Outline

1. Databases: A Brief History
2. What goes around comes around
1960’s: Computers finally become attractive, and enterprises start using it. Most applications initially used their own data stores.

Database: coined in military information systems to denote "shared data banks" by multiple applications.

Why?
- Each application had its own format
- Although the data was there, basically unavailable to other programs
  - Often original object code was lost

Instead, define a data format, store it as a "data dictionary", and allow general-purpose "data-base management" software to access it.

Issues:
- How to write data dictionaries? How to access data?
- Disadvantages of integration: integrity, security, privacy concerns
- Who controls the data?
1960’s

- Birth of "hierarchical model" and "network model"
  - Both allowed "connecting" records of different types (e.g., connect "accounts" with "customers")
  - Network model attempted to be very general and flexible —— Charlie Bachman received Turing Award
- IBM designed its IMS hierarchical database in 1966 for the Apollo space program; still around today
  - ".. more than 95 percent of the top Fortune 1000 companies use IMS to process more than 50 billion transactions a day and manage 15 million gigabytes of critical business data" (from IBM Website on IMS)
- Predates "hard disks"

- However, both models exposed too much of the internal data structures/pointers etc to the users
Databases: A Brief History

1970’s: Relational Model
- Origins in Set Theory
- Some early work by D.L. Childs (somewhat forgotten)
- Edgar F. "Ted" Codd: Developed the relational model
  - Elegant, formal model that provided almost complete "data independence"
  - Users didn’t need to worry about how the data was stored, processed etc.
  - High level query language (relational algebra)
  - Notion of normal forms —- Allowed one to reason about and remove redundancies
Databases: A Brief History

- **1970’s: Relational Model**
  - Led to two influential projects: INGRES (UC Berkeley), System R (IBM)
  - Also paved the way for a 1977 startup called "Software Development Laboratories"
    - Didn’t care about IMS/IDMS compatibility (as IBM had to)
- **1976: Peter Chen proposed "Entity-Relationship Model"**
  - Allowed higher-level, conceptual modeling; easier for humans to think about
- **1980: Commercialization/wide-spread acceptance**
  - SQL emerged as a standard, in large part because of IBM’s backing
  - People still sometimes complain about its limitations
- **Late 80’s: Object-oriented, object-relational databases**
  - Enriching the expressive power of relational model
  - Other proposals for semantic data models
Late 80’s, early 90’s
- Many database companies, but starting to consolidate
- Parallel databases beginning to emerge
- Data mining/OLAP (online analytical processing)
- Focus on client tools for application development: PowerBuilder (Sybase), Oracle Developer etc.
- Client-server model becomes popular

Mid/late 90’s
- Web arrives: Growth of middleware that connects web apps to databases
  - Active server pages, Enterprise Java Beans, ColdFusion
- OLAP matures and becomes mainstream
Databases: A Brief History

Early 00’s to mid 00’s
- A sudden boom in data warehousing/analytics
  - Companies like: Aster Data, Greenplum, Vertica, Kickfire, and probably 10 others
  - Some consolidation recently
- Column-stores, analytics, streaming data, become important

Late 00’s
- map-reduce: framework for large-scale data analysis
- key-value stores: framework for “scale-first” data management
- Databases late to react, but have adopted
  - Exploring different design points in integration of databases and map-reduce
  - Transactions and consistency in distributed key-value stores

Next?
Outline

1. Databases: A Brief History
2. What goes around comes around
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.

Two Hierarchical Organizations

![Diagram showing two hierarchical organizations with Supplier and Part entities.]

IMS chose a hierarchical data base because it facilitates a simple data manipulation language, DL/1. Every record in an IMS data base has a hierarchical sequence key (HSK). Basically, an HSK is derived by concatenating the keys of ancestor records, and then adding the key of the current record. HSK defines a natural order of all records in an IMS data base, basically depth-first, left-to-right. DL/1 intimately used HSK order for the semantics of commands. For example, the “get next” command returns the next record in HSK order. Another use of HSK order is the “get next within parent” command, which explores the subtree underneath a given record in HSK order.

Using the first schema, one can find all the red parts supplied by Supplier 16 as:

Get unique Supplier (sno = 16)

Until failure do
  Get next within parent (color = red)
Enddo

The first command finds Supplier 16. Then we iterate through the subtree underneath this record in HSK order, looking for red parts. When the subtree is exhausted, an error is returned.

