

Relational Databases

Borrowed from G. Ulman Lecture Notes

What *Is* a Relational Database Management System ?

Database Management System = DBMS

Relational DBMS = RDBMS

- A collection of files that store the data
- A big C program written by someone else that accesses and updates those files for you

Example of a Traditional Database Application

Suppose we are building a system
to store the information about:

- students
- courses
- professors
- who takes what, who teaches what

Can we do it without a DBMS ?

Sure we can! Start by storing the data in files:

students.txt courses.txt professors.txt



Now write C or Java programs to implement specific tasks

Doing it without a DBMS...

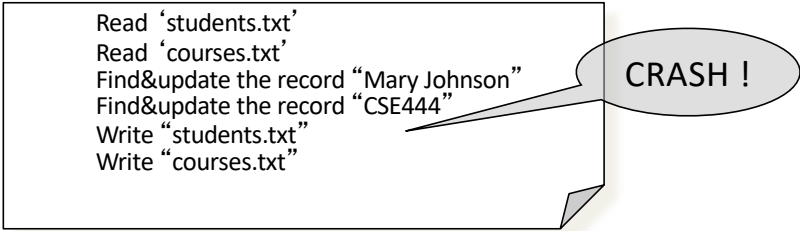
- Enroll “Mary Johnson” in “CSE444”:

Write a C program to do the following:

```
Read 'students.txt'  
Read 'courses.txt'  
Find&update the record "Mary Johnson"  
Find&update the record "CSE444"  
Write "students.txt"  
Write "courses.txt"
```

Problems without an DBMS...

- System crashes:



```
Read 'students.txt'  
Read 'courses.txt'  
Find&update the record "Mary Johnson"  
Find&update the record "CSE444"  
Write "students.txt"  
Write "courses.txt"
```

CRASH !

- What is the problem ?

- Large data sets (say 50GB)

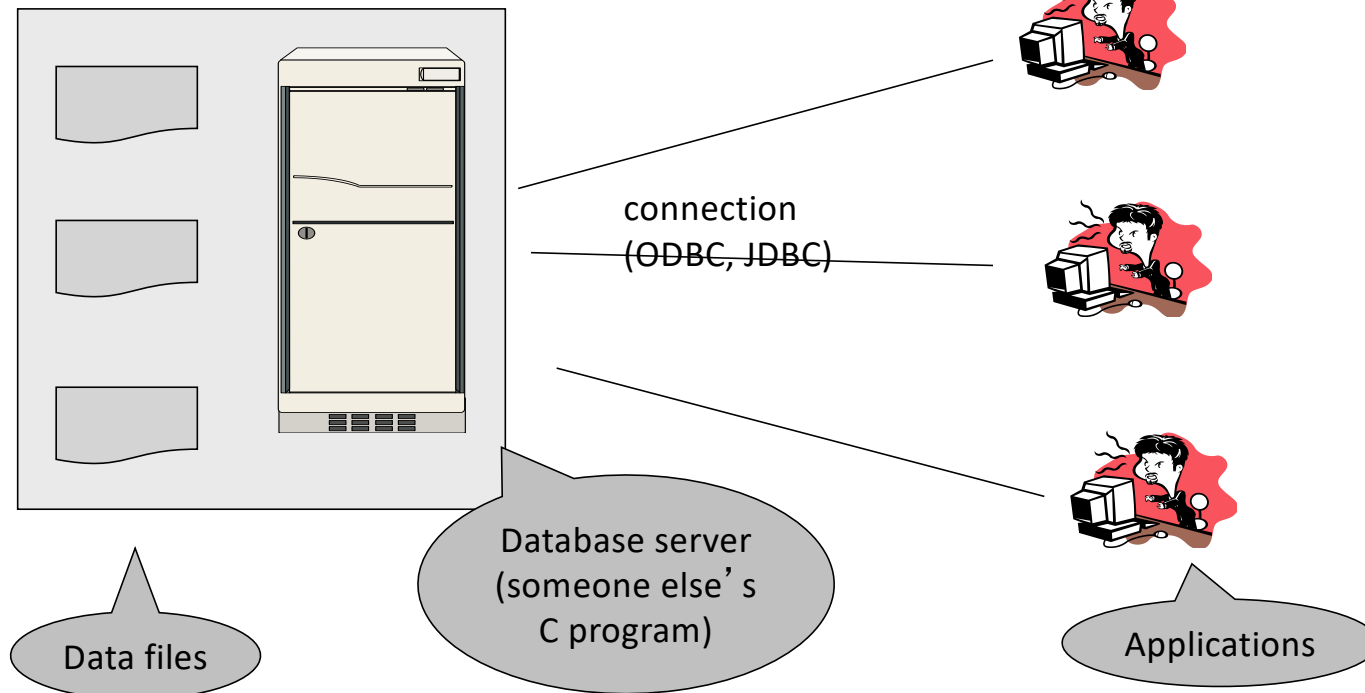
- What is the problem ?

