Object-relational Databases

Extend relational databases with OO features

Recently seen (Meijer, Bierman: CACM 4/11)

SQL	coSQL (noSQL)
Children point to parents	Parents point to children
Closed world	Open world
Entities have identity (extensional)	Environment determines identiy (intensional)
Necessarily strongly typed	Potentially dynamically typed
ACID transactions	BASE updates (eventual, single value)
Environment coordinates changes	Entities responsible to react to changes
Value-based, strong reference	Computation-based, weak reference
Not compositional	Compositional
Automatic query optimization	Manual optimization / patterns

- A great deal of confusion: expect huge changes
  - 100M programmers, cloud, new HW, big data, globalization
- You must understand the basic concepts and recompose!

OO 101

- Object Identity
  - Every object can be uniquely identified
  - Independent of state of the object
  - Different comparison operators (value, id)
- Substitutability in Type Hiearchy
  - Set inclusion relationship between sub and super-type
  - Enables reuse via inheritance and polymorphism
- Information Hiding
  - Couple behavior to data structure
- *How would you rank importance? Orthogonal?*

Why Objects and Databases

- New kinds of applications, new kinds of data
  - text, audio, video, maps
  - Hierarchies (Region, Country, City)
  - Time series
- Complex Functions
  - Routing, FFT, ...
- Use different kinds of data within a single query

  Find all maps that contain routes from Zurich to Cologne!

- Possible future: Merge DB and App layer (?)

Problems of relational databases

- Each polyhedron is composed of at least four surfaces
- Each surface has at least three edges
- Each vertex belongs to at least three edges
- *How do you rotate a cube?*

Surface

pid	sid
p1	s1
p1	...
p1	s6

Edge

eid	SA	SB	PA	PB
e1	s1	s4	v1	v4
e2	s1	s2	v2	v3
...	...	...	...	...

Vertex

vid	X	Y	Z
v1	0.0	0.0	0.0
v2	1.0	0.0	0.0
...	...	...	...
v8	0.0	1.0	1.0

Problems of relational databases

- Structure of polyeder is only implicit in RDB
  - Makes it difficult to implement operations like rotate
  - Difficult to reuse implementations of rotate (view?)
- Inefficient to execute operations like rotate
  - Join between three tables
  - Eight different updates
- Solution
  - Support the definition of complex *objects* in database
  - Support the definition of complex *operations* in database

Object-oriented Databases

- Research since 1984; many nice results
- Some products in the late 80 ‘s, early 90 ‘s (Versant, GemStone, ObjectStore, O2, Poet)
- ODMG Standard 2.0: Cattell 1997
- Applications: CAD and some other niches
- Overall failure: did not scale, were not robust
  - Too much focus on „navigation“ and „C++“
  - Not really a DB (poor query optimization & TAs)
- Revival lately: Object caches (e.g., SAP)
- Literature: Kemper, Moerkotte 1994

Object-relational Databases

- Laguna Beach Report 1990 (Stonebraker et al.)
- Products since ca. 1997
  Informix Universal Server, Oracle 8i, IBM UDB
- Evolutionary Approach
  - Take RDBMS as a starting point
  - Add OO functionality to RDBMS
  - Do not sacrifice any RDBMS features
- Standards: SQL 99, SQL-3
- Not really a success either. But, reality!
- Literature: Chamberlin, Using the New DB2

Features of ORDBMS

- Large Objects
- User-defined types (UDT)
- Object Identity, Object References
- Inheritance
- Nested tables
- User-defined functions (UDF)
- ADTs (Data Blades, Extenders, Cartridges)

Warning: We mix SQL99, IBM, and Oracle syntax.
Might be outdated. Please, consult manual!!!

Features of ORDBMS

- Large Objects
- User-defined types (UDT)
- Object Identity, Object References
- Inheritance
- Nested tables
- User-defined functions (UDF)
- ADTs (Data Blades, Extenders, Cartridges)

How to look at these extensions?

- Compare with features of Java – what & why different?
- How can these extensions be implemented? Disruptive?

Large Objects

- „varchars“ are limited to 4096 Bytes
- Larger objects (e.g., videos) stored in blobs
  - Blob = Binary Large OBject
  - Clob = Character Large OBject
- Configuration: Compression and TA properties

```
CREATE TABLE personalWebPage(
        url  varchar(100),
        picture blob(100K) NOT LOGGED COMPACT,
        html clob(20K) NOT LOGGED COMPACT
);
```

Processing Blobs

- Restrictions
  - No indexes
  - No comparisons such as „like“, <, ...
  - No „distinct“, „group by“ or „max“
  - More features provided by data blades (see later)
- Data stored separately in file system
  - Database only stores *Locator* in the table
- Locators can be used in application programs

