Object-Relational Model

Oracle Link: http://docs.oracle.com/cd/B19306_01/appdev.102/b14260/toc.htm
SECOND APPROACH: OBJECT-RELATIONAL MODEL

• Object-oriented model tries to bring the main concepts from relational model to the OO domain
  • The heart is OO concepts with some extensions

• Object-relational model tries to bring the main concepts from the OO domain to the relational model
  • The heart is the relational model with some extensions
  • Extensions through user-defined types
CONCEPTUAL VIEW OF OBJECT-RELATIONAL MODEL

• Relation is still the fundamental structure

• Relational model extended with the following features
  • Type system with primitive and structure types (UDT)
    • Including set, bag, array, list collection types
    • Including structures like records
  • Methods
    • Special operations can be defined over the user-defined types (UDT)
    • Specialized operators for complex types, e.g., images, multimedia, etc.
  • Identifiers for tuples
    • Unique identifiers even for identical tuples
  • References
    • Several ways for references and de-references
CONCEPTUAL VIEW OF OBJECT-RELATIONAL MODEL

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>birthdate</th>
<th>movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher</td>
<td></td>
<td>9/9/99</td>
<td></td>
</tr>
<tr>
<td></td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maple</td>
<td>H’wood</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Locust</td>
<td>Malibu</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamill</td>
<td></td>
<td>8/8/88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>street</td>
<td>city</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oak</td>
<td>B’wood</td>
<td></td>
</tr>
</tbody>
</table>

- Allow of nested relations
- Repeating movies inside the stars records is redundancy
- To avoid redundancy, use pointers (references)
SUPPORT FROM VENDORS

• Several major software companies including IBM, Informix, Microsoft, Oracle, and Sybase have all released object-relational versions of their products.

• Extended SQL standards called SQL-99 or SQL3.
SQL-99: QUERY LANGUAGE FOR OBJECT-RELATIONAL MODEL

- User-defined types (UDT) replace the concept of classes

- Create relations on top of the UDTs
  - Multiple relations can be created on top of the same UDT

Create Type <name> AS (attributes and method declarations)
Creating a type for the address of stars

A hierarchy of types (inheritance)
DEFINING METHODS

CREATE TYPE PERSON AS OBJECT
(
    name VARCHAR(30),
    ssn NUMBER,
    addr ADDRESS
);

Member Function getName return varchar

If we have member function, then we need to define the type body

Create Type Body Person IS
    Member Function getName return varchar is
        Begin
            return name;
        End;
End;
"
CREATING RELATIONS

• Once types are created, we can create relations

• In general, we can create tables without types
  • But types provide encapsulation, inheritance, etc.
CREATE TYPE PERSON AS OBJECT
{
    name   VARCHAR(30),
    ssn    NUMBER,
    addr   ADDRESS
}
/

/*** Create a typed table for PERSON objects ***/
CREATE TABLE persons OF PERSON;

- Each record in the table is an object.
- That is not a relational table
/* Create a relational table with references to types */

CREATE TABLE employees
(
    empnumber INTEGER PRIMARY KEY,
    person_data REF PERSON,
    manager REF PERSON,
    office_addr ADDRESS,
    salary NUMBER
)

/* Create ADDRESS UDT */

CREATE TYPE ADDRESS AS OBJECT
(
    street VARCHAR(60),
    city VARCHAR(30),
    state CHAR(2),
    zip_code CHAR(5)
)

/* Create PERSON UDT */

CREATE TYPE PERSON AS OBJECT
(
    name VARCHAR(30),
    ssn NUMBER,
    addr ADDRESS
)

/* TABLES IN OR MODEL (II) */

12
CREATE TYPE PERSON AS OBJECT
(
  name VARCHAR(30),
  ssn NUMBER,
  addr ADDRESS
)
/

/*** Create a typed table for PERSON objects ***/
CREATE TABLE persons OF PERSON;
/

/*** Insert some data--2 objects into the persons typed table ***/
INSERT INTO persons VALUES (
  PERSON('Wolfgang Amadeus Mozart', 123456,
  ADDRESS('Am Berg 100', 'Salzburg', 'AT', '10424')))
/