- Simultaneous access by many users

- Need locks: we know them from OS, but now data on disk;
and is there any fun to re-implement them ?

Enters a DMBS

“Two tier database system”



Functionality of a DBMS

The programmer sees SQL, which has two components:

- Data Definition Language - DDL
- Data Manipulation Language - DML
 - query language

Behind the scenes the DBMS has:

- Query optimizer
- Query engine
- Storage management
- Transaction Management (concurrency, recovery)

Functionality of a DBMS

Two things to remember:

- Client-server architecture
 - Slow, cumbersome connection
 - But good for the data
- It is just someone else's C program
 - In the beginning we may be impressed by its speed
 - But later we discover that it can be frustratingly slow
 - We can do any particular task faster outside the DBMS
 - But the DBMS is *general* and *convenient*

How the Programmer Sees the DBMS

- Start with DDL to *create tables*:

```
CREATE TABLE Students (  
    Name CHAR(30)  
    SSN CHAR(9) PRIMARY KEY NOT NULL,  
    Category CHAR(20)  
) ...
```

- Continue with DML to *populate tables*:

```
INSERT INTO Students  
VALUES( 'Charles' , '123456789' , 'undergraduate' )  
. . . .
```

How the Programmer Sees the DBMS

- Tables:

Students:

SSN	Name	Category
123-45-6789	Charles	undergrad
234-56-7890	Dan	grad

Takes:

SSN	CID
123-45-6789	CSE444
123-45-6789	CSE444
234-56-7890	CSE142
	...

Courses:

CID	Name	Quarter
CSE444	Databases	fall
CSE541	Operating systems	winter

- Still implemented as files, but behind the scenes can be quite complex

“data independence” = separate logical view from physical implementation

Transactions

- Enroll “Mary Johnson” in “CSE444”:

```
BEGIN TRANSACTION;  
  
INSERT INTO Takes  
  SELECT Students.SSN, Courses.CID  
  FROM Students, Courses  
  WHERE Students.name = 'Mary Johnson' and  
         Courses.name = 'CSE444'  
  
-- More updates here....  
  
IF everything-went-OK  
  THEN COMMIT;  
ELSE ROLLBACK
```

If system crashes, the transaction is still either committed or aborted

Transactions

- A *transaction* = sequence of statements that either all succeed, or all fail
- Transactions have the ACID properties:
 - A = atomicity
 - C = consistency
 - I = independence/Isolation
 - D = durability

Queries

- Find all courses that “Mary” takes

```
SELECT C.name  
FROM   Students S, Takes T, Courses C  
WHERE  S.name="Mary" and  
        S.ssn = T.ssn and T.cid = C.cid
```

- What happens behind the scene ?
 - Query processor figures out how to answer the query efficiently.