Distinct Types

- Simplest kind of user-defined type (UDT)
- Supports strong typing
- Explicit casts to base types allowed

```
CREATE DISTINCT TYPE dollar AS DECIMAL(9,2);
CREATE DISTINCT TYPE euro AS DECIMAL(9,2);

CREATE TABLE product(
        name  varchar(100),
        USAsales dollar,
        Esales euro);
```

Distinct Types

- Strong typing: Helps to avoid following bugs

```
SELECT *
FROM    product
WHERE  Esales > USAsales;

SELECT name, sum(Esales + USAsales)
FROM    product
GROUP BY name;
```

Complex UDTs

- Work like „struct“ in C
  - There is no magic here: implementation straightforward
- Can be used as column and row types

```
CREATE TYPE emp(
      name  varchar(30),
      salary euro,
      picture blob(100K))
```

```
CREATE TYPE manager(
      name varchar(30),
      history clob(100K),
      salary dollar)
```

```
CREATE TABLE project(
      leader manager,  // UDT as column type
      secretary emp)
```

Implementation of UDTs

```
CREATE TABLE project(
  leader manager,
  secretary emp);
```

is the same as

```
CREATE TABLE project(
  leader.name varchar(30),
  leader.salary euro,
  leader.picture blob(100K),
  emp.name varchar(30),
  …);
```

Queries with Complex Types

- Constructors, Observers, Mutators provided implicitly by ORDBMS.

```
INSERT INTO project(leader, secretary)
  VALUES( manager(„Lisa“, [], dollar(30000)),
                  emp(„Heini“, euro(20000), [])) );

SELECT *
FROM   project p
WHERE p.leader.name like „%isa%“;
```

Row Types

- Use complex UDT for a whole row
- Support references on rows

```
CREATE ROW TYPE manager(name varchar(100));
CREATE ROW TYPE emp(
        name varchar(100),
        boss  REF(manager));

CREATE TABLE manTable OF manager;
CREATE TABLE empTable OF emp(
        PRIMARY KEY name,   // keys belong to tabs!
        SCOPE FOR boss IS manTable);
```

Object Identity, References

- Only top-level tuples (i.e., rows) have an identity
- Identity is independent of value of tuple
  - Very different concept than key/foreign key!!!
  - Identifiers are defined by ORDBMS
- References can be typed and scoped
  - You should do both. Why?
- Usage of references using path expressions
  - Be careful with „.“ vs. „->“ notation

Path Expressions

```
SELECT e.boss->address.city
FROM   empTable e
WHERE e.boss->name like „%isa%“;
```

- Security: How do you do authorization?
  - Can only be done statically for scoped references.
- Java does not differentiate between object and reference to object. Why do SQL and C do that?
  - Implications on life time of object, syntax („.“ vs. „->“)

Inheritance

- A row type can be derived from other row type(s)
- Row types: Inheritance of attributes of super-type
- Tables: Substitutability for references
- Separates „inheritance“ and „substituabilty“!!!
  - Inseparable in Java. What is better: SQL3 or Java?

```
CREATE ROW TYPE emp(name, salary);
CREATE ROW TYPE manager UNDER emp(bonus);

CREATE TABLE empTable OF emp;
CREATE TABLE manTable OF manager UNDER emptable;
```

Row Types vs. Classes

- Row Types (SQL 3)
  - separate „inheritance“ and „substituability“
  - separate definition of „type“ and „extension“
- Classes (Java)
  - all concepts are lumped together
  - there is only one „extension“ to a type
    - or user-defined using Collection types
    - but, then you lose strong typing in Java
- Not clear what is better!
  - Why did SQL go that route?

Nested Tables (Oracle)

- Table Types. Constructor for table.
- Non-first normal form relations (NF^2) [Schek et al.]

```
CREATE TYPE empSet AS TABLE OF emp;

CREATE TABLE dept(
        no      varchar(100),
        emps  empSet);

SELECT *
FROM deptTable;
```

Nested Tables (Oracle)

- Using a nested table in a from clause
- Example: find manager of Dept 4711

```
SELECT m.name
FROM   THE(   SELECT emps
              FROM   dept
              WHERE  no = „4711“) d,
          manTable m
WHERE m.name = d.name;
```

Nested Tables (Oracle)

- Inserting a new table
  - Cast a regular SQL table into a nested table

```
INSERT INTO dept (name, emps)
VALUES(„007“,
  CAST(MULTISET(
     SELECT   *
     FROM     empTable
     WHERE    salary > 100000)));
```

Functions

A database has three kinds of functions

- Built-in (defined in SQL standard)
  - e.g., +, <, avg, max, is null, ...
- System generated (based on user schema)
  - E.g., casts for distinct types
- User-defined functions (UDFs)
  - scalar functions: sourced and external
  - Methods
  - order functions
  - table functions