INSERT INTO persons VALUES (
  PERSON('Ludwig van Beethoven', 234567,
  ADDRESS('Rheinallee', 'Bonn', 'DE', '69234')))
/
/** Put a row in the employees table **/
INSERT INTO employees (empnumber, office_addr, salary)
VALUES
(1001,
 ADDRESS('500 Oracle Parkway', 'Redwood Shores', 'CA', '94065'), 50000)
UPDATING DATA (I)

```sql
/** Set the manager and PERSON REFs for the employee **/
UPDATE employees
SET manager =
  (SELECT REF(p) FROM persons p
WHERE p.name = 'Wolfgang Amadeus Mozart')
```
UPDATE employees
SET person_data =
(SELECT REF(p) FROM persons p
    WHERE p.name = 'Ludwig van Beethoven')
COLLECTIONS AND LARGE OBJECTS

- **Book Type contains collections**
  - Arrays of authors → capture the order of authors
  - Set of keywords

```sql
create type Book as
(title varchar(20),
 author-array varchar(20) array [10],
 pub-date date,
 publisher Publisher,
 keyword-set setof(varchar(20)))
```

- **Large object types**
  - **CLOB**: Character large objects
    - *book-review* CLOB(10KB)
  - **BLOB**: binary large objects
    - *image* BLOB(10MB)
    - *movie* BLOB(2GB)

Usually provide methods inside the UDT to manipulate CLOB & BLOB
COLLECTION TYPES IN ORACLE

- Variable-Length Arrays

  CREATE TYPE typename IS VARRAY(n) OF datatype;

- Nested Tables

  CREATE TYPE typename AS TABLE OF datatype;
CREATE TYPE PHONE_ARRAY IS VARRAY(10) OF varchar2(30) /

CREATE TABLE employees
( empnumber INTEGER PRIMARY KEY, 
  person_data REF person, 
  manager REF person, 
  office_addr address, 
  salary NUMBER, 
  phone_nums phone_array 
) /
/
EXAMPLE (CONT’D)

• Inserting into the array

```sql
CREATE TABLE employees
( empnumber INTEGER PRIMARY KEY,
  person_data REF person,
  manager REF person,
  office_addr address,
  salary NUMBER,
  phone_nums phone_array
)
/

/** Put a row in the employees table **/
INSERT INTO employees (empnumber, office_addr, phone_nums)
VALUES
  (1001,
    ADDRESS('500 Oracle Parkway', 'Redwood Shores', 'CA', '94065'),
    phone_array('111-222-3333', '111-222-4444'))
/
```
NESTED TABLE VS. ARRAY

- An array has a declared number of elements.
- A nested table does not. The size of a nested table can increase dynamically.

- An array is always dense.
- A nested table is dense initially, but it can become sparse, because you can delete elements from it.
ALTER TYPES

• **Using an ALTER TYPE statement, you can:**
  • Add and drop attributes
  • Add and drop methods
  • Modify a numeric attribute to increase its length, precision, or scale
  • Modify a varying length character attribute to increase its length
  • ...
REFERENCES

- Actual Object
- References without scope
- References with scope
ACTUAL OBJECTS

CREATE TYPE PERSON AS OBJECT
{
  name    VARCHAR(30),
  ssn     NUMBER,
  addr    ADDRESS
}

/*** Create a typed table for PERSON objects ***/
CREATE TABLE persons OF PERSON;

addr is the entire object

/*** Insert some data--2 objects into the persons typed table ***/
INSERT INTO persons VALUES (PERSON('Wolfgang Amadeus Mozart', 123456,
  ADDRESS('Am Berg 100', 'Salzburg', 'AT','10424')))
CREATE TYPE emp_person_typ AS OBJECT (  
  name VARCHAR2(30),  
  manager REF emp_person_typ );
/

CREATE TABLE emp_person_obj_table OF emp_person_typ;

INSERT INTO emp_person_obj_table VALUES (  
  emp_person_typ ('John Smith', NULL));

INSERT INTO emp_person_obj_table
SELECT emp_person_typ ('Bob Jones', REF(e))
  FROM emp_person_obj_table e
WHERE e.name = 'John Smith';
CREATE TABLE contacts_ref (  
  contact_ref REF person_typ SCOPE IS person_obj_table,  
  contact_date DATE );

- Reference to a type
- The scope is a table containing objects of that type
CREATE TABLE contacts_ref (  
  contact_ref  REF person_typ SCOPE IS person_obj_table,  
  contact_date  DATE );

