Relational
& Object-Oriented
& Object-Relational DBMSs

Objectives

- Define a relational database management system
- Review object oriented database management systems
- Examine object-relational database management systems
- Examine SQL3 / SQL-99 as applied by Oracle.

Next Generation Database Systems

First Generation DBMS: Network and Hierarchical

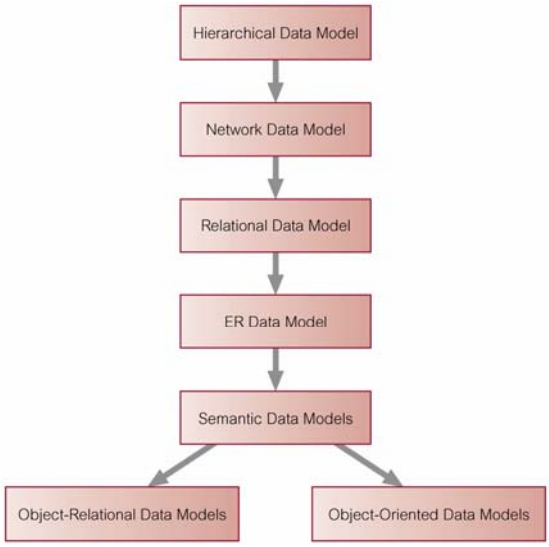
- Required complex programs for even simple queries.
- Minimal data independence.
- No widely accepted theoretical foundation.

Second Generation DBMS: Relational DBMS

- Helped overcome these problems.
- Strong theoretical foundation.

Third Generation DBMS: OODBMS and ORDBMS.

Next Generation Database Systems



Relational

- By far the most dominant type of database management system today
- RDBMS made up 80 percent of the total DBMS software market.
- The pre-relational and object database markets saw negative growth in 2000, but the relational database management system (RDBMS) sector grew 15 percent.

A Change: Motivation

- Relational model (70's): Clean and simple.
  - Great for administrative data.
  - Only atomic domains (Codd's 1NF)
    - fragmentation of 'real-world' entities during normalization
    - introduction of BLOBs without manipulation functions
  - Data separate from the operations
    - stored procedures not integrated with the data model
    - no encapsulation of attributes
  - Bad support for non standard DB applications (CASE, CAD/CAM, GIS)
    - limited reusability of model constructs

- Object-Oriented models (80's): Complicated, but some influential ideas.
  - Complex data types.
  - Object identity/references.
  - ADTs (Abstract Data Types: encapsulation, behavior goes with data).
  - Inheritance.
- Idea: Build DBMS based on OO model.

Object-Oriented DBMS's

- Standards group: ODMG = Object Data Management Group.
- ODL = Object Description Language, like CREATE TABLE part of SQL.
- OQL = Object Query Language, tries to imitate SQL in an OO framework.

Framework

- ODMG imagines OO-DBMS vendors implementing an OO language like C++ or Java with extensions (OQL) that allow the programmer to transfer data between the database and "host language" seamlessly.
- ODL is used to define *persistent* classes, those whose objects may be stored permanently in the database.
  - ODL classes look like Entity sets with binary relationships, plus methods.
  - ODL class definitions are part of the extended, OO host language.

ODL Overview

- A class declaration includes:
  1. A name for the class.
  2. Optional key declaration(s).
  3. *Extent* declaration = name for the set of currently existing objects of the class.
  4. Element declarations. An *element* is either an attribute, a relationship, or a method.

An Asset Management Scenario

- Nusomac Entertainment Corp.
  - Assets: cartoon videos, stills, sounds
  - "Chase the Border Collie" films show worldwide
  - Civu licenses Chase videos, stills, sounds for various purposes:
    - action figures
    - video games
    - product endorsements
  - DBMS must manage assets and business data

Why not a Standard RDBMS?

create table frames (frameno integer, image BLOB, category integer)

- Binary Large Objects (BLOBs) can be stored and fetched.
- User-level code must provide all logic for BLOBs.
- Scenario: Client (Machine A) requests "thumbnail" images for all frames in DBMS (Machine B).
  - Inefficient, too hard to express queries.

Solution 1: Object-Oriented DBMS

- Idea: Take an OO language like C++ or Java, add persistence & collections.

```
class frame {
   int frameno;
   jpeg *image;
   int category;
}
persistent set <frame *> frames;
foreach (frame *f, frames)
   return f->image->thumbnail();
```

- Shut down the program. Start it up again. Persistent vars (e.g. frames) retain values!