Sourced Functions

- Used for distinct types
- Based on the definition of another function

```
CREATE DISTINCT TYPE money as DECIMAL(9,2);

CREATE FUNCTION +(money, money) RETURNS money
                SOURCE + (DECIMAL(), DECIMAL())

CREATE FUNCTION sum(money) RETURNS money
                SOURCE sum(DECIMAL())
```

Sourced Functions: Example

```
CREATE TABLE manager(
        name  varchar(100),
        salary money,
        bonus money
);
```

```
SELECT sum(salary + bonus)
FROM   emp;
```

External Scalar Functions

- Used if SQL is to not expressive enough
- Typical programming languages: C, Java, PL/SQL
- Return a scalar value or instance of a UDT
- Integrated into database server
  - Impacts security, robustness, and performance
- A great deal of fineprint
- Can be used in any kind of query

External Scalar Functions

```
CREATE FUNCTION deltaDays(date, date) RETURNS integer
        EXTERNAL NAME „/home/kossmann/deltaDays.o“
        LANGUAGE C
        DETERMINISTIC
        FENCED                  ...  <other Options>
        PARAMETER STYLE DB2SQL
        NO SQL;
```

Find all orders with more than 10 days of delays

```
SELECT *
FROM order o
WHERE deltaDays(o.promisedShipDate, CURRENTDATE) > 10;
```

Some of the fineprint

- Deterministic or External Action
  - Impacts optimizer: exactly once execution?
- Fenced / Unfenced
  - runs in the same process as DB server
  - Impacts security and performance
- NULL call
  - Defines behavior if NULL is passed as a parameter
- Type compatibility and promotion
  - E.g., smallint -> int
- Scratchpad
  - preserve state between two calls
  - Impacts parallelization

Implementation of the C Function

```
SQL_API_FN deltaDays(
        char * date1In,  /* IN: first date */
        char * date2In,  /* IN: second date */
        int    * out,         /* OUT: result */
        short * nullDate1In, /* IN: date1In = null? */
        short * nullDate2In, /* IN: date2In = null? */
        short * nullOut,  /* OUT: result = null? */
        char * sqlState,   /* OUT: error code */
        char * message   /* OUT: error message */)  {
  ...
}
```

Methods

- Special scalar functions which receive a reference to a row type as a parameter
  - „bound“ to that type
  - Simulates methods in Java, C++, etc.
- Huge differences between systems...

```
CREATE FUNCTION raise (r REF(emp))
        RETURNS euro
        LANGUAGE SQL
        RETURN
          CASE r->status
                WHEN EXEMPT THEN r->salary * 1.2
                 ELSE r->salary * 1.1
        END;
```

Order Functions

- Special compare functions for distinct types
  - Used for <, >, <=, ... of instances of UDTs
- Technically modelled as follows
  - Two methods EQUAL and LESSTHAN
  - EQUAL, LESSTHAN bound to UDT T
  - EQUAL, LESSTHAN take a parameter of Type T
- Again, watch for differences between vendors

Table Functions

- Special external functions: Result is a table
- Can be used in FROM clause
- Implementation as an iterator
  - C function which returns next tuple with every call
- Used to integrate external data
  - E.g., tables from Web pages
- Mechanism and declaration like external scalar fct.

Table Functions

Example: Web search with Google

```
CREATE FUNCTION searchWeb(varchar(100))
        RETURNS TABLE(url varchar(100),
                      score Integer,
                      country varchar(50))
        EXTERNAL NAME „google.o“
        ...
```

Table Functions

Search for all Web pages for Cologne

```
SELECT  w.url
FROM    TABLE(searchWeb(„Köln“)) w
WHERE   w.country = „Germany“
```

How does predicate push-down work here?

- Cost of mixing programming paradigms (C and SQL)

Table Functions

Search for the „Heimatort“ of students.

```
SELECT   s.name, ( SELECT w.url
                   FROM TABLE(searchWeb(s.city)) w)
FROM     Student s;
```

Data Blades, Extenders, ...

- An API to define new data types in the DBMS
  - UDTs become first-class citizens of DBMS
  - Operations and indexes on UDTs become first-class
  - Examples: Time series, Maps, Images, etc.
- Establish market for third-party vendors
  - „app store for DB plug-ins“
- SQL Built-in types
  - Can be seen as pre-defined data blades

Summary

- DBMS vendors add features to their systems
  - Control more and more data
  - Increase lock-in
  - Justify maintenance fees, sell new licenses
- OO + DB Standardization
  - SQL 99 and SQL 3: emerging implementations
  - ODMG
  - None of them a real success
- Where is this going? (personal speculation)
  - leaner DBMS: users do not want to pay for complexity
  - App + DB: personally, I believe in XML + XQuery