

## Lecture 4

9/22/09

**HW 1 Due Today. How was it?**

**Lab 1 will be posted this evening. Due Oct 1 (next Thursday!) Tutorial times: Th @ 8pm? Can you guys make it?**

Project partners due a week from today.

### What makes a good decomposition?

Our goal is to decompose a given schema into a collection of schemas that satisfy BCNF (usually).

Want a **lossless** decomposition, which means that:

for all instances  $r$  of  $R$  that satisfy FDs, a lossless decomposition of  $r$  into  $x$  and  $y$  will have the property that

$x \text{ join } y = r$  (e.g., don't put names and address in different tables -- couldn't reconnect them)

Also might want decompositions to be **dependency preserving**.

This is true only if the closure of the union of the dependencies of all the decomposed relations is equivalent to the original set of dependencies.

Important so that we can check semantics of data (e.g., all hobbies have the same cost)

Is decomposition of hobbies dependency preserving? (yes)

Is BCNF always dependency preserving? (no!)

Example

Account	Client	Office
a	joe	1
b	mary	1
a	john	1
c	joe	2

Client, Office  $\rightarrow$ , Account  
Account  $\rightarrow$  Office

Key is C, O

has an insertion anomaly -- change the office that an account belongs to, and we have to change a number of records

A  $\rightarrow$  O violates BCNF

BCNF decomposition is:

Account	Office (Account $\rightarrow$ Office)
Account	Client ( )

### Is this dependency preserving?

Now, it's BCNF, but we lost our primary key constraint that client and office uniquely determine C, O, A.

This is problematic.

So what can we do? No perfectly satisfying answer, but one thing we can do is introduce a new normal form, 3NF.

- 3NF :

Also allows FDs  $X \twoheadrightarrow Y$  such that

Y is a subset of K for some key K of R (not that Y is a part of a superkey!)

Acct, Client, Office

Client, Office  $\rightarrow$  Acct

Acct  $\rightarrow$  Office

key Client, Office

is in 3NF because Office is a subset of the key {Client, Office}

3NF keeps FDs that BCNF might throw away, to preserve dependencies. This introduces redundancy.

In general, always possible to find a 3NF decomposition that is dependency preserving, but as we saw, it may still have some redundancy. So there's a tradeoff between which normal form is better.

Any relation in BCNF is also in 3NF (but not vice versa).

### Denormalization. When and where do we want to do it?

Do we always want to decompose a relation? Why or why not?

Generally speaking, decomposition :

- decreases storage overhead by eliminating redundancy and
- increases query costs by adding joins.

This isn't always true! Sometimes it increases storage overhead or decreases query costs.

Sometimes (for performance issues) you don't want to decompose.

### So how much does this really matter?

Eliminating redundancy really is important.

Adding lots of joins can really screw performance.

These two are sometimes at odds with each other.

In practice, what people do is what we did for hobbies -- think about entities, join them on keys. "Entity relationship" model provides a way to do this (see Stonebraker paper, or book). Will result in something in BCNF.

### Today: Relational Database Systems

probably doesn't all make sense right now -- you should look at both of these papers through the semester for context

system R -

history lesson (stolen from <http://www.cs.berkeley.edu/~brewer/cs262/SystemR.html>)

# UCB 1974-77

- \* a "pickup team", including Stonebraker & Wong. early and pioneering. begat Ingres Corp (CA), CA-Universe, Britton-Lee, Sybase, MS SQL Server, Wang's PACE, Tandem Non-Stop SQL.

# System R : IBM San Jose (now Almaden)

- \* 15 PhDs. begat IBM's SQL/DS & DB2, Oracle, HP's Allbase, Tandem Non-Stop SQL. System R arguably got more stuff "right", though there was lots of information passing between both groups

- \* Jim Gray: Turing Award #22 (1998)

- \* Lots of Berkeley folks on the System R team, including Gray (1st CS PhD @ Berkeley), Bruce Lindsay, Irv Traiger, Paul McJones, Mike Blasgen, Mario Schkolnick, Bob Selinger, Bob Yost. See

# early 80's: commercialization of relational systems

- \* Ellison's Oracle beats IBM to market by reading white papers.