Notice that DL/1 is a “record-at-a-time” language, whereby the programmer constructs an algorithm for solving his query, and then IMS executes this algorithm. Often there are multiple ways to solve a query. Here is another way to solve the above specification:
Data Models

- Hierarchical/IMS
  - Root records can either be:
    - Stored sequentially
    - Indexed in a B-tree using the key of the record
    - Hashed using the key of the record
  - Dependent records are found from the root using either
    - Physical sequentially
    - Various forms of pointers.
- Leads to heavy physical data dependence
Data Models

- Hierarchical/IMS
  - 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

A CODASYL Network
Figure 5

Supplier (sno, sname, scity, sstate)

Part (pno, pname, psize, pcolor)

Supply(qty, price)

Supplies

Supplied_by
Data Models

- **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 ≠ 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.

\[
\begin{align*}
\text{sum} & := 0; \\
\text{customer.customer.city} & := "Harrison"; \\
\text{find any customer using customer.city}; \\
\text{while DB-status = 0 do} & \begin{align*}
\text{begin} & \begin{align*}
\text{get customer}; & \text{print (customer.customer.name)}; \\
\text{find duplicate customer using customer.city}; & \end{align*} \\
\text{end}; & \end{align*}
\end{align*}
\]

\[
\begin{align*}
\text{sum} & := \text{sum} + \text{account.balance}; \\
\text{find next account within depositor}; & \end{align*}
\]

print( sum);
Data Models

- **Relational**
  - Constructs: Relations, relational algebra/calculus, functional dependencies
  - Physical and logical data independence
  - Cons:
    - Transitive closure
    - (initially) performance
    - (initially) too complex and mathematical languages
Many debates in 1970’s
Relational Model Advocates
  - Nothing as complex as CODASYL can possibly be a good idea
  - CODASYL does not provide acceptable data independence
  - Record-at-a-time programming is too hard to optimize
  - CODASYL and IMS are not flexible enough to easily represent common situations (such as marriage ceremonies)
CODASYL Advocates
  - COBOL programmers cannot possibly understand the new-fangled relational languages
  - It is impossible to implement the relational model efficiently
  - CODASYL can represent tables, so what’s the big deal?
Both camps changed positions to move towards each other
(According to Authors) Effectively settled by mini-computer revolution, and by IBM who announced new relational products
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

![An E-R Diagram](image)

An E-R Diagram
Figure 8
Data Models

- **Relational++**
  - Constructs: set-valued attributes, aggregation, generalization etc
  - Gem [Zaniolo 83]
    - Add to relational model: —- Set-valued attributes —-
      Tuple-reference as a data type and cascaded dot notation
    - Inheritance hierarchies
  - Limitations: Didn’t prove terribly useful either functionally or performance-wise

- **Semantic data models**
  - Constructs: class, class variable, multiple inheritance etc
  - Similar limitations
Data Models

- Object-oriented
  - Designed to get around the impedance mismatch
  - Essentially became persistent PLs
  - Interesting technical challenge: Pointer "swizzling"
  - Weak support for transaction, queries etc.
    - Largely single-user systems
    - DBMS must run in the same address space as the application
- Several reasons didn’t succeed
  - No major additional functionality for most applications
  - No standards
  - Too tied to a single programming language
Data Models

- 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 heterogeneity
  - Standard for wire format

- Javascript Object Notation (JSON)
  - Very similar to XML, and seems to be increasingly replacing it
Data Models

- **RDF: Resource Description Framework**
  - Originally intended as a "metadata data model"
  - Key construct: a "subject-predicate-object" triple
    - E.g., subject=sky - predicate=has-the-color - object=blue
  - Direct mapping to a labeled, directed multi-graph
  - Typically stored in relational databases, or what are called "triple-stores"
  - But some graph database products out there as well (e.g., DEX)
Data Models

- RDF: Resource Description Framework

```
http://www.example.org/staffid/85740

http://www.example.org/terms/address

http://www.example.org/terms/city

Bedford

http://www.example.org/terms/street

1501 Grant Avenue

http://www.example.org/terms/state

Massachusetts

http://www.example.org/terms/postcode

01730
```
Graph Data Models

- Increasing interest in graph databases
- Property Graph Model (used by many open source tools)

Applications on top are quite different though
Array Data Models

- Proposed recently for “Scientific Data Management”
- Stonebraker one of the key proponents
- Key abstract: an “array”
- Key operations include slicing, subsetting, Filtering etc.
- Presumably a better fit for scientific applications
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