OBJECT RELATIONAL DBMS

Motivations

- Necessity to maintain compatibility with existent investments
  - Relational DBMS
  - Client-server applications
  - SQL-based application building tools
- Necessity to support complex data and operations
  - Textual
  - Geometric
  - Geographic
  - Multimedia
  - ...
Weaknesses of the Relational Model

- 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

Strengths of the Object Model

- Objet Identity
  - favors data sharing
  - supports typed pointers
- Data Encapsulation
  - enables the isolation of data from operations
  - facilitates the evolution of data structures
- Inheritance of operations and structures
  - facilitates reusability of data types
  - allows programs customization according to application needs
- Possibility to define abstract operations (polymorphism)
  - increase developers productivity
Application vs Data Complexity

Complex Applications

Object-Oriented

CASE CAD

Simple Data

ADDS

OT

Network

T

Relational

Simple Applications

Some Market Facts
Third Generation Database System Manifesto (90)

- Support rich object structures and rules
  - Rich type system, inheritance, encapsulation
  - Functions, optional unique ids, rules/triggers
- Subsume second generation database systems
  - High-level query-oriented interface
  - Stored and virtual collections
  - Updatable views
  - Data model/performance feature separation
- Open to other subsystems (tools, middleware)
  - Accessible from multiple languages
  - Layered persistence-oriented language bindings
  - Query-shipping architecture

What are the Options?

- Build a new DB technology
  - Object-oriented DBMS
    (Tightly integrated):
    OOPL w/built-in DBMS
- Wrap Relational DBMS
  - Object-oriented client wrapper
    (Loosely integrated):
    OOPL + relational DBMS
- Extend Relational Systems
  - Objet-Relational DBMS
    (Newly integrated):
    Relational model + OO features
OO Client Wrappers

- Available from a number of vendors
  - Ardent, Persistence Software, Ontologic, HP, ...
- Language-specific relational wrappers
  - Proxy classes for C++ or Java (or Smalltalk)
  - Mapping of row data into language objects
  - Client-side (or middle-tier) object caching and method execution
- Why is this approach attractive?
  - Good use of existing systems
  - Rapid development of OO applications, against existing enterprise data, for "business objects"
  - New or changed wrapper to leverage investment

Ardent Java Relational Binding

- Java objects are stored transparently in a relational database
  - Object identity=primary key
  - Object reference=foreign key
- Provides code persistence
- 100% Java (any JDBC 1.2 compliant driver)
- JDK 1.1
OO Client Wrappers are not The Solution

- Paradigm mismatch for querying
  - C++ or Java for simple business logic and navigation, against object-oriented schema
  - SQL for queries, against relational schema
- Choice forced for business logic & rules
  - Do on server, using DBMS facilities?
    - Check constraints, referential integrity constraints, triggers, stored procedures, authorization
  - Do on client, using OO wrapper facilities?
    - C++ or Smalltalk (or Java) programming
- This had better be a stop-gap solution
  - RDBMS could become a storage manager, throwing away 20+ years of successful R&D!

The Object-Relational Model

- 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
  - Objet identity for relation rows
  - Operators overloading
- Extension of SQL
  - Schemas: tables at the top, OO richness within
  - Queries: extensions to support the added richness
### Tables & Objets: Example (Oracle8)

<table>
<thead>
<tr>
<th>Name</th>
<th>Style</th>
<th>Live-Time</th>
<th>Influences</th>
<th>Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Claude Monet</td>
<td>Impressionism</td>
<td>1840-1926</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Artifacts

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Title</th>
<th>Material</th>
<th>Photo</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edouard Manet</td>
<td>1863</td>
<td>Haystacks at Chailly at Sunrise</td>
<td>Oil on Canvas</td>
<td><img src="image" alt="Haystacks at Chailly at Sunrise" /></td>
<td></td>
</tr>
<tr>
<td>Eugene Boudin</td>
<td>1863</td>
<td>Wheatstacks End of Summer</td>
<td>Oil on Canvas</td>
<td><img src="image" alt="Wheatstacks End of Summer" /></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Meule, Soleil Couchant</td>
<td>Oil on Canvas</td>
<td><img src="image" alt="Meule, Soleil Couchant" /></td>
<td></td>
</tr>
</tbody>
</table>