INSERT INTO contacts_ref  
  SELECT REF(p), '26 Jun 2003'  
  FROM person_obj_table p  
  WHERE p.idno = 1;
WHAT’S NEXT

• Second Approach: Object-Relational Model
  • Conceptual view
  • Data Definition Language (Creating types, tables, and relationships)
  • Querying object-relational database (SQL-99)
QUERYING OBJECT-RELATIONAL DATABASE

- Most relational operators work on the object-relational tables
  - E.g., selection, projection, aggregation, set operations

- Some new operators and new syntax for some existing operators

- SQL-99 (SQL3): Extended SQL to operate on object-relational databases
EXEMPLARY I

1) CREATE TYPE MovieType AS (  
   title  CHAR(30),  
   year  INTEGER,  
   inColor BOOLEAN  
);  
2) CREATE TABLE Movie OF MovieType (  
   REF IS movieID SYSTEM GENERATED,  
   PRIMARY KEY (title, year) 
);  

CREATE TYPE StarType AS (  
   name  CHAR(30),  
   address  AddressType,  
   bestMovie REF(MovieType) SCOPE Movie 
);  
3) CREATE TABLE MovieStar OF StarType (  
   REF IS starID SYSTEM GENERATED  
);  

CREATE TABLE StarsIn (  
   star  REF(StarType) SCOPE MovieStar,  
   movie  REF(MovieType) SCOPE Movie 
);  

Q1: Find the year of movie ‘King Kong’

Select m.year  
From Movie m  
Where m.title = ‘King Kong’;

Variable m is important to reference the fields

Q2: Find the title of the best movie ‘Jim Carry’

Select s.bestMovie->title  
From MovieStar s  
Where s.name = ‘Jim Carry’;

Follow a reference (pointer) using -> operator
Examples II: De-referencing

1) CREATE TYPE MovieType AS ( 
   title CHAR(30), 
   year INTEGER, 
   inColor BOOLEAN 
 );

5) CREATE TABLE Movie OF MovieType ( 
   REF IS movieID SYSTEM GENERATED, 
   PRIMARY KEY (title, year) 
 );

CREATE TYPE StarType AS ( 
   name CHAR(30), 
   address AddressType, 
   bestMovie REF(MovieType) SCOPE Movie 
 );

CREATE TABLE MovieStar OF StarType ( 
   REF IS starID SYSTEM GENERATED 
 );

CREATE TABLE StarsIn ( 
   star REF(StarType) SCOPE MovieStar, 
   movie REF(MovieType) SCOPE Movie 
 );

Q3: Find movies starred by ‘Jim Carry’

Select DEREF(movie) 
From StarsIn 
Where star->name = ‘Jim Carry’;

DEREF: Get the tuple pointed to by the given pointer

Q4: Find movies starred by ‘Jim Carry’ (Another way)

Select s.movie->title, s.movie->year, s.movie->inColor, 
From StarsIn s 
Where s.star->name = ‘Jim Carry’;

*** Using a variable for StarsIn (s in Q4) is not necessary because the table is not based on type.
EXAMPlES III: COMPARISON

1) CREATE TYPE MovieType AS (  
   title CHAR(30),  
   year INTEGER,  
   inColor BOOLEAN  
 );

5) CREATE TABLE Movie OF MovieType (  
   REF IS movieID SYSTEM GENERATED,  
   PRIMARY KEY (title, year)  
 );

CREATE TYPE StarType AS (  
   name CHAR(30),  
   address AddressType,  
   bestMovie REF(MovieType) SCOPE Movie  
 );

CREATE TABLE MovieStar OF StarType (  
   REF IS starID SYSTEM GENERATED  
 );

CREATE TABLE StarsIn (  
   star REF(StarType) SCOPE MovieStar,  
   movie REF(MovieType) SCOPE Movie  
 );

Q5: Find distinct movies starred by ‘Jim Carry’ or ‘Mel Gibson’

Select Distinct DEREF(movie)  
From StarsIn  
Where star->name = ‘Jim Carry’  
Or star->name = ‘Mel Gibson’;

• That is wrong because all objects of type MovieType are unique even if they have the same content

