
- The “next great wave”
- Upward compatible with RDB
  - Capable of processing relational data structures (relational tables) and object data structures (object tables)
  - User-defined types extended with arbitrary complex structured types
- Migration from RDB to ORDB not really addressed by the standard
ORDB modeling primitives

- **Structured type** ~ ODB interface ~ UML class
- **Multiple interface inheritance**
- **Table** ~ the storage mechanism
- **Object table** ~ has columns defined as structured types

**Row type**
- To specify complex internal structure without even using structured types
- Row type column implies **nested table**

ORDB types

- **User-defined types**
  - Distinct type ~ ODB atomic object type
  - Structured type ~ ODB structured object type
    - Attributes (state)
    - Operations (behavior)
    - Operations to define equality/inequality, ordering, and conversion of structured type objects

- **Structured type attribute can be a collection**
  - Set
  - List
  - Multiset
  - Array

Type declarations in ORDB

```plaintext
<<structured type>>
EmployeeTY

emp_id : char(7)
emp_name : PersonName
date_of_birth : date
gender : char
phone_num : set(varchar(12))
salary : money
```
ORDB object tables

- **Table** (object table) - a set of rows having one or more columns
- **Object** - a row in object table (the smallest unit of data that can be inserted or deleted from a table)
- To store objects (instances of a type), a table has to be created
- create table Employee of EmployeeTY;
- SQL:1999 has not defined **encapsulation** of attributes of a structured type by operations
- **observer** (get) and **mutator** (set) operations generated automatically by ORDB for each attribute

ORDB row types

- **Sequence of fields** (<field name><data type> pairs)
- Column in a table can contain row values

create table Contact
  | contact_id integer,
  | contact_name row
  | [family_name varchar(30),
  | first_name varchar(20)],
  | postal_address row
  | (po_box varchar(10),
  | post_code varchar(10),
  | address row
  | (street varchar(30),
  | city varchar(20),
  | state varchar(20),
  | country varchar(25)));

ORDB reference types

- **Keyword ref** is used to define references
- create ref(EmployeeTY)
- In SQL:1999, reference type must be known at compilation type (i.e. dynamic classification not supported)
- Value of a reference type is an **OID**
- **Collections of references** - to implement many-to-many associations
ORDB columns, fields, attributes

- **Column** - structural component of a table
  - Can be **nullable**
  - Can be an **identity** column (taking OID values)
  - Can be a **reference** type

- **Field** - structural component of a row type
  - Can be a **reference** type

- **Attribute** - structural component of a structured type
  - Can be a **reference** type

ORDB OF and UNDER inheritance

- Single implementation inheritance
- Table hierarchy can correspond to type hierarchy (but a type can be "skipped" in table hierarchy)

Mapping entity classes to ORDB

- Row type is NOT an OO concept!
- Dependency relationship

MACIASZEK (2001): Req Analysis & Syst Design
RDB model

- ANSI and ISO standards
- SQL92 – the current (and the last) standard
- Replaced hierarchical and network database models
- Will be replaced by ORDB model
- ODB model provides Object Storage API to interface applications to RDB

RDB modeling primitives

- Relational table
  - Columns with **atomic values** only
- RDB model does not support:
  - Object types and associated concepts (inheritance, methods, etc.)
  - Structured types
  - Collections
  - References

RDB modeling primitives
RDB columns, domains, rules

- **Column** – atomic data types
- **Domain** – legal set of values for a column
  
  ```
  create domain Gender char(1);
  ```

- **Rule** can define
  
  - default value (e.g., if no value is provided for city, assume “Sydney”)
  - range of values (e.g., the allowed age is in the range 18 to 80)
  - list of values (e.g., the allowed color is “green”, “yellow” or “red”)
  - case of value (e.g., the value must be in uppercase or lowercase)
  - format of value (e.g., the value must start with letter “K”)

RDB relational tables

- **Fixed set of columns**
- **Any number of rows**
- **Mathematical set** → no duplicate rows → key
  
  - **Primary key**
  - **Candidate (alternate) keys**

```sql
<table>
<thead>
<tr>
<th>Employee</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>emp_id</td>
<td>char(5)</td>
<td>not null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>family_name</td>
<td>VARCHAR(50)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>first_name</td>
<td>VARCHAR(50)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>date_of_birth</td>
<td>DATE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gender</td>
<td>char(1)</td>
<td>not null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phone_num1</td>
<td>VARCHAR(12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>phone_num2</td>
<td>VARCHAR(12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>salary</td>
<td>DEC(8,2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Referential integrity

- **Primary to foreign key correspondence**
- **Primary and foreign keys defined on the same domain**
- **Declarative or procedural (triggers)**
Referential integrity

Triggers

create trigger keepdpt
on Department
for delete
as
if @@rowcount = 0
    return /* avoid firing trigger if no rows affected */
    
    if exists
    (select * from Employee, deleted
    where Employee.dept_id =
    deleted.dept_id)
    begin
        print "Test for RESTRICT DELETE failed. No deletion"
        rollback transaction
        return
    end
    return

Mapping entity classes to RDB
Mapping associations to RDB

Mapping aggregations to RDB

From UML generalization ...

(c) Addison Wesley Chapter 8
To RDB model

Map each class to a table

- Person
  - person_id
  - name

- Employee
  - employee_id
  - person_id

- Student
  - student_id

StudentEmployee
- employee_id
- student_id

To RDB model

Map the entire class hierarchy to a single "superclass" table

<table>
<thead>
<tr>
<th>Person</th>
<th>person_id</th>
<th>unique_identifier</th>
<th>not null</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>is_employee</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is_student</td>
<td>null</td>
</tr>
</tbody>
</table>

To RDB model

Map each concrete class to a table

- Employee
  - employee_id

- Student
  - student_id

StudentEmployee
- employee_id
- student_id
... To RDB model

- Map each disjoint concrete class to a table

```
Employee
employee_id (CHAR(10)) not null
is_student BOOLEAN not null

Student
student_id (CHAR(10)) not null
is_employee BOOLEAN not null
```

Mapping generalizations to RDB

```
BaseTable
base_code CHAR(10) not null
base_name VARCHAR(50) not null

MiddleTable
mid_code CHAR(10) not null
mid_name VARCHAR(50) not null

 DerivedTable
der_code CHAR(10) not null
der_name VARCHAR(50) not null
```

Summary

- Three levels of database models – external, logical and physical
- Mapping of objects to databases = mapping of a UML class model to a database logical model
- The mapping to the ODB logical model is the easiest
- The ORDB logical model is more complex than the ODB model
- The mapping to the RDB logical model is the most cumbersome