

Today:
 Data models
 Relational model
 Relational algebra
 Data Independence
 Data anomalies

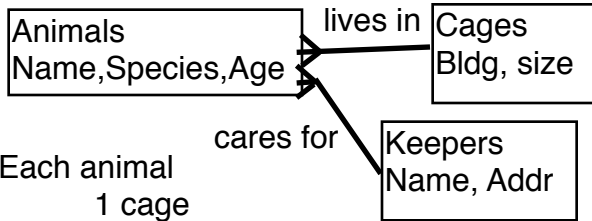
Early models

IMS
 Codasyl
 Relational

Relational calculus

Normalization

Zoo Example
 Animals, Cages, Keepers



Each animal
 1 cage
 1 keeper

9/15/09

PS1 out. Late policy. Check for announcements.
 Office hours. (Monday @11, Tues @4 -- 32 G9 Lounge)

Database prehistory -- how did we end up with the relational model

Data processing one of the first applications of computers
 Many early languages

By the 1970's, several systems had been existence for many years -- in particular, IMS (In-formation Management System) developed in the Mid 60's, and CODASYL, developed in the late 60's/early 70's as a replacement.

Zoo example:

Animals, cages, keepers
 Each animal lives in 1 cage, cared for by 1 keeper
 Keepers keep multiple cages
 Cages contain multiple animals

IMS (as originally proposed)

hierarchical data model -- can't represent fact that animals have 2 parents

What does this imply?

Redundancy (e.g.,
 repeated cage info for animals in same cage w/ different keepers or
 repeated cage info for animals which share a cage in #1

IMS programming model:

each segment (record) has a "hierarchical segment key" -- HSK,
 which is key of segment, plus keys of parents in hierarchy
 e.g., sam, 1, jimmy

GU -- get unique
 GN -- get next
 GNP -- get next parent

IMS --

Segment types (classes of records)

Segments (instances)

Schema -- hierarchical

Keepers	Keepers	Sam (32 vassar)
		1 (3, 50 sq ft)
Animals	Cages	Jimmy (antelope)
		2 (4, 10 sq ft)
Cages	Animals	Sally (salam.)

IMS storage:

root can be
 sorted sequential
 sorted index
 hashed
 (dependents are always unsorted)

IMS Programming

GU -- get unique -- find a root
 GN iterate through nodes at same level
 GNP iterate through children of last root

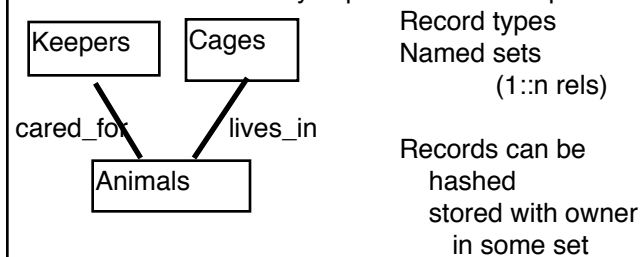
Cages sam keeps: GU keepers (name = 'sam')
 Until no more: GNP Cages

IMS problems:

- duplicate data
- low level programming interface (search algo)
- lack of physical data independence
 - can't use GN on hashed roots
 - insert on sequential root disallowed (changes to physical structure change program!)
- lack of logical data independence
 - changes to the schema
 - require me to rewrite programs

Codasyll:

Network model -- directly captures relationships



Keepers (hash)

Cages, animals stored with keepers

Find cages sam keeps:

Find keepers (name = 'sam')

Find next Animal in cared_for

Find parent in lives_in

Programming model -- navigation in N-dimensional space (yucky)

CODASYL Limitations

- Programming model
- No physical data independence (changing record rep can require code changes)
- No logical data independence

All of these fixed in later versions, after Codd's arguments

IMS has many problems

Why is data independence important?

Schemas will change; performance considerations will change; programs with queries in them may last a long time; don't want to rewrite / recompile

Later versions address these limitations, adding general networks, some form of logical data independence, etc. -- programming model is still awful.

Codaysyl

Network model

Better fit for data than IMS

Avoid redundancy problems

Not much better in other respects

- Ugly programming model where user navigates between records
 - still explicitly encoding algorithm
 - programmer has to keep track of where they are in database

(Example)

- Still no physical or logical data independence

Relational model

Tables

- Rows unordered
- Columns ordered
- No duplicate rows
- Primitive fields

Schema

Keys -- Primary, Foreign

Example (zoo)

kid	kname	address	cid	bdlg	size
1	sam	1 main st	1	1	50
2	robert	1 lake st.	2	2	5
3	nizam	1 south st.			

keepers

aid	aname	a_kid	a_cid
1	jimmy	1	1
2	jenny	2	1
3	sally	1	2

animals

cages

Relational ops

$\pi_c(A)$ - columns c of A
(example) $\pi_{1,2}$ (keepers)

$sel_{pred}(A)$

filter A according to pred

$join_{pred}(A)$

$sel_A(a \times b)$ (example)

How does the relational model improve things?

Simple representation (tables)

High level query languages don't require "navigating"

Also provide fixes for data independence issues

Simple set of operators

High level languages can be used with other data models as well!

We will return to issue of data dependency.