Attribute and Relationship Declarations

- Attributes are (usually) elements with a type that does not involve classes.
  attribute <type> <name>;
- Relationships connect an object to one or more other objects of one class.
  relationship <type> <name>
  inverse <relationship>;

Multiplicity of Relationships

- All ODL relationships are binary.
- Many-many relationships have Set<…> for the type of the relationship and its inverse.
- Many-one relationships have Set<…> in the relationship of the "one" and just the class for the relationship of the "many."
- One-one relationships have classes as the type in both directions.

Coping With Multiway Relationships

- ODL does not support 3-way or higher relationships.
- We may simulate multiway relationships by a "connecting" class, whose objects represent tuples of objects we would like to connect by the multiway relationship.

OQL

- OQL is the object-oriented query standard.
- It uses ODL as its schema definition language.
- Types in OQL are like ODL's.
- Set(Struct) and Bag(Struct) play the role of relations.

The OO Database system manifesto

1. Complex objects must be supported.
2. Object identity must be supported.
3. Encapsulation must be supported.
4. Types or Classes must be supported.
5. Types or Classes must be able to inherit from their ancestors.
6. Dynamic binding must be supported.
7. The DML must be computationally complete.
8. The set of data types must be extensible.
9. Data persistence must be provided.
10. The DBMS must be capable of managing very large databases.
11. The DBMS must support concurrent users.
12. DBMS must be able to recover from hardware/software failures.
13. DBMS must provide a simple way of querying data.

+ optional features including type checking/inferencing, versions…

OODBMS applications

- OODBMSs good for:
  - complex data
  - fixed set of manipulations (no ad-hoc queries)
  - improved performance for non traditional apps.
  - applicability to advanced database apps.

OODBMS problems

- Problems:
  - Still rudimentary query support (OQL)
  - Schema evolution very difficult
  - Lack of Universal Data Model.
  - Lack of Experience.
  - Lack of Standards.
  - Object Level Locking may impact Performance.
  - Complexity.
  - Lack of Support for Views.
  - Lack of Support for Security.
  - Some argue it's back to the network data model

- A modest success in the marketplace

- Another problem
  "Data tends to stick where it lands."

- Relational databases are not going away. Too much invested in them already.

- Object-oriented DBMS's failed because they did not offer the efficiencies of well-entrenched relational DBMS's.
- Object-relational extensions to relational DBMS's capture much of the advantages of OO, yet retain the relation as the fundamental abstraction.

- Back to the original problem with Nusomac Entertainment Corp.

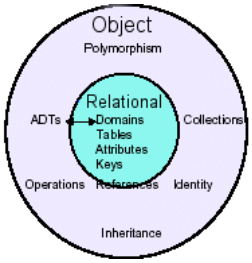
  What to do?

Solution 2: Object-Relational

- Idea: Add OO features to the type system of SQL. i.e. "plain old SQL", but...
  - ORDBMs keep "relation" as the fundamental abstraction
    - Unlike the "class" concept in ODBMs
  - Extension of the relational model
    - Structured & multivalued attributes
    - Inheritance for both relations & types
    - ADTs for domains
    - Object identity for relation rows
    - Operators overloading

Object-Relational

- Extension of SQL
  - Schemas: tables at the top, OO richness within
  - Queries: extensions to support the added richness
- Relational vendors all moving this way (SQL3 or SQL-99). Big business!



SQL-99 and Oracle Features

- SQL-99 includes many of the object-relational features to be described.
- However, being so new, different DBMS's use different approaches.
  - We'll sometimes use features and syntax from Oracle.

Object-Relational User Defined Types

- A *user-defined type*, or UDT, is essentially a class definition, with a structure and methods.
- Two uses:
  - As a *rowtype*, that is, the type of a relation.
  - As the type of an attribute of a relation.

Object-Relational UDT Definition

```
CREATE TYPE <typename> AS (
      <list of elements, as in CREATE TABLE>
);
```

- Oracle syntax:
  1. Add "OBJECT" as in CREATE … AS OBJECT.
  2. Follow with / to have the type stored.

