CSE544
Data Management

Lecture 3: Data Models
Announcements

• Today: office hour ends at 12:10

• Friday: Homework 1 is due

• Next Monday: MLK day, no lecture

• Start thinking about class projects
References

Data Model Motivation

• Applications need to model real-world data

• User somehow needs to define data to be stored in DBMS

• Data model enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk
Different Types of Data

• **Structured data**
  – All data conforms to a schema. Ex: business data

• **Semistructured data**
  – Some structure in the data but implicit and irregular

• **Unstructured data**
  – No structure in data. Ex: text, sound, video, images

• **Our focus: structured data & relational DBMSs**
Outline

• Early data models
  – IMS
  – CODASYL

• Physical and logical independence in the relational model

• Data models that followed the relational model
Early Proposal 1: IMS*

• What is it?

* IBM Information Management System
Early Proposal 1: IMS*

- **Hierarchical data model**

- **Record**
  - **Type**: collection of named fields with data types
  - **Instance**: must match type definition
  - Each instance has a **key**
  - Record types arranged in a **tree**

- **IMS database** is collection of instances of record types organized in a tree

* IBM Information Management System
IMS Example

- Figure 2 from “What goes around comes around”

What does this mean?
IMS Example

- Figure 2 from “What goes around comes around”

What does this mean?

File on disk:

<table>
<thead>
<tr>
<th>Supp</th>
<th>Part</th>
<th>Part</th>
<th>...</th>
<th>Supp</th>
<th>Part</th>
<th>Part</th>
<th>...</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
</table>

Supplier (sno, sname, scity, sstate)

Part (pno, pname, psize, pcolor)

Part (pno, pname, psize, pcolor, qty, price)

Supplier (sno, sname, scity, sstate, qty, price)
IMS Example

• Figure 2 from “What goes around comes around”

What does this mean?

File on disk:

<table>
<thead>
<tr>
<th>Supp</th>
<th>Part</th>
<th>Part</th>
<th>...</th>
<th>Supp</th>
<th>Part</th>
<th>Part</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part</th>
<th>Supp</th>
<th>Supp</th>
<th>...</th>
<th>Part</th>
<th>Supp</th>
<th>Supp</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
</table>
IMS Limitations
IMS Limitations

• Tree-structured data model
  – Redundant data; existence depends on parent
IMS Limitations

• Tree-structured data model
  – Redundant data; existence depends on parent

• **Record-at-a-time** user interface
  – User must specify algorithm to access data
IMS Limitations

• **Tree-structured data model**
  – Redundant data; existence depends on parent

• **Record-at-a-time** user interface
  – User must specify algorithm to access data

• **Very limited physical independence**
  – Phys. organization limits possible operations
  – Application programs break if organization changes

• **Some logical independence but limited**
Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?
Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

• Each record has a hierarchical sequence key (HSK)

• HSK defines semantics of commands:
  – get_next; get_next_within_parent

• DL/1 is a record-at-a-time language
  – Programmers construct algorithm, worry about optimization
Data storage

How is data physically stored in IMS?
Data storage

How is data physically stored in IMS?

• Root records
  – Stored sequentially (sorted on key)
  – Indexed in a B-tree using the key of the record
  – Hashed using the key of the record

• Dependent records
  – Physically sequential
  – Various forms of pointers

• Selected organizations restrict DL/1 commands
  – No updates allowed due to sequential organization
  – No “get-next” for hashed organization
Data Independence

What is it?
Data Independence

What is it?

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**
Lessons from IMS

- Physical/logical data independence needed
- Tree structure model is restrictive
- Record-at-a-time programming forces user to do optimization
Early Proposal 2: CODASYL

What is it?
Early Proposal 2: CODASYL

What is it?

• **Networked data model**

• Primitives are also **record types with keys**
• Record types are organized into **network**
• Multiple parents; arcs = “sets”
• More flexible than hierarchy

• **Record-at-a-time** data manipulation language
CODASYL Example

- Figure 5 from "What goes around comes around"
CODASYL Limitations

• No data independence: application programs break if organization changes

• Record-at-a-time: “navigate the hyperspace”

The Programmer as Navigator
by Charles W. Bachman
Outline

• Early data models
  – IMS
  – CODASYL

• Physical and logical independence in the relational model

• Data models that followed the relational model
Relational Model Overview

Ted Codd 1970

- What was the motivation? What is the model?
Relational Model Overview

Ted Codd 1970

- Motivation: logical and physical data independence
- Store data in a **simple data structure** (table)
- Access data through **set-at-a-time** language
- No need for physical storage proposal

Relational Database: A Practical Foundation for Productivity
Great Debate

- **Pro relational**
  - What were the arguments?

- **Against relational**
  - What were the arguments?

- How was it settled?
Great Debate

- **Pro relational**
  - CODASYL is too complex
  - No data independence
  - Record-at-a-time hard to optimize
  - Trees/networks not flexible enough

- **Against relational**
  - COBOL programmers cannot understand relational languages
  - Impossible to implement efficiently

- Ultimately settled by the market place
Data Independence

How it is achieved today:

• Physical independence: SQL to Plan

• Logical independence: Views in SQL
Physical Data Independence

• In SQL we express *What* data we want to retrieve

• The optimizers figures out *How* to retrieve it
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
    x.price > 100 and z.city = 'Seattle'

We say What we want
Logical Query Plan

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
    x.price > 100 and z.city = 'Seattle'

