CSE 544
Principles of Database Management Systems

Magdalena Balazinska

Fall 2007

Lecture 2 - Early data models

Announcements

- Remember to email us your team information today
- You do not need to pick a project today
- But you need to pick a project this week
- Schedule an appointment with magda to discuss your project (end of this week or early next week)
- Project proposals due on October 10th

References

- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed. Sections 1 through 4.
- R&G Book. Chapter 3: "The relational model"

Outline

- Different types of data

- Early data models
  - IMS
  - CODASYL

- Relational model (introduction)

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
  - Ex: resume, ads
- **Unstructured data**
  - No structure in data. Ex: text, sound, video, images

- Our focus: structured data & relational DBMSs
- Other types of data toward end of quarter

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Relational model (introduction)

Early Proposal 1: IMS

- **Hierarchical data model**

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

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

IMS Example

- See Figure 2 in paper “What goes around comes around”

Data Manipulation Language: DL/1

- Each record has a hierarchical sequence key (HSK)
  - Records are totally ordered: depth-first and left-to-right
- HSK defines semantics of commands:
  - get_next
  - get_next_within_parent
- **DL/1 is a record-at-a-time language**
  - Programmer constructs an algorithm for solving the query
  - Programmer must worry about query optimization

Data storage

- 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 with sequential organization
  - No “get-next” for hashed organization

Data Independence

- **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**
- **Why are these properties important?**
  - Reduce program maintenance as
  - Logical database design changes over time
  - Physical database design tuned for performance

IMS Limitations

- **Tree-structured data model**
  - **Redundant** data, existence **depends on parent, artificial structure**
- **Record-at-a-time** user interface
  - User must specifies **algorithm** to access data
- **Very limited physical independence**
  - Phys. organization limits possible operations
  - Application programs break if organization changes
- Provides **some logical independence**
  - DL/1 program runs on logical database

Early Proposal 2: CODASYL

- **Networked data model**

- Primitives are also **record types** with **keys**
- Record types are organized into **network**
  - A record can have multiple parents
  - Arcs between records are named
  - At least one entry point to the network
- Network model is **more flexible than hierarchy**
  - Ex: no existence dependence
- **Record-at-a-time** data manipulation language

CODASYL Example

- See Figure 5 in paper “What goes around comes around”

CODASYL Limitations

- **No physical data independence**
  - Application programs break if organization changes
- **No logical data independence**
  - Application programs break if organization changes
- Very **complex**
- Programs must "**navigate** the hyperspace"
- Load and recover as **one gigantic object**

Outline

- Different types of data
- Early data models
  - IMS
  - CODASYL
- Relational model (introduction)

Relational Model Overview

- Proposed by Ted Codd in 1970
- Motivation: better logical and physical data independence
- Overview
  - Store data in a **simple data structure** (table)
  - Access data through **set-at-a-time** language
  - **No need for physical storage proposal**

Relation Definition

- **Database is collection of relations**
- **Relation R is subset of S_1 x S_2 x … x S_n**
  - Where $\mathbf{S_i}$ is the domain of attribute **i**
  - **n** is number of attributes of the relation
- Relation is basically a table with rows & columns
  - SQL uses word table to refer to relations

Properties of a Relation

- Each row represents an n-tuple of R
- Ordering of rows is immaterial
- All rows are distinct
- Ordering of columns is significant
  - Because two columns can have same domain
  - But columns are labeled so
  - Applications need not worry about order
  - They can simply use the names
- Domain of each column is a primitive type

- Relation consists of a relation schema and instance

More Definitions

- **Relation schema**: describes column heads
  - Relation name
  - Name of each field (or column, or attribute)
  - Domain of each field
- **Degree (or arity) of relation**: nb attributes
- **Database schema**: set of all relation schemas

Even More Definitions

- **Relation instance**: concrete table content
  - Set of tuples (also called records) matching the schema
- **Cardinality of relation instance**: nb tuples
- **Database instance**: set of all relation instances

Example

- Relation schema

  Supplier(<u>sno: integer</u>, sname: string, scity: string, sstate: string)

- Relation instance

sno	sname	scity	sstate
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

Integrity Constraints

- **Integrity constraint**
  - condition specified on a database schema
  - restricts data that can be stored in db instance
- DBMS enforces integrity constraints
  - Ensures only legal database instances exist
- Simplest form of constraint is domain constraint
  - Attribute values must come from attribute domain

Key Constraints

- **Key constraint:** “certain minimal subset of fields is a unique identifier for a tuple”

- **Candidate key**
  – Minimal set of fields
  – That uniquely identify each tuple in a relation

- **Primary key**
  – One candidate key can be selected as primary key

Foreign Key Constraints

- A relation can refer to a tuple in another relation
- **Foreign key**
  - Field that refers to tuples in another relation
  - Typically, this field refers to the primary key of other relation
  - Can pick another field as well

Key Constraint SQL Examples

```
CREATE TABLE Part (
  pno integer,
  pname varchar(20),
  psize integer,
  pcolor varchar(20),
  PRIMARY KEY (pno)
);
```

Key Constraint SQL Examples

```
CREATE TABLE Supply(
  sno integer,
  pno integer,
  qty integer,
  price integer
);
```

Key Constraint SQL Examples

```
CREATE TABLE Supply(
  sno integer,
  pno integer,
  qty integer,
  price integer,
  PRIMARY KEY (sno,pno)
);
```

Key Constraint SQL Examples

```
CREATE TABLE Supply(
  sno integer,
  pno integer,
  qty integer,
  price integer,
  PRIMARY KEY (sno,pno),
  FOREIGN KEY (sno) REFERENCES Supplier
  FOREIGN KEY (pno) REFERENCES Part
);
```

Key Constraint SQL Examples

```
CREATE TABLE Supply(
   sno integer,
   pno integer,
   qty integer,
   price integer,
   PRIMARY KEY (sno,pno),
   FOREIGN KEY (sno) REFERENCES Supplier
                              ON DELETE NO ACTION,
   FOREIGN KEY (pno) REFERENCES Part
                              ON DELETE CASCADE
);
```

General Constraints

- Table constraints serve to express complex constraints over a single table

```
CREATE TABLE Part (
  pno integer,
  pname varchar(20),
  psize integer,
  pcolor varchar(20),
  PRIMARY KEY (pno),
  CHECK ( psize > 0 )
);
```

- It is also possible to create constraints over many tables

Summary

- Data independence is desirable
  - Both physical and logical
  - Early data models provided very limited data independence
  - Relational model facilitates data independence
    - Set-at-a-time languages facilitate phys. indep. [more next lecture]
    - Simple data models facilitate logical indep. [more next lecture]

- Flat models are also simpler, more flexible

- User should specify what they want
  - Not how to get it
  - Query optimizer does better job than human