Object-Relational UDT in Oracle

- Oracle permits definition of types similar to the types of SQL.

```
CREATE TYPE PointType AS OBJECT (
  x NUMBER,
  y NUMBER );
/
```

Object-Relational in Oracle

- An object type can be used like any other type in further declarations of object-types or table-types. For example

```
CREATE TYPE LineType AS OBJECT (
  end1 PointType,
  end2 PointType );
/
```

- Then, could create a relation that is a set of lines with ``line ID's'' as:

```
CREATE TABLE Lines (
  lineID INT,
  line LineType );
```

- Construct two values of type PointType, these values are used to construct a value of type LineType

```
INSERT INTO Lines
VALUES(27, LineType(
        PointType(0.0, 0.0),
        PointType(3.0, 4.0)
        )
);
```

Declaring and Defining Methods in Oracle

- If want to add a length member function to LineType:

```
CREATE TYPE LineType AS OBJECT (
  end1 PointType,
  end2 PointType,
  MEMBER FUNCTION length(scale IN  NUMBER)
        RETURN NUMBER,
  PRAGMA RESTRICT_REFERENCES(length,          WNDS)
  );
/
```

Declaring and Defining Methods in Oracle

```
CREATE TYPE BODY LineType AS
   MEMBER FUNCTION length(scale NUMBER)
        RETURN NUMBER IS
   BEGIN
        RETURN scale *
        SQRT((SELF.end1.x-SELF.end2.x)*(SELF.end1.x-SELF.end2.x) +
                (SELF.end1.y-SELF.end2.y)*(SELF.end1.y-SELF.end2.y)
        );
   END;
END;
/
```

Object-Relational in Oracle: Queries

- The following query finds the lengths of all the lines in relation Lines, using scale factor 2 (i.e., it actually produces twice these lengths).
- Uses the member function length.

```
SELECT lineID, ll.line.length(2.0)
  FROM Lines ll;
```

Object-Relational in Oracle: Other Queries

```
SELECT ll.line.end1.x, ll.line.end1.y
FROM Lines ll;
```

- prints the x and y coordinates of the first end of each line.

```
SELECT ll.line.end2
FROM Lines ll;
```

- prints the second end of each line, but as a value of type PointType, not as a pair of numbers.
- For instance, one line of output would be PointType(3,4).

Object-Relational in Oracle: Nested Tables

- A more powerful use of object types in Oracle is the fact that the type of a column can be a table-type.
- That is, the value of an attribute in one tuple can be an entire relation
- In order to have a relation as a type of some attribute, first have to define a type using the AS TABLE OF clause.
- For instance:

```
CREATE TYPE PolygonType
  AS TABLE OF PointType;
/
```

- says that the type PolygonType is a relation whose tuples are of type PointType; i.e., they have two components, x and y, which are real numbers.

- Now, can declare a relation one of whose columns has values that represent polygons; i.e., they are sets of points.
- A possible declaration, in which polygons are represented by a name and a set of points is:

```
CREATE TABLE Polygons (
   name VARCHAR2(20),
   points PolygonType)
   NESTED TABLE points STORE AS PointsTable;
```

Object-Relational in Oracle: Nested Tables - Insert

- Here is a statement inserting a polygon named ``square'' that consists of four points, the corners of the unit square.

```
INSERT INTO Polygons VALUES( 'square',
    PolygonType(PointType(0.0, 0.0),
                    PointType(0.0, 1.0),
                    PointType(1.0, 0.0),
                    PointType(1.0, 1.0)
                )
);
```

Object-Relational in Oracle: Nested Tables - Query

- We can obtain the points of this square by a query such as:

```
SELECT points
FROM Polygons
WHERE name = 'square';
```

OO/OR-DBMS Summary

- Traditional SQL is too limited for new apps.
- OODBMS: Persistent OO programming.
  – Difficult to use, rudimentary query language.
- ORDBMS: Best (?) of both worlds:
  – Catching on in industry and applications.
  – Pretty easy for SQL folks to pick up.
  – Still has growing pains (SQL-3 standard still a moving target).

Summary (Contd.)

- ORDBMS offers many new features.
  - But not clear how to use them!
  - Schema design techniques not well understood
  - Query processing techniques still in research phase.
    - A moving target for OR DBA's!

- Prediction: You will use an ORDBMS in the future.

Stonebraker's Application Matrix

	No Query	Query
Complex Data	OODBMS	ORDBMS
Simple Data	File System	RDBMS

Supposition: Most applications will move to the upper right.