Object-Oriented Databases
Chapter 22

Limitations of the Relational Model

- limited constraints expressible
- limited types of relationships
- normalization leads to atomization, inefficiency
- Limited built-in datatypes
  - No support for multimedia types: images, video, sound, designs, texts, etc.
  - BLOBs (binary large objects) are one workaround

Language Fit

- COBOL and DB grew up together
  - COBOL pioneered the "record" construct
  - character-based types
- Poor fit to today's languages like C++

Versioning

- In some applications, old versions of data must be accessible
  - designs (architecture, CAD, etc.)
  - documents
  - multimodule systems
- Often are complex relationships between versions
- Not necessarily an OO concept.

A Look Back

- Before we look ahead...
- Hierarchical and Network (CODASYL) models were popular before relational
  - Network had extremely rich semantics
  - Complex relationships directly expressed (no joins)
  - Primarily "navigational"
    - Custom programs locate data via knowledge of schema, following pointers
    - No standardized query languages

Object-Oriented Trends

- Trends in OO Programming seem promising for databases
  - Rich, user-defined data types (support of new media, lift 1NF restriction)
  - Inheritance (important type of relationship)
  - Encapsulation of data and functions
  - Increasing emphasis on components and reusability; cross-platform
  - Tighter integration with C++
Review of OO Programming Concepts

• Class: description of data structure and operations (i.e., a data type)
  – encapsulation: data and ops are wrapped together; only an interface is externally visible.
• Object: an instance of a class
• Class B inherits from class A: B has all the properties of A, plus some new or altered properties (data/functions)

Strict OO Viewpoint

• Where possible: model the behavior and relationships of the real world
• Everything is an object
• Objects communicate only by passing messages
  – In practice, a message is a function name plus a set of arguments
• Types can be determined at run-time
• Smalltalk is the model: untyped; interpreted; interactive

Hybrids and Compromises

• Example: C++
  – retains all features of non-OO C language, adds classes, inheritance, polymorphism
• OODBs tend to be compromises
  – May retain relational facilities: ORDBMS
  – Add OO features such as: user-defined types & classes, inheritance, etc.
  – Add features like “persistence” and versioning
  – SQL3 will have OO features

OIDs

• Object Identifiers (OID)
• Unique (database-wide) identifier for each object
  – independent of key
• One object can reference another via OID
  – Allows complex embedding

Challenges for Query Languages

• DDL: coordinating PL with QL
• Encapsulation issues
  – how much is visible?
  – must all operations be predefined?
• Multimedia
  – what does "query" mean?
  – how to display results?

Persistence

• The idea: it's easy for a program to work with a complex data structure in memory, but hard to flatten it into a file. It would be convenient if some variables were persistent, i.e., could exist on disk between executions of the program, i.e., be part of the DB.
• Not strictly on OO concept
• One challenge: mapping OIDs between in-memory pointers and disk addresses
  – "pointer swizzling"
Deductive Databases

• Another (non-OO) approach to relieving relational limitations
• DB viewed as a set of facts and rules
  – a row can be viewed as a fact which satisfies a predicate
• Logic-based languages
  – Datalog: DB extension of Prolog
• Excellent at expressing complex constraints, making deductions and discoveries, etc.