Chapter 8  

Database Design

Copyright © 2000 by Addison Wesley  
Version 1.0

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
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\(<\text{Dept}>\)
Type declarations in ODB

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

<<ODB>>

Employee

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

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

```java
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 (Date, 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

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

```
<<ODB Interface>>
PersonShortName
family_name : string
first_name : string
```

```
<<ODB Interface>>
PersonLongName
middle_initial : char
```

```
<<ODB Class>>
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 ...

```
0..1
Organization
<<PK>> organization_id : Integer
generated by SequenceGenerator
organization_name : String
tax : String
email : String
```

```
0..1
theOrganization
```

```
0..1
Contact
<<PK>> contact_id : Integer
generated by SequenceGenerator
family_name : String
tax : String
e-mail : String
```

```
1
theOrganization
```

```
0..* task
```

```
0..* description : String
```

```
0..* startDate : Date
```

```
0..* endDate : Date
```

```
0..1
created
```

```
0..1
```

```
0..1
due
```

```
0..1
```

```
0..1
```

```
0..1
```

```
0..1
```

```
0..1
done
```

```
0..1
```

```
0..1
```

MACIASZEK (2001): Req Analysis & Syst Design
... To ODB

**<<ODB Class>>**
Organization
- organization_id : unsigned short
- phone : set<string>
- fax : set<string>
- email : string
- p_current : boolean
- postal_address : PostalAddress
- courier_address : CourierAddress
- contact : Set<Contact>
- task : Set<Task>

**<<ODB Class>>**
Task
- description : string
- created_dt : date
- value : float
- created_emp : Employee
- event : List<Event>
- organization : Organization
- contact : Set<Contact>

**<<ODB Class>>**
Contact
- contact_id : unsigned short
- contact_name : PersonShortName
- phone : set<string>
- fax : set<string>
- email : set<string>
- postal_address : PostalAddress
- courier_address : CourierAddress
- organization : Organization
- task : Set<Task>

**<<ODB Class>>**
Employee
- employee_id : string
- employee_name : PersonLongName
- created_task : List<Task>
- created_event : Set<Event>
- due_event : List<Event>
- completed_event : Set<Event>
- priority : char

From UML aggregations ...

**<<PK>>**
Student
- student_id : String
- student_name : String
- current_fees : Money

**<<PK>>**
Course
- course_code : String

**<<CK>>**
CourseOffering
- year : Date
- semester : Integer
- enrolment_quota : Integer

AcademicRecord
- course_code : String
- year : Date
- semester : Integer
- grade : String

AcademicInCharge

0..*

0..*

0..1

takes

takes_crscoff

0..*

0..1
... To ODB

<<ODB Interface>>
YearSemester
  year : date
  semester : unsigned short

<<ODB Interface>>
AcademicRecord
  course_code : string
  year_sem : YearSemester
  grade : string

<<ODB Class>>
Student
  student_id : string
  student_name : string
  current_fees : float
  course_off : list<CourseOffering>
  academic_record : set<AcademicRecord>

<<ODB Class>>
Course
  course_code : string
  course_name : string
  credit_points : unsigned short
  course_offering : list<CourseOffering>

<<ODB Class>>
CourseOffering
  year_sem : YearSemester
  enrolment_quota : unsigned short
  course : Course
  student : list<Student>
  academic_in_charge : AcademicInCharge

<<ODB Class>>
AcademicInCharge
  course_off : list<CourseOffering>

From UML generalizations ...

<<PK>> MovieTitle
  movie_code : String
  movie_title : String
  director : String
  is_in_stock : Boolean

VideoMedium
  video_condition : Byte
  $number_currently_available : Integer

VideoTape
  is_taped_over : Boolean

BetaTape

VHSTape

VideoDisk
different_languages : Boolean
different_endings : Boolean

DVDDisk

RentalConditions
  rental_period_in_days : Integer
  rental_charge_per_period : Currency
... To ODB

```sql
<<ODB Interface>>
RentalConditions
rental_period_in_days : unsigned short
rental_charge_per_period : float

<<ODB Interface>>
VideoMedium
video_condition : char
number_currently_available : unsigned short
rentalcond : RentalConditions
movie_title : MovieTitle

<<ODB Class>>
MovieTitle
<<key>>
movie_code : string
movie_title : string
director : string
is_in_stock : boolean
video_medium : set<VideoMedium>

<<ODB Interface>>
VideoTape
is_taped_over : boolean

<<ODB Interface>>
VideoDisk
different_languages : boolean
different_endings : boolean

<<ODB Class>>
BetaTape

<<ODB Class>>
VHSTape

<<ODB Class>>
DVDDisk
```

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

<<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

```sql
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**
  ```sql
  emp 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

```sql
create table Contact of ContactTY
  (person_short_name row
   family_name varchar(30)
   first_name varchar(20)));
```

```sql
EmployeeTY
  employee_id : char(8)
  employee_name : PersonLongNameTY
<<structured type>>

PersonLongNameTY
  family_name : varchar(30)
  first_name : varchar(20)
  middle_initial : char
<<structured type>>

ContactTY
  contact_id : integer
  phone : set(varchar(15))
  fax : set(varchar(15))
  email : set(varchar(50))
<<structured type>>

Employee
<<object table>>

Contact
<<object table>>

Row type is NOT an OO concept!

Dependency relationship

Mapping associations to ORDB

