CMSC724: What goes around comes around

Amol Deshpande

University of Maryland, College Park

March 11, 2008
Data Modeling

- A data model theory typically involves
  - “structural part” – collection of concepts/constructs to *represent* data
  - “integrity part” – constraints to *ensure data integrity*
  - “manipulation part” – constructs for *manipulating* the data

- We would like it to be:
  - Sufficiently expressive – can capture real-world data well
  - Easy to use
  - Lends itself to good performance
Hierarchical/IMS

Constructs: Hierarchy, keys, record types, DL/1 lang.

Issues:

- Navigational interaction
- No physical data independence
- Repetition of information (m-to-n relationships)
- Inability to represent information
- Last two solved by a latter extension

Some logical data independence
Network/CODASYL

- Constructs: “set” type, network structure
- “programmer as a navigator”

Cons:
- No physical or logical data independence
- Bulk loading
- 3-way relationships
- Very complex programming constructs
Let us illustrate the general effect that the find and get statements have on the program work area. Consider the sample data base of Figure A.16. Suppose that the current state of the program work area of a particular application program is as shown in Figure A.20. Further suppose that a find command is issued to locate the customer record belonging to Johnson. This command causes the following changes to occur in the state of the program work area:

- The current of record type customer now points to the record of Johnson.
- The current of set type depositor now points to the record of Johnson.

We have enclosed part of the query in a while loop, because we do not know in advance how many such customers exist. We exit from the loop when DB-status $\neq 0$. This action indicates that the most recent find duplicate operation failed, implying that we have exhausted all customers residing in Harrison.

A.4.4 Access of Records within a Set

The previous find commands located any database record of type <record type>. In this subsection, we concentrate on find commands that locate records in a particular DBTG set. The set in question is the one that is pointed to by the <set-type> currency pointer. There are three different types of commands. The basic find command is find first <record type> within <set-type> which locates the first member record of type <record type> belonging to the current occurrence of <set-type>. The various ways in which a set can be ordered are discussed in Section A.6.6.

To step through the other members of type <record type> belonging to the set occurrence, we repeatedly execute the following command:

find next <record type> within <set-type>

The find first and find next commands need to specify the record type since a DBTG set can have members of different record types.

As an illustration of how these commands execute, let us construct the DBTG query that prints the total balance of all accounts belonging to Hayes.

```plaintext
customer.customer_city := "Harrison";
find any customer using customer.city;
while DB-status = 0 do
begin
    get customer;
    print (customer.customer_name);
    find duplicate customer using customer.city;
end;
```

```plaintext
sum := 0;
customer.customer.name := "Hayes";
find any customer using customer.city;
find first account within depositor;
while DB-status = 0 do
begin
    get account;
    sum := sum + account.balance;
    find next account within depositor;
end
print( sum);
```
Relational

- Constructs: Relations, relational algebra/calculus, functional dependencies
- Physical and logical data independence
- Cons:
  - Transitive closure
  - (initially) performance
  - (initially) too complex and mathematical languages
Don Chamberlin of IBM was an early CODASYL advocate (later co-invented SQL)

“He (Codd) gave a seminar and a lot of us went to listen to him. This was as I say a revelation for me because Codd had a bunch of queries that were fairly complicated queries and since I’d been studying CODASYL, I could imagine how those queries would have been represented in CODASYL by programs that were five pages long that would navigate through this labyrinth of pointers and stuff. Codd would sort of write them down as one-liners. These would be queries like, "Find the employees who earn more than their managers." [laughter] He just whacked them out and you could sort of read them, and they weren’t complicated at all, and I said, "Wow." This was kind of a conversion experience for me, that I understood what the relational thing was about after that.”
Data Models

- **Entity-relational**
  - Constructs: entity, relationship
  - Limitations: Lack of query language, ease of mapping into relational
  - The conceptual model of choice to design the schema

- **Relational++**
  - Constructs: set-valued attributes, aggregation, generalization etc
  - Limitations: Didn’t prove terribly useful either functionally or performance-wise
Data Models

- Semantic data models
  - Constructs: class, class variable, multiple inheritance etc
- OO
  - Essentially persistent PLs
  - Less impedance mismatch
  - Weak support for Xions, queries etc.
  - Niche market (e.g. CAD)
OR

- Constructs: User-defined functions, types, access methods

Semi-structured

- “Schema last”
- Constructs: DTD, XMLSchema, union types etc
- Issues: Limited applicability ??, doesn’t really solve semantic heterogenuity
- Standard for wire format
Some lessons

- Physical/logical data independence desirable
- Record-at-a-time, navigational interfaces force manual query optimization and won’t scale
- Technical debates settled by the marketplace in many cases
- KISS
- OO: Packages will not sell to users without “major pain”
- User-defined functions and access method effective
- Schema-last is probably a niche market
- Semantic heterogeneity not solved by XML