• Need a mechanism to define how objects compare to each other (needed for any comparison, e.g., ordering, duplicate elimination, grouping, etc.)
ORDERING RELATIONSHIPS

- Need to define how to compare objects of a given type $T$

Create Ordering For $T$

Equality or non-equality ($=, \neq$)

- EQUALS

Full comparison ($=, <, >, \leq, \geq, \neq$)

- ORDERING FULL

Identical content

- ONLY BY STATE;

User-defined function $F(O1, O2)$ and returns 0, -ve, +ve

- BY RELATIVE WITH $F;$
ORDERING FUNCTION

1) CREATE TYPE MovieType AS (  
2)    title CHAR(30),  
3)    year INTEGER,  
4)    inColor BOOLEAN  
);  
5) CREATE TABLE Movie OF MovieType (  
6)    REF IS movieID SYSTEM GENERATED,  
7)    PRIMARY KEY (title, year)  
);  

CREATE TYPE StarType AS (  
    name CHAR(30),  
    address AddressType,  
    bestMovie REF(MovieType) SCOPE Movie  
);  
CREATE TABLE MovieStar OF StarType (  
    REF IS starID SYSTEM GENERATED  
);  
CREATE TABLE StarsIn (  
    star REF(StarType) SCOPE MovieStar,  
    movie REF(MovieType) SCOPE Movie  
);  

CREATE ORDERING FOR AddressType  
ORDER FULL BY RELATIVE WITH AddrLEG;  

1) CREATE FUNCTION AddrLEG(  
2)    x1 AddressType,  
3)    x2 AddressType  
4)    ) RETURNS INTEGER  
5) IF x1.city() < x2.city() THEN RETURN(-1)  
6) ELSEIF x1.city() > x2.city() THEN RETURN(1)  
7) ELSEIF x1.street() < x2.street() THEN RETURN(-1)  
8) ELSEIF x1.street() = x2.street() THEN RETURN(0)  
9) ELSE RETURN(1)  
END IF;
EXAMPLES IV: COMPARISON

1) CREATE TYPE MovieType AS (  
   title CHAR(30),  
   year INTEGER,  
   inColor BOOLEAN  
);  

5) CREATE TABLE Movie OF MovieType (  
   REF IS movieID SYSTEM GENERATED,  
   PRIMARY KEY (title, year)  
);  

CREATE TYPE StarType AS (  
   name CHAR(30),  
   address AddressType,  
   bestMovie REF(MovieType) SCOPE Movie  
);  

CREATE TABLE MovieStar OF StarType (  
   REF IS starID SYSTEM GENERATED  
);  

CREATE TABLE StarsIn (  
   star REF(StarType) SCOPE MovieStar,  
   movie REF(MovieType) SCOPE Movie  
);  

Create Ordering For MovieType Equals Only By State;  

Q5: Find distinct movies starred by ‘Jim Carry’ or ‘Mel Gibson’  

Select Distinct DEREF(movie)  
From StarsIn  
Where star->name = ‘Jim Carry’  
Or star->name = ‘Mel Gibson’;
EXAMPLES V: GROUPING & NESTING

1) CREATE TYPE MovieType AS (
   title CHAR(30),
   year INTEGER,
   inColor BOOLEAN
);

5) CREATE TABLE Movie OF MovieType ( 
   REF IS movieID SYSTEM GENERATED,
   PRIMARY KEY (title, year)
);

CREATE TYPE StarType AS ( 
   name CHAR(30),
   address AddressType,
   bestMovie REF(MovieType) SCOPE Movie
);

CREATE TABLE MovieStar OF StarType ( 
   REF IS starID SYSTEM GENERATED
);

CREATE TABLE StarsIn ( 
   star REF(StarType) SCOPE MovieStar,
   movie REF(MovieType) SCOPE Movie
);

Q6: Find stars who participated in less than 10 movies

Select DEREF(star) 
From StarsIn 
Group by DEREF(star) 
Having count(movie) < 10;

Create at least an equality ordering on StarType

Q7: Find movie titles in 2000 where ‘Jim Carry’ is not in

Select m 
From Movie m 
Where m.year = 2000 
And m.title Not In ( 
   Select movie->title 
   From StarsIn 
   Where star->name = ‘Jim Carry’ 
   And movie->year = 2000);
**QUERYING COLLECTIONS & ARRAYS**

```sql
create type Book as
  (title varchar(20),
   author-array varchar(20) array [10],
   pub-date date,
   publisher Publisher,
   keyword-set setof(varchar(20)))
```

To get a relation containing pairs of the form “title, author-name” for each book and each author of the book:

```sql
select B.title, A
from books as B, unnest (B.author-array) as A
```

Find all books that have the word “database” as one of their keywords:

```sql
select title
from books
where ‘database’ in (unnest(keyword-set))
```

Get 1<sup>st</sup> and 2<sup>nd</sup> authors of certain book:

```sql
select author-array[1], author-array[2]
from books
where title = `Database System Concepts`
```
GENERATORS AND MUTATORS

• How to insert new data into tables

• **Generators**
  • Like the constructors in OO programming
  • Create new objects

• **Mutators**
  • Modify the value of an existing object

• For each attribute \( x \) in UDT \( T \), the system automatically creates:
  • Generator \( T() \) that returns an empty object of \( T \)
  • Mutator \( x(v) \) that sets the value of attribute \( x \) to value \( v \)
EXAMPLE

1) CREATE TYPE MovieType AS (  
   title   CHAR(30),  
   year    INTEGER,  
   inColor BOOLEAN  
);

5) CREATE TABLE Movie OF MovieType (  
   REF IS movieID SYSTEM GENERATED,  
   PRIMARY KEY (title, year)  
);

CREATE TYPE StarType AS (  
   name      CHAR(30),  
   address   AddressType,  
   bestMovie REF(MovieType) SCOPE Movie  
);

CREATE TABLE MovieStar OF StarType (  
   REF IS starID SYSTEM GENERATED  
);

CREATE TABLE StarsIn (  
   star     REF(StarType) SCOPE MovieStar,  
   movie    REF(MovieType) SCOPE Movie  
);

1) CREATE PROCEDURE InsertStar(  
   IN s CHAR(50),  
   IN c CHAR(20),  
   IN n CHAR(30)  
)

5) DECLARE newAddr AddressType;
6) DECLARE newStar StarType;

BEGIN
7)   SET newAddr = AddressType();
8)   SET newStar = StarType();
9)   newAddr.street(s);
10)  newAddr.city(c);
11)  newStar.name(n);
12)  newStar.address(newAddr);
13)  INSERT INTO MovieStar VALUES(newStar);
END;

CALL InsertStar('345 Spruce St.', 'Glendale', 'Gwyneth Paltrow');

If DBMS allows creating generators with parameters

INSERT INTO MovieStar VALUES(  
   StarType('Gwyneth Paltrow',  
               AddressType('345 Spruce St.', 'Glendale')));
CREATING RECORDS OF COMPLEX TYPES

- Collection and array types

```
create type Book as
    (title varchar(20),
     author-array varchar(20) array [10],
     pub-date date,
     publisher Publisher,
     keyword-set setof(varchar(20)))
```

Array construction

```
array [‘Silberschatz’, ‘Korth’, ‘Sudarshan’]
```

Set value attributes

```
set( v1, v2, ..., vn)
```

To insert the preceding tuple into the relation books

```
insert into books values
    (‘Compilers’, array[‘Smith’, ‘Jones’], null,
     Publisher(‘McGraw Hill’, ‘New York’),
     set(‘parsing’, ‘analysis’))
```
WHAT WE COVERED

• **First Approach: Object-Oriented Model**
  • Concepts from OO programming languages
  • ODL: Object Definition Language
  • What about querying OO databases???
    • OQL: Object Oriented Query Language

• **Second Approach: Object-Relational Model**
  • Conceptual view
  • Data Definition Language (Creating types, tables, and relationships)
  • Querying object-relational database (SQL-99)

Make use of the interesting features of Object-Oriented into database systems ➔ ODBMSs
WHEN TO CONSIDER OODBMS OR ORDBMS

• **Complex Relationships**  
  • A lot of many-to-many relationships, tree structures or network (graph) structures.

• **Complex Data**  
  • Multi-dimensional arrays, nested structures, or binary data, images, multimedia, etc.

• **Distributed Databases**  
  • Need for free objects without the rigid table structure.