---

### The Fully Object-Relational DBMS (Stonebraker 96)

- **Base type extension**
  - dynamic linking
  - client or server activation
  - security
  - callback
  - user-defined access methods
  - arbitrary-length data types

- **Inheritance**
  - data and function inheritance
  - overloading
  - inheritance of types, not tables
  - multiple inheritance

- **Complex objects**
  - type constructors (set, record, reference)
  - user-defined functions must have support for use of complex types
  - arbitrary-length complex data types
  - SQL support

- **Rule system**
  - events and actions
  - integration of rules with inheritance and type extension
  - rich execution semantics for rules
  - no infinite loops
The Object Relational Approach

- Commercial systems exist today
  - IBM DB2 CS (V2.1) and CA-Ingres
    - User-defined types & functions, large objects, triggers
  - Illustra, UniSQL/X
    - Early providers of ADTs, row objects, inheritance
  - IBM DB2 UDB, Informix, Oracle
    - "Universal server" products contain subsets of all this stuff

- Standards right around the corner
  - SQL support ("intergalactic data speak")

- However, more an evolution than a revolution

Risk being a compromise implementation and hence inefficient

The SQL3 Standard
The SQL3 Components

- Part 1: Framework
- Part 2: Foundation & General Purpose Facilities (SQL Foundation 846 p)
- Part 3: Call Level Interface (SQL/CLI 181 p)
- Part 4: Persistent SQL Modules (SQL/PSM 143 p)
- Part 5: Host Language Bindings (SQL/Bindings 209 p)
- Part 6: Transaction Monitor XA Interface (SQL/XA 51 p)
- Part 7: Temporal Extensions (SQL/Temporal)
- Other components
  - specification of multimedia ADT (SQL/MM)
  - specification of RDA protocol (SQL/RDA)

The Normalization Process

- International group
  - ISO/IEC JTC1/SC 21/WG3 DBL
- Active Countries
  - Australia, Brazil, Canada, France, Germany, Japan, Korea, The Netherlands, United Kingdom, United States
- ANSI X3H2 (http://www.ansi.org)
- Two versions of the SQL standard (available from ANSI):
  - ISO/IEC 9075:1992, "Database Languages - SQL"
  - ANSI X3.135-1992, "Database Language SQL"
- Under validation by NIST (http://ncsl.nist.gov)
  - SQL2-92 entry level
- Planning
  - Committee Draft – 1/96
  - Draft International Standard – 12/98
  - International Standard – 7/99
SQL3 - The Objet Model

- User-defined data types
  - ADTs with optional OID (encapsulation of structure+behavior)
  - Named row types with implicit OID (unencapsulated)
  - Distinct types (scalar types)
- Support of complex objects
  - Type constructors for collection types (sets, lists, and multisets)
  - Type constructors for row types and reference types (REF)
- Inheritance
  - Definition of subtypes & subtables
  - Multiple inheritance is supported
- User-defined functions and procedures
  - Internal (i.e. in SQL3) or External (i.e. in a PL)
- Support for large objects (BLOBs and CLOBs)

SQL3 Abstract Data Types

- `CREATE TYPE < ADT name > <ADT body>`
- `<ADT body>`
  - `<OID options> ::= WITH OID [NOT] VISIBLE`
    - objects without an OID by default
    - OIDs can be seen by queries, constraints and other ADTs
  - `<subtype clause> ::= UNDER <supertype clause>`
    - multiple inheritance is supported with explicit conflict resolution
  - `<member list>`
    - `<attribute definition>`: encapsulation levels public, private, protected
    - `<operator name list>`: overloaded operators (Boolean result)
    - `<ordering definition>`: EQUALS, LESS THAN, RELATIVE, HASH
    - `<function declaration>`: with the ADT as parameter or result
    - `<cast clause>`: functions for valid type conversion
    - `<procedure clause>`: with body defined externally or in SQL3
SQL3 ADTs: Examples

- **A type with OID**
  ```sql
  CREATE TYPE WITH OID VISIBLE Address (  
    PUBLIC num INT, street CHAR(20), city CHAR(15), country CHAR(10),  
    EQUALS DEFAULT, LESS THAN NONE,  
    PUBLIC FUNCTION distance(a Address, b Address) RETURNS FLOAT  
    PUBLIC FUNCTION fullAddr(a Address) RETURNS CHAR(45)
  )
  ```

- **A type without OID**
  ```sql
  CREATE TYPE Person (  
    PUBLIC name CHAR(50), address: Address, Nationality: VARCHAR,  
    PRIVATE birth-date DATE,  
    PUBLIC FUNCTION age (DATE, DATE) RETURNS INT
  )
  ```

- **A subtype**
  ```sql
  CREATE TYPE Artist UNDER Person (PUBLIC style VARCHAR)
  ```

Use of ADTs as Domains of Attributes

SQL3 Functions

- **FUNCTION**<F name><F params>RETURNS<type> AS  
  <F body> END FUNCTION
- `<F body> = <SQL procedure> | <external procedure>
- **Examples**
  ```sql
  FUNCTION fullAddr (a Address) RETURNS CHAR(45) AS
    z CHAR(10)
    BEGIN
      :z = findZip (a.street, a.city);
      RETURN (a.street "|| a.city "|| z);
    END;
    END FUNCTION
  ```

  ```sql
  FUNCTION findZip (CHAR(20), CHAR(15)) RETURNS CHAR(10) AS
    EXTERNAL NAME './findzip.so'
    LANGUAGE C;
    END FUNCTION
  ```
**SQL3 Type Constructors**

- **Basic constructors**: SET, MULTISET, LIST
  
  ```sql
  CREATE TYPE Person (..., address LIST (Address))
  ```

- **Object references**: for types created “without OID”
  
  ```sql
  CREATE TYPE Artist UNDER Person (...,
  influences SET (REF(Person)))
  ```

- **Unnamed or Named row types**: for tuple values
  
  ```sql
  CREATE TYPE Artist UNDER Person (...,
  influences SET (ROW (name: CHAR(50), date DATE)))
  CREATE ROW TYPE Museum(denomination VARCHAR, addr Address)
  ```

- **Distinct types**: declare that two otherwise equivalent type declarations are to be treated as separate data types
  
  ```sql
  CREATE DISTINCT TYPE US_dollar AS DECIMAL(9,2)
  CREATE DISTINCT TYPE Canadian_dollar AS DECIMAL(9,2)
  ```

**SQL3 Tables**

- **Tables may have**
  
  - attributes of an ADT type
  - attributes with complex values (SET, MULTISET, LIST, ROW)
  - attributes of Reference type (REF <type> or with OID)

- **Possibility to use predefined types**: ADTs or ROW types

  ```sql
  CREATE TABLE Artists OF Artist;
  CREATE TABLE Museums OF Museum;
  ```

- **Possibility to define new types**: the tuple type of the table

  ```sql
  CREATE TABLE Artifacts OF NEW TYPE Artifact (title CHAR(30),
  creator REF(Artist), Material VARCHAR, ...);
  ```

- **Possibility to refine the tables**

  ```sql
  CREATE TABLE Paintings UNDER Artifacts (Location REF(Museum))
  ```
SQL3 Functions & Operators Call

- Find the museums near 2 miles to the San Diego Museum of Art
  
  ```sql
  SELECT m2.denomination
  FROM Museums m1, Museums m2
  WHERE m1.denomination = 'San Diego Museum of Art' and
  distance(m1.addr,m2.addr) < 2 ;
  ```

- Find the artists living in the same address
  
  ```sql
  SELECT a1.name, a2.name
  FROM Artists a1, Artists a2
  WHERE a1.address = a2.address and
  a1.name != a2.name;
  ```

Dereferencing Objects in SQL3

- Possibility to apply the functions Ref and DeRef (implicit)
  
  ◆ Find the Museums of San Diego
    
    ```sql
    SELECT m.denomination FROM Museums m
    WHERE m.address..city = 'San Diego'
    ```

  ◆ What is the street of the Museum exhibiting “Haystacks”
    
    ```sql
    SELECT p.located.address..street FROM Paintings p
    WHERE p.title="Haystacks"
    ```
Dereferencing Objects in SQL3

- Possibility to use a cascading “dot” notation
  - Find the Paintings of the Artists influenced by “Manet”
    
    ```
    SELECT REFER(p) FROM Artists a, Paintings p
    WHERE p.creator.name = a.name and
    'Manet' in a.influences.name
    ```

- Generalization to multiple paths
  - Find the Paintings of Artists influenced by “Manet” at 1863
    
    ```
    SELECT REFER(p) FROM Artists a, Paintings p
    WHERE p.creator.name = a.name and
    influences.(name = 'Manet' and year=1863)
    ```

- Any collection may play the role of a table

SQL3 Stored Procedures (PSM)

- A number of new statement types have been added in SQL3 in order to make SQL computationally-complete:
  - variables declaration
  - assignment statement for SQL values
  - CALL and RETURN statements for SQL procedures
  - control statements CASE, IF for execution paths
  - LOOP, WHILE and REPEAT statements for repeated execution of a block of SQL statements
  - exceptions SIGNAL, RESIGNAL

- Additional control facilities available include compound statements and exception handling
  - CONDITION and HANDLER declarations for exceptions
COMPARING SQL3 AND ODMG

SQL3: A Standard under Evolution

- Rival proposal to ODMG
- Agreement between constructors of object DBMS
- Support of the core OMG object model
- SQL variants to process nested collections
- Agreement between ANSI X3 H2 and ODMG
- Definition of a query language integrating relational & object models
- Convergence of objet-relational worlds about SQL3
- Several pending issues
- Visibility of OID?
- Identity of ROW types?
- Multi-valued paths?
- Referential Integrity?
Programming Environment

- OQL relies on the following assumptions
  - Its statements are embedded in a PL sharing the same data model
  - The programming language is object-oriented (e.g. C++, Java)
- SQL3 objects are not necessarily objects of the host PL
  - In all SQL versions there is a cursor-based mechanism allowing to pass stored data from the database to the host PL variables
  - SQL3 external functions for ADTs provide additional communication interfaces

The Role of Relations

- Relations is The kernel of SQL3
  - Row types are used to define relations while ADTs represent new attribute domains
  - References to tuples can be viewed as object references, however objects can’t persist outside of relations
  - Row types are not encapsulated
- Relations are also supported by ODMG
  - But relations are only one of the possible types which can be constructed using ODMG collections (e.g. sets of tuples)
  - ODMG classes may be of tuple type but unlike row types class behavior can be also defined
  - ODMG classes are must closer to SQL3 ADTs
The Object Identity

- ODMG class objects have
  - an OID generated by the system and cannot be stored or manipulated by the user
  - queries may create new objects
- SQL3 ADTs & row types may have
  - a tuple identity playing the role of a key, which can be stored as an ordinary value
  - queries may only retrieve reference values

AN ORDBMS EXAMPLE: ILLUSTRAT
Introduction

- Based on Postgres (Post Ingres)
- Product = client support + server + DataBlade modules
- Principal extensions to support object relational
  - type extensions
  - complex objects
  - user functions and operators
  - inheritance
- Other features
  - Rules and alerters
  - Time travel
  - archiving
  - OLTP support

Illustra Architecture

- TCP/IP for communications
- Clients use Illustra API library
- User functions can run in client or server address spaces
Type Extensions

- Standard base types
  - Numerics
  - Text
  - Special e.g. date
- User defined base types
- User defined composite types
- User defined functions and operators

User defined Composite Types: Examples

- - person has name, age
  create type person_t
  ( first_name varchar,
    last_name varchar,
    age integer );
- - student is a person who
  is on a course
  create type student_t
  ( course varchar
  under person_t );
- - students table
  create table students
  ( student student_t );

- - employees table
  create table employees
  ( emp person_t,
    position varchar );
- - after populating the tables
  select emp.last_name, emp.age
  from employees
  where position = 'boss';
  select student.last_name,
    student.first_name
  from students
  where student.course = 'BA';
User defined Functions & Operators

- Operators and functions for standard base types
  - arithmetic and comparison operators
  - count, sum etc. functions
- User defined functions and operators can take complex types as arguments and return complex types as results
- Datablades add their own functions and operators

User defined Functions: Examples

- - find who is on a course
  create function is_on_course(varchar)
  returns setof(student_t)
  as
  select student
  from students
  where course = $1;

- - and use
  select student.name
  from is_on_course('BA');
User defined Complex Objects

- complex user defined types
- create tables using these types
- type constructors
  - sets
    - table = set of composite type
    - nesting is allowed
  - arrays
    - arrays of base types, of arrays, of references
  - references
    - OID of row of a table
    - ref/deref operators
- supports arbitrary complexity e.g. sets of references to composite objects which include composite objects …

User defined Complex Types: Examples

- department type
  create type dept_t
  (  dept_name       char(20),
      employees     setof(emp_t),
      manager       ref(emp_t));

- and a table of departments
  create table departments
  (department    dept_t),

- function to find manager of a department
  create function the_manager(varchar)
  returns emp_t
  as
  select dref(manager)
  from departments
  where dept_name = $1;

- to use
  select the_manager(sales).last_name;
Inheritance

- data types and functions
- reuse data type definitions
  - inheritance hierarchies
  - multiple inheritance
- reuse function definitions
  - function overloading

Illustra DataBlades

- Standard software modules that plug into the database to extend its capabilities with domain specific data management:
  - content-based query capability
  - comparison operators
  - appropriate index methodology
  - intelligent query optimization
  - simple aggregation functions
- Examples:
  - Text
  - 2D Spatial
  - 3D Spatial
  - Image
  - Time series
Example - 2D DataBlade

- Application
  - Geographical Information Systems (GIS)
  - Anything requiring maps, layouts
- 2D data types and functions
  - Points, lines, paths, circles, polygons, etc.
  - Distance of a point from a line
- System supplied constructor functions
- Rtree indexing
- Support for large 2D spatial types

Data Type

Data Structure

Data Behavior

Index Schema

Access Methods

Interface to the database server

More 2D Functions

- Return data about objects e.g. area, angles
- Constructs objects e.g. bounding box given a circle, path given a set of co-ordinates
- Numeric data about objects e.g. distance between two points
- Boolean data about objects e.g. is one object contained within another, does one object intersect another
External Representation

- Text string e.g. storing a new 'box' into table 'boxes' with single attribute of type 'box'
  - insert into boxes
  - values ('(1,2, 3, 5)');
  - creates

Airport and Cities: an Example

- our airport
- that city
- this city
- A42
- M42
Airport and Cities: Create and Populate Tables

- create tables
  - create table roads
    (road_number char(6),
     road iseg);
  - create table cities
    (city_name char(20),
     city ellp);
  - create table airports
    (airport_name char(20),
     airport box);

- create airport, cities and roads
  - insert in roads
    values (‘M42’, (8,0,8,10));
  - insert in roads
    values (‘A42’, (0,4,20,4));
  - insert into airports
    values (‘our airport’, (6,2,9,3));
  - insert in cities
    values (‘this city’, (5,7,4,2,90));
  - insert in cities
    values (‘that city’, (14,3,6,4,90));

Airports and Cities: Queries

- which roads goes to the airport
  select road_number
  from roads, airports
  where intersects(road, airport);
- which roads go through ‘that city’
  select road_number
  from roads, cities
  where intersect(road, city)
  and city_name = ‘that city’;
- how far is ‘this city’ from ‘our airport’
  select distance(center(city), center(airport))
  from cities, airports
  where city_name = ‘this city’ and airport_name = ‘our airport’;
Rules and Alerters

- protect the database integrity
- triggered by update or retrieval event
- action can involve update or retrieval
- rules are inherited
- can execute before or after an event

REFERENCES

- Nial Wareing: “Object-Relational Databases” DB@bton Database Course Slides 1999