Properties

- All data is described by "relations", which are tables of records, or "tuples"
- Tables are unordered sets (no duplicates in relational model)
- Database is one or more tables
- Each relation has a "schema" which describes the types of the columns, or fields
- Each field is a primitive type -- in particular, it is not a set/relation

"keys" allow us to manage the relationship between records

- primary key : unique identifier for a particular record
- foreign key : reference to a primary key in another table (example)

(These relationships are explicitly enforced by database, typically)

When this model was proposed, it was very controversial:

- Hard to follow (Codd was a mathematician, people didn't get it)
- No one believed it could be made efficient

UNION (A,B)
 DIFFERENCE (A,B)
 CROSS PRODUCT

INTERSECTION (A,B)
 $R \cup S - (R - S) - (S - R)$

JOIN
 Defined as CP + select
 Aggregates not originally a part of the model

Algebra is closed
 Algebraic xforms

join: commutative : $A \text{ join } B = B \text{ join } A$
 associative: $(A \text{ j } B) \text{ j } C = A \text{ j } (B \text{ j } C)$
 sel: commutative: $Sa(Sb(A)) = Sa \text{ and } b(A) = Sb(Sa(A))$
 sel push: $Sa(A \text{ j } b) = Sa(A) \text{ j } b$
 proj push: $Pra (A \text{ j } b) = Pra(A) \text{ j } b$

Find size of cages sam keeps
 $\pi_{\text{size}} ((\sigma_{\text{kname}='sam'} \text{keepers}) \text{ j }_{\text{kid}=\text{a_kid}} \text{animals}) \text{ j }_{\text{cid}=\text{a_cid}} \text{cages})$

Data independence

Physical: change representation on disk without breaking code

Logical: change schema without breaking code

Physical -- programs written in terms of tables, not physical rep

Logical -- can change schema

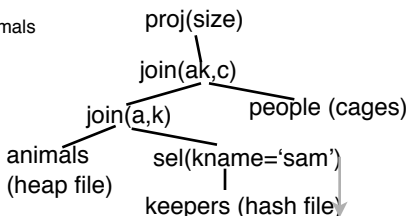
Relational

Tables \neq physical layout

Example -- Cages sam keeps

$\pi_{\text{size}} ((\sigma_{\text{kname}='sam'} \text{keepers}) \text{ j }_{\text{kid}=\text{a_kid}} \text{animals}) \text{ j }_{\text{cid}=\text{a_cid}} \text{cages})$

SELECT size
 FROM cages, keepers, animals
 WHERE kname='sam'
 AND a_kid = kid
 AND a_cid = cid



Ops:

Projection

$\pi_c(A)$ = columns c of table A
 (eliminate duplicates)

Selection

Join

several defs:

natural join?

$(a,b) \text{ join } (b,c) = (a,b,c)$ -- glue together on shared attribute

more generally

$A \text{ join } B \text{ p} = \text{sel}_p(a \times b)$

Other ops (union, diff, cross product, intersection)

Operations all compose with each other -- by def, an algebra is "closed", meaning that any operation on a relation produces a relation

Some operations also commute

Show example of relation expression for cages sam keeps

How does the relational model provide physical independence?

No specification of what storage looks like at all -- users interact with operators on tables.

For example, table might be sorted or hashed. Not visible to user.

So how does the user take advantage of this fact?

User isn't supposed to know.

Administrator tries to layout data on disk in a way that is good for user's queries.

Query optimizer picks best "access method" for query at hand

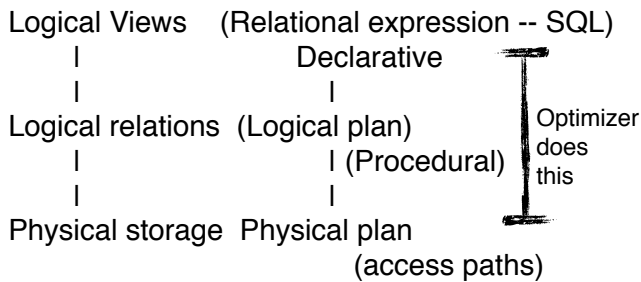
Relational calculus
SQL is an instantiation (example)

Equivalent expressiveness as algebra
- Declarative (not procedural)
- Add'l. independence
(Best plan may change w/ layout)

Views \neq actual schema

create view my_hobbies as
(select ...)

query rewriting SQL(view) \rightarrow SQL (table)
always possible



How does the relational model provide logical independence?

User interacts with logical **views**, that aren't necessarily the same as the physical schema (example)

change tables so that each keeper keeps one cage

alter table cages add column c_kid int refs keepers.kid

select sid,name,a_cid into animals2 from animals

drop table animals

create view animals as (select sid, name, a_cid, c_kid from animals, cages where a_cid = cid)

IMS was later changed to provide both of these.

Don't views hurt performance?

Yes. (Hence, materialized views.)

Don't those hurt performance too (yes, update performance)

Data independence == layer of indirection

Generally bad for performance, good for maintainability

(This is what Codd valued above all else.)

Hierarchy of view \rightarrow relations \rightarrow storage

Calculus \rightarrow algebra \rightarrow physical plan

Recap why relational model is important, Codd visionary but dense.