• **Repetitive use of Large Working Sets of Objects**  
  • To make use of inheritance and reusability

• **Expensive Mapping Layer**  
  • Expensive decomposition of objects (normalization) and re-composition at query time
OBJECT-ORIENTED VS. OBJECT-RELATIONAL

- **Object-oriented DBMSs**
  - Did not achieve much success (until now) in the marketplace
  - No query support (Indexing, optimization)
  - No security layer

- **Object-relational DBMSs**
  - Better support from big vendors
  - Tries to make use of all advances in RDBMSs
    - Indexes, views, triggers, query optimizations, security layer, etc.
    - **Work in progress --- Long way to go**
MODIFICATIONS TO RDBMS

• **Parsing**
  • Type-checking for methods pretty complex

• **Query Rewriting**
  • New rewriting rules including complex types and collections

• **Optimization**
  • New algebra operators needed for complex types.
  • Must know how to integrate them into optimization.
  • WHERE clause exprs can be expensive!
    • Selection pushdown may be a bad idea.
MODIFICATIONS TO RDBMS (CONT’D)

• **Execution**
  - New algebra operators for complex types.
  - OID generation & reference handling.
  - Dynamic linking and overriding.
  - Support objects bigger than 1 page.
  - Caching of expensive methods.

• **Access Methods**
  - Indexes on methods, not just columns.
  - Indexes over collection hierarchies.
  - Need indexes for new WHERE clause exprs (not just <, >, =)

• **Data Layout**
  - Clustering of nested objects.
  - Chunking of arrays.
## COMPARISON

<table>
<thead>
<tr>
<th>Criteria</th>
<th>RDBMS</th>
<th>ODBMS</th>
<th>ORDBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product maturity</td>
<td>Relatively old and so very mature</td>
<td>This concept is few years old and so relatively mature feature</td>
<td>Still in development stage so immature</td>
</tr>
<tr>
<td>The use of SQL</td>
<td>Extensive supports SQL</td>
<td>OQL is similar to SQL, but with additional features like Complex objects and object-oriented features</td>
<td>SQL3 is being developed with OO features incorporated in it</td>
</tr>
<tr>
<td>Advantages</td>
<td>Its dependence on SQL, relatively simple query optimization hence good performance</td>
<td>It can handle all types of complex applications, reusability of code, less coding</td>
<td>Ability to query complex applications and ability to handle large and complex applications</td>
</tr>
<tr>
<td>Disadvantage</td>
<td>Inability to handle complex applications</td>
<td>Low performance due to complex query optimization, inability to support large-scale systems</td>
<td>Low performance in web application</td>
</tr>
<tr>
<td>Support from vendors</td>
<td>It is considered to be highly successful so the market size is very large but many vendors are moving towards ORDBMS</td>
<td>Presently lacking vendor support due to vast size of RDBMS market</td>
<td>All major RDBMS vendors are after this so has very good future</td>
</tr>
</tbody>
</table>
Table 2

A Comparison of Database Management Systems

<table>
<thead>
<tr>
<th>Criteria</th>
<th>RDBMS</th>
<th>ORDBMS</th>
<th>ODBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining standard</td>
<td>SQL2 (ANSI X3H2)</td>
<td>SQL3/4 (in process)</td>
<td>ODMG-V2.0</td>
</tr>
<tr>
<td>Support for object-oriented</td>
<td>Poor; programmers spend 25% of coding time mapping the program object to the database</td>
<td>Limited mostly to new data types</td>
<td>Direct and extensive</td>
</tr>
<tr>
<td>programming</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simplicity of use</td>
<td>Table structures easy to understand; many end-user tools available</td>
<td>Same as RDBMS, with some confusing extensions</td>
<td>OK for programmers; some SQL access for end users</td>
</tr>
<tr>
<td>Simplicity of development</td>
<td>Provides independence of data from application, good for simple relationships</td>
<td>Provides independence of data from application, good for simple relationships</td>
<td>Objects are a natural way to model; can accommodate a wide variety of types and relationships</td>
</tr>
<tr>
<td>Extensibility and content</td>
<td>None</td>
<td>Limited mostly to new data types</td>
<td>Can handle arbitrary complexity; users can write methods and on any structure</td>
</tr>
<tr>
<td>Complex data relationships</td>
<td>Difficult to model</td>
<td>Difficult to model</td>
<td>Can handle arbitrary complexity; users can write methods and on any structure</td>
</tr>
</tbody>
</table>