```sql
create table Contact of ContactTY
  (person_short_name row
   family_name varchar(30)
   first_name varchar(20)));
```

```sql
Dependency relationship
```

```sql
OrganizationTY
  organization_id : integer
  phone : set(varchar(15))
  fax : set(varchar(15))
  email : varchar(30)
  is_current : boolean
<<structured type>>

Organization
<<object table>>

TaskTY
  description : varchar(255)
  start_date : date
  due_date : date
  value : money
<<structured type>>

Task
<<object table>>

EmployeeTY
  employee_id : char(8)
  employee_name : PersonLongNameTY
<<structured type>>

PersonLongNameTY
  family_name : varchar(30)
  first_name : varchar(20)
  middle_initial : char
<<structured type>>

Employee
<<object table>>

Contact
<<object table>>

Organization
<<object table>>

```

Dependency relationship

MACIASZEK (2001): Req Analysis & Syst Design
Mapping aggregations to ORDB

```
<<structured type>>
AcademicRecordTY
  course_code : char(7)
  year : date
  credit_code : integer

<<structured type>>
CourseOfferingTY
  course : ref(Course)

<structured type>
Student
  student_id : char(5)
  student_name : varchar(50)
  current_fees : money
  academic_record : ref(AcademicRecordTY)

<<object table>>
CourseOffering
  year : date
  semester : integer
  enrollment_data : integer
```

Mapping generalizations to ORDB

```
<<structured type>>
MovieTitleTY
  movie_code : char(10)
  movie_title : varchar(100)
  director : varchar(30)

<<structured type>>
RentalConditionsTY
  rental_period_in_days : integer
  rental_charge_per_period : money

<<structured type>>
VideoMediumTY
  video_condition : char
  rental_code : ref(RentalConditionsTY)

<<structured type>>
VideoTapeTY
  is_taped_over : boolean

<structured type>
BetaTape
  movie_title : ref(MovieTitle)

<structured type>
VHSTape
  movie_title : ref(MovieTitle)

<structured type>
DVDDisk
  movie_title : ref(MovieTitle)

<<structured type>>
VideoDiskTY
  different_languages : boolean
  different_endings : boolean
```
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

- **Referential Integrity**
- **View**
- **Table**
- **Index**
- **Column**
- **Key**
- **Rule**
- **Domain**
- **Trigger**
- **Stored procedure**

RDB columns, domains, rules

- **Column** – atomic data types
- **Domain** – legal set of values for a column

```sql
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

<table>
<thead>
<tr>
<th>Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp_id</td>
</tr>
<tr>
<td>family_name</td>
</tr>
<tr>
<td>first_name</td>
</tr>
<tr>
<td>date_of_birth</td>
</tr>
<tr>
<td>gender</td>
</tr>
<tr>
<td>phone_num1</td>
</tr>
<tr>
<td>phone_num2</td>
</tr>
<tr>
<td>salary</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

```sql
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
```

Triggers
Mapping entity classes to RDB

Contact
- contact_id: SMALLINT, <sp> not null
- contact_name: VARCHAR(20), not null
- fax: VARCHAR(15), null
- email: VARCHAR(20), null

Employee
- employee_id: CHAR(8), <sp> not null
- family_name: VARCHAR(20), not null
- first_name: VARCHAR(20), not null
- middle_initial: CHAR(1), null

ContactPhone
- phone: VARCHAR(15), <sp> not null
- contact_id: SMALLINT, <sp> not null

Mapping associations to RDB

Task
- task_id: SMALLINT
- description: VARCHAR(250)
- due_date: DATE
- priority: SMALLINT

Event
- event_id: INT32
- task_id: SMALLINT
- employee_id: CHAR(8)
- completed: CHAR(8)
- description: VARCHAR(250)
- due_date: DATE
- completed: SMALLINT

Employee
- employee_id: INT32
- family_name: VARCHAR(20)
- first_name: VARCHAR(20)
- middle_initial: CHAR(1)
Mapping aggregations to RDB

From UML generalization ...
... To RDB model

- Map each class to a table

```
Person
person_id <pk> person_id = person_id

Employee
employee_id <pk>
person_id <fk>
employee_id = employee_id

Student
student_id <pk>
person_id <fk>
student_id = student_id

StudentEmployee
employee_id <pk, fk1>
student_id <pk, fk2>
```

... To RDB model

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

```
Person
person_id uniqueidentifier <pk> not null
is_employee char(1) null
is_student char(1) null
```
... To RDB model

- Map each concrete class to a table

Employee

- employee_id <pk>

Student

- student_id <pk>

StudentEmployee

- employee_id <pk>1
- student_id <pk>2

employee_id = employee_id

... To RDB model

- Map each disjoint concrete class to a table

Employee

- employee_id NCHAR(8) <pk> not null
- is_student BOOLEAN not null

Student

- student_id NCHAR(10) <pk> not null
- is_employee BOOLEAN 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