- \* IBM releases multiple RDBMSs, settles down to DB2. Gray (System R), Jerry Held (Ingres) and others join Tandem (Non-Stop SQL), Kapali Eswaran starts EsVal, which begets HP Allbase and Cullinet

- \* Relational Technology Inc (Ingres Corp), Britton-Lee/Sybase, Wang PACE grow out of Ingres group

- \* CA releases CA-Universe, a commercialization of Ingres

- \* Informix started by Cal alum Roger Sippl (no pedigree to research).

- \* Teradata started by some Cal Tech alums, based on proprietary networking technology (no pedigree to software research, though see parallel DBMS discussion next semester!)

# mid 80's: SQL becomes "intergalactic standard".

- \* DB2 becomes IBM's flagship product.

1990's:

- \* Postgres project at UC Berkeley

- \* turned into successful open source project by a large community, mostly driven by a group in russia

- \* Illustra --> Informix --> IBM

- \* MySQL

2000's:

- \* Postgres --> Netezza, Vertica, Greenplum, ...

- \* MySQL --> Infobright

- \* Ingres --> DATAlegro ...

System R is generally considered the more influential of the two -- you can see how many of the things they proposed are still in a database system today. However, Ingres probably had more "impact" by virtue of training a bunch of grad students who went on to fund companies + build products (e.g., BerkeleyDB, Postgres, etc.)

Stuff System R got wrong:

- o shadow paging (backup page maps and pages for recovery)
- o never really used links in the RDS
- o rejected hashing

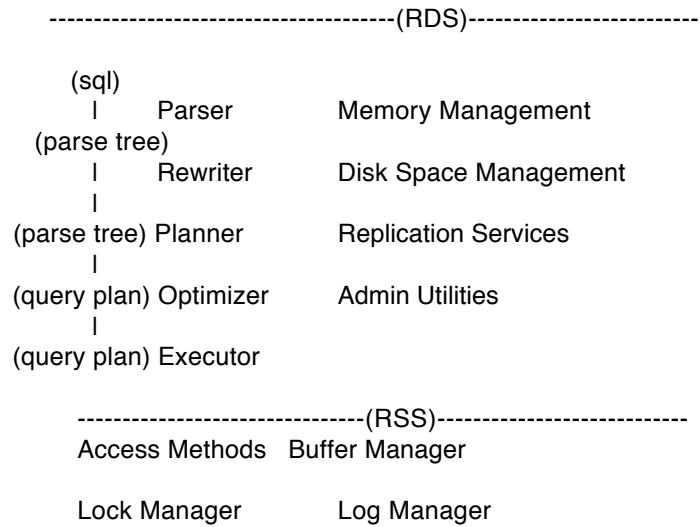
~\$20 billion/year industry now

### Anatomy of a database system

Major Components:

Admission Control

Connection Management



show flow of a query:

skip multiprocessor stuff for now -- we will revisit when we talk about parallel and distributed dbs

process models --

why does this matter? (what happens if you get it wrong)

process per connection (why is this bad? why is it good? how does memory management work?)

process per server, w/ worker threads

additional processes for asynchronous I/O

how many threads/processes should i have?

- dispatch thread
- worker threads
- I/O for asynchrony

### **query processing**

step 0: admission control / authorization

step 1 : query rewrite (logical optimization)

view rewrite example

```
create view sals as (  
    select dept, avg(sal) as sal  
    from emp  
    group by dept  
)
```

emp : id, sal, age, dept

```
select sal from sals where dept = 'eecs';
```

```
select sal from  
(  
    select dept, avg(sal) as sal  
    from emp  
    group by dept  
)  
where dept = 'eecs';
```

constant elimination, logical predicates, etc.

e.g., WHERE sal > 1000 + 4000

predicate injection based on constraints

removal of redundant predicates

subquery flattening

```
select avg(sal) as sal  
from emp  
where dept='eecs'  
group by dept
```

a little tricky; suppose view was:

```
create view sals as {  
    select distinct dept, sal  
    from emp  
}
```

and query was

```
select avg(sal) from sals
```

is equivalent to:

```
select avg(sal) from (  
    select distinct dept, sal  
    from emp  
)
```

but can we flatten more?

e.g., to

```
select avg(distinct sal
```

no! why? duplicates:

```
eid, dept, sal  
1, eecs, 100K  
2, eecs, 100K  
3, me, 50K  
4, arch, 50K
```

average = 75 K

sals:

```
eecs,100K  
me,50K  
arch, 50K
```

average = 66 K

"rule based optimizer" -- situations in which subqueries can be merged!  
(see 'query rewrite rules in IBM DB2 Universal Database')

step 2 : plan formation (SQL -> relational algebra)

notation

```
emp (eno, ename, sal, dno)  
dept (dno, dname, bldg)  
kids (kno, eno, kname, bday)
```

```
select ename, count(*)  
from emp, dept, kids  
and emp.dno=dept.dno  
and kids.eno=emp.eno  
and emp.sal > 50000  
and dept.name = 'eecs'
```

group by ename  
having count(\*) > 7

relational algebra:

query plan:

generating the best plan is the job of the optimizer

physical storage:

all records are stored in a region on disk ("extent" in system R); probably easiest to just think of each table being in a file in the file system.

tuples are arranged in pages in some order --> "heap file"

*access path* is a way to access these tuples on disk.

several alternatives:

**heap scan**

**index scan ("image" in system R)** provide an efficient way to find particular tuples.

**link** -- connection between tuples in two different files (not going to discuss)

what is an index? what does it do?

insert (key, recordid) --> points from a key to a record on disk

{records} = lookup (key)

{records} = lookup ([lowkey ... highkey])

hierarchical indices are the most common type used-- e.g., B-Trees

indices typically point from key values to records in the heap file

diagram:

typically, in a database, indices are keyed on a particular attribute (e.g., employee salary), which allows efficient lookup on that attribute. values are just the tuples themselves.

what does it mean to "cluster" an index? (arrange keys on disk so that they are in order of index)

why is that good?

typically, an access path also supports a "scan" operation, that allows access to all tuples in the table.

b/c a given lookup or scan can return lots of tuples, most database indices use an "iterator" abstraction:

```
it = am.open(predicate)
loop:
    tup = it.get_next()
```

we can place different access methods at the leaves of query plans:

want to talk about optimization; before we do that, let's develop a simple model of how a query is executed:

### **step 3 : query execution**

database query plans -- iterator model

```
init(sarg)
next()
close()
```

examples

iterators can be arbitrarily re-ordered

### **"query optimization" -- spend a lot more time on this**

joe doesn't really tell you how this is done -- just says how the world differs from the selinger paper (which we will read next time).

considerations:

- 1) ordering of operations, (example)
- 2) the selection of access methods, and (example)
- 3) the choice of algorithms (example)

how to do this?

simplest idea is what System R originally proposed -- heuristic evaluation.

These are just rules -- e.g., "when there is an equality predicate, use and image that supports equality lookup".

example?

what could go wrong?

- no statistics, estimation could be incorrect
- rules aren't always valid
- what about complex joins?
  - putting an explosive join at the bottom of the plan is a bad idea

### **plan types:**

left deep vs. bushy

(discuss pipelining)

pipelining -- means that results of one operator can be fed into another operator

Buffer management and storage system:

what is the purpose of the buffer pool or buffer manager? why is it needed?

cache for memory access

way to explicitly control what is on disk and what is in memory -- important for committing transactions

convenient "bottleneck" through which references to underlying pages go  
useful when checking to see if locks can be acquired or not

How does it work?