We say **What** we want
Physical Query Plan

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
    x.price > 100 and z.city = 'Seattle'
```

We say *What* we want

Says *How* to get it
Query Optimizer

• Rewrite one relational algebra expression to a better one
• Very brief review now, more details next lectures
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and 
  x.price > 100 and z.city = 'Seattle'
Optimization

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid AND y.cid = z.cid AND
    x.price > 100 AND z.city = 'Seattle'
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
      x.price > 100 and z.city = 'Seattle'

More about this next lectures
Logical Data Independence

A View is a Relation defined by a SQL query

It can be used in any SQL query as a normal relation
View Example

View definition:

```
CREATE VIEW Big_Parts AS
SELECT * FROM Part
WHERE psize > 10;
```
View Example

View definition:

```
CREATE VIEW Big_Parts AS
SELECT * FROM Part
WHERE psize > 10;
```

Virtual table:

```
Big_Parts(pno,pname,psize,pcolor)
```
Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)

**View Example**

**View definition:**

```sql
CREATE VIEW Big_Parts AS
SELECT * FROM Part
WHERE psize > 10;
```

**Virtual table:**

Big_Parts(pno, pname, psize, pcolor)

**Querying the view:**

```sql
SELECT *
FROM Big_Parts
WHERE pcolor='blue';
```
Two Types of Views

• Virtual views:
  – Default in SQL, and what Stonebraker means in the paper
  – `CREATE VIEW xyz AS …`
  – Computed at query time

• Materialized views:
  – Some SQL engines support them
  – `CREATE MATERIALIZED VIEW xyz AS`
  – Computed at definition time

• Pros and cons?
Levels of Abstraction

- **External Schema**: a.k.a logical schema, describes stored data in terms of data model, includes storage details, file organization, indexes.
- **Conceptual Schema**: views, access control.
- **Physical Schema**: Disk.

Diagram:

- External Schema to Conceptual Schema
- External Schema to Conceptual Schema
- External Schema to Conceptual Schema
- Conceptual Schema to Physical Schema
- Physical Schema to Disk
Recap

• Physical data independence:
  – Updates to the physical representation do not affect the SQL query
  – Achieved using RA and query optimization

• Logical data independence
  – Updates to the logical schema do not affect SQL query
  – Achieved using views
Outline

• Early data models
  – IMS
  – CODASYL

• Physical and logical independence in the relational model

• Data models that followed the relational model
Other Data Models

• **Entity-Relationship**: 1970’s
  – Successful in logical database design

• **Extended Relational**: 1980’s

• **Semantic**: late 1970’s and 1980’s

• **Object-oriented**: late 1980’s and early 1990’s
  – Address impedance mismatch: relational dbs ↔ OO languages
  – Interesting but ultimately failed (several reasons, see references)

• **Object-relational**: late 1980’s and early 1990’s
  – User-defined types, ops, functions, and access methods

• **Semi-structured**: late 1990’s to the present
Semistructured vs Relational

- Relational data model
  - “Schema first”

- Semistructured data model: XML, Json, Protobuf
  - “Schema last”
  - Hierarchical (trees)
Semistructured, self-describing schema
JSon

Example from: http://www.jsonexample.com/

myObject = {
    "first": "John",
    "last": "Doe",
    "salary": 70000,
    "registered": true,
    "interests": [ "Reading", "Biking", "Hacking" ]
}

Semistructured, self-describing schema
Discussion

• Stonebraker (circa 1998)
  – “schema last” is a niche market

• Today (circa 2020)
  – Major vendors scramble to offer efficient schema discovery while ingesting Json

• Why? What changed?
Discussion

• Stonebraker (circa 1998)
  – “schema last” is a niche market

• Today (circa 2020)
  – Major vendors scramble to offer efficient schema discovery while ingesting Json

• Why? What changed?
  – Today datasets are available in text format, often in Json; ingest first, process later
NoSQL Data Model(s)

- Web boom in the 2000’s created a scalability crises
  - DBMS are single server and don’t scale; e.g. MySQL

- NoSQL answer:
  - “Shard” data, i.e. distribute on AWS
  - Simple data mode: key/value pairs
Key-Value Pair Data Model

• **Data model**: (key, value) pairs
  – Key = string/integer, unique for the entire data
  – Value = can be anything (very complex object)
Key-Value Pair Data Model

• **Data model**: (key, value) pairs
  - Key = string/integer, unique for the entire data
  - Value = can be anything (very complex object)

• **Operations**
  - get(key), put(key, value)
  - Operations on value not supported
Key-Value Pair Data Model

• **Data model**: (key,value) pairs
  – Key = string/integer, unique for the entire data
  – Value = can be anything (very complex object)

• **Operations**
  – get(key), put(key,value)
  – Operations on value not supported

• **Distribution / Partitioning** – w/ hash function
  – No replication: key k is stored at server h(k)
  – 3-way replication: key k stored at h1(k),h2(k),h3(k)
Example

- How would you represent the Flights data as key, value pairs?
Example

• How would you represent the Flights data as key, value pairs?

• Option 1: key=fid, value=entire flight record
Example

• How would you represent the Flights data as key, value pairs?
  
• Option 1: key=fid, value=entire flight record

• Option 2: key=date, value=all flights that day
Example

• How would you represent the Flights data as key, value pairs?

• Option 1: key=fid, value=entire flight record

• Option 2: key=date, value=all flights that day

• Option 3: key=(origin,dest), value=all flights between

How does query processing work?

No physical data independence!
Conclusion

• Data model: a formalism to describe/query the data

• Relational data model: tables+relational language; no description of physical store

• Data independence: efficiency needs to be realized separately, by the query optimizer

• Many competing “more efficient” data models have been proposed, and will be proposed, but fail because of lack of data independence