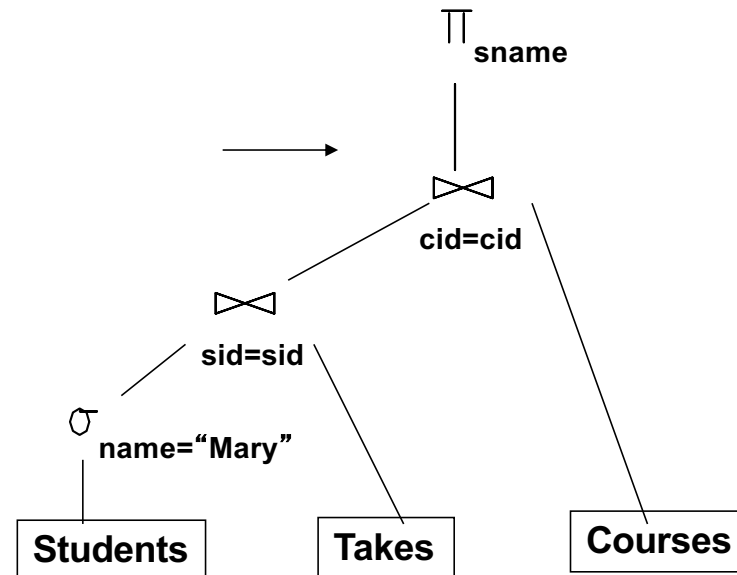
Queries, behind the scene

Declarative SQL query



Imperative query execution plan:

```
SELECT C.name  
FROM Students S, Takes T, Courses C  
WHERE S.name="Mary" and  
       S.ssn = T.ssn and T.cid = C.cid
```



The **optimizer** chooses the best execution plan for a query

Query Optimization

- Knows about
 - Data layout
 - Data sizes
 - Data Indexes
 - Storage properties

Database Systems

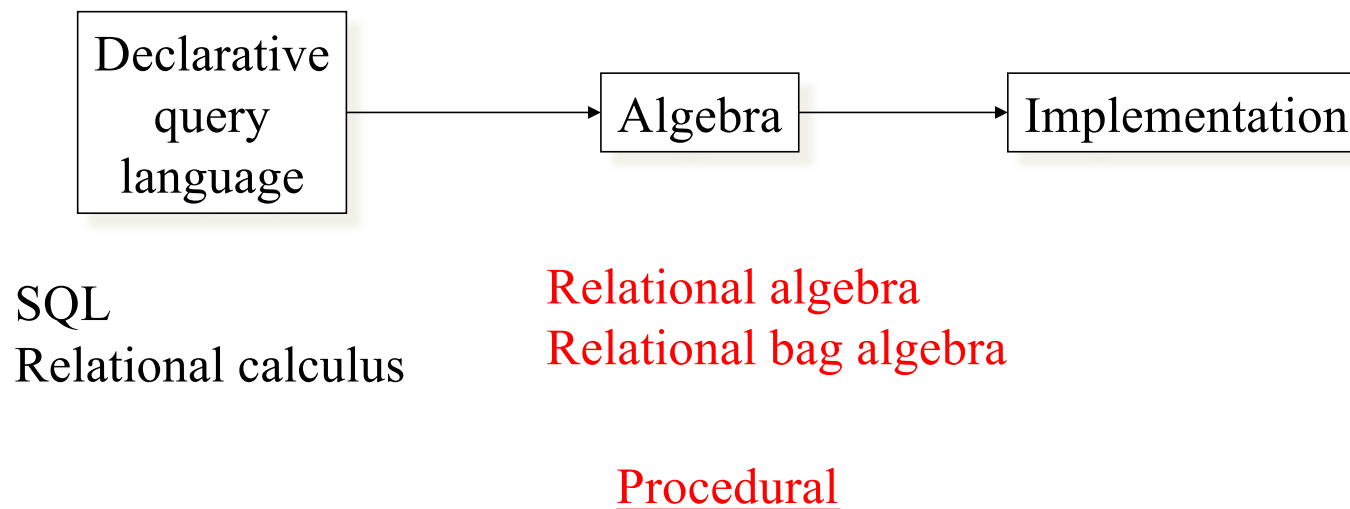
- The big commercial database vendors:
 - Oracle
 - IBM (with DB2) bought Informix recently
 - Microsoft (SQL Server)
 - Sybase
- Some free database systems (Unix) :
 - Postgres
 - Mysql

What is a Data Model?

1. Mathematical representation of data.
 - Examples: relational model = tables; semistructured model = trees/graphs.
2. Operations on data.
3. