3 data models from the 1970s

Hierarchical (IMS + DL/1)
Network (Codasyl + Codasyl DML)
Relations (Codd proposal + DL/alpha
Relational algebra
SQL
Quel)

Themes:

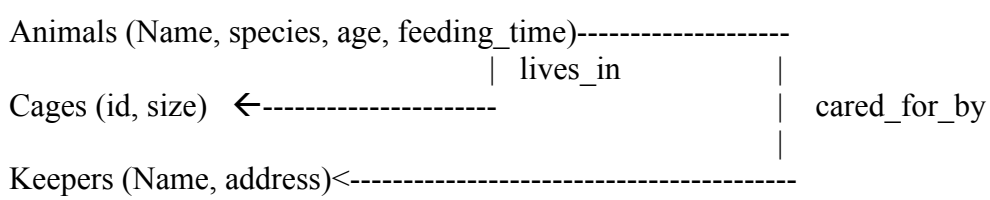
- Data redundancy
- Physical data independence
- Logical data independence
- High level language

Why study ancient history?

> "Those that do not understand the mistakes of their ancestors will end up repeating them"

Use Zoo example (with one more kind of object)

3 objects
2 relationships



Each animal in ONE cage, multiple animals can share a cage
Each animal cared for by ONE keeper, a keeper cares for multiple animals

---

IMS (IBM 1968)

Segment types (record types)
Segments (instances)

Schema (hierarchical collection of segment types – must be a tree)

Possible schemas

1)

2)

Instance of 1)

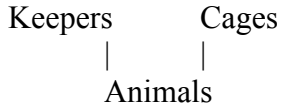
Sam
- Freddie
  - 1
- Jimmy
  - 2
- Sally
  - 1

All have redundancy!

1) repeat cage info for animals which share a cage
2) repeat cage info for animals in a shared cage with different keepers

Bad: possibility of inconsistency

Fundamental problem:



Cannot be represented as a hierarchy!

IMS Storage

Root	dependents
*****	*********
Sequential	Sequential
Index	Sequential
Hash	Pointer spaghetti
Index	Pointer spaghetti

Note: no indexes on dependent segments!

DL/1

Every segment has a hierarchical sequential key (HSK)

Key of the segment, prepended by keys of path back to the root

All segments are logically in HSK order (for purposes of DL/1)

Commands

GU [segment type] [predicate]
GN
GNP
D
I

Find all cages that Sam enters:

GU Keepers (name = 'Sam')
Until no more
GNP Cages

Find the keepers that enter cage 6

GU Keepers
GNP Cages (id = 6)

Until no more
GN Keepers
GNP Cages (id = 6)

Notes: GU is really get first

Some commands are fast; some are slow; depends on the storage chosen and the schema chosen. IMS wizards make gobs of money; even today.

IMS problems:

a) duplication of data (we talked about this above)

b) painful low level programming interface – have to program the search algorithm

c) limited physical data independence

change root from indexed to hash --- programs that do GN on the root segment will fail

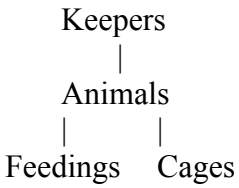
cannot do inserts into sequential root structure

maintenance required if you change the tuning knobs! Hard to tell how bad it will be.

One of Codd's key points!!! This is terrible system design

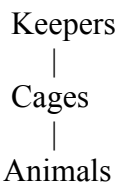
d) limited logical data independence

Zoo acquires a pair of Pandas – they have 2 feeding times!



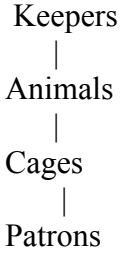
Must change the schema!

As a cost cutting measure, Zoo management decides a keeper will be responsible for a cage – and all the animals in that cage.



Should change the schema to match the business problem

Management decides to have "patrons" who buy cages



Schemas change for all these reasons – plus

Feds change the rules (OSHA)
Tax rules change (IRS)
Merge with another zoo

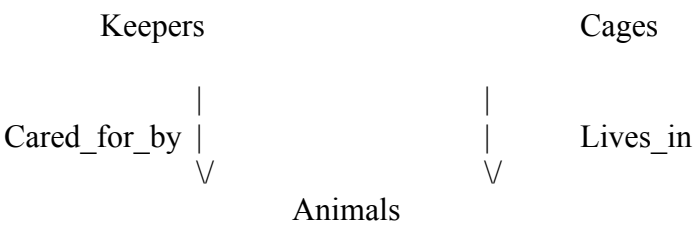
Whenever the logical schema changes – you need program maintenance – unknown complexity!! In the worst case, toss everything; begin again!!

This was what was motivating Codd –

Codasyl (Committee on Data Systems Languages)

BF Goodrich (the guys without the blimp) built a prototype
Commercialized by Cullinet Corp. as IDS
Codasyl committee wrote standardization documents closely linked to IDS

Record types (like IMS)
Connected by named Sets (1::n relationships)
Arranged in a graph



Models network data directly. Ought to be better than IMS.

Record types
1) hashed
2) clustered with the owner record in some set

Pointer spaghetti for set implementation

If a record is not hashed, then it can only be accessed through set membership
.

Codasyl DML

Find the cages that Sam enters

```
Find Keepers (name = 'Sam')
Until nomore
        Find next Animal in Cared_for_by
        Find parent in Lives_in
```

IMS has current segment; current parent (2 pins)

Codasyl has

Current of app
Current of every record type
Current of every set type

(6 pins)

Programming is
Find an entry point
Navigate in N-D space

For a defense of this programming style see 1973 Turing award lecture by Charlie Bachmann.

Codasyl issues

Horrible complexity.

No physical data independence – change most anything → recode

No logical data independence – change most anything -> recode

If you screw up the data structure inadvertently, then must reload everything. (No isolation)

Initial load must be done all at once. – many hours.

Codd: relations

Unordered collections of tuples

Animals (name, species, age, feeding_time)
Cages (id, size)
Keepers (id, name, address)
****

Animals (name, species, age, feeding_time, cid, kid)
Cages (id, size)
Keepers (id, name, address)

Or

Animals (name, species, age, feeding_time)
Cages (id, size)
Keepers (id, name, address)
Lives_in (aname, cid)
Cared_for_by (aname, kid)

Data base design problem – which one to choose

High level language for access (physical data independence)
What you want not how to get it

Codd's proposals (he did cellular automata previously)

Data language alpha (basically 1st order predicate calculus)
Relational algebra (a collection of operations chained together – APL style)

No mere mortals could understand either language

SQL (and Quel) were much more accessible

Find the cages that Sam touches

```
Select cid
From Animals
Where zid in
        Select id
        From Keepers
        Where name = 'Sam'
```

Complete physical data independence

Eliminates redundancy

Better change at logical data independence (views)

```
Define view (Sam_Cages) as
        Select cid
        From Animals
        Where zid in
                Select id
                From Keepers
                Where name = 'Sam'

Select …
From Sam_Cages
Where …
```

All queries and many updates can be supported on views. Interested reader is referred to SIGMOD 1976 and a litany of papers that have followed.

Debate raged throughout the 1970's over:

Efficiency
Programmability of high level languages
Cobol

Won hands down by relational model.

6.830 / 6.814 Database Systems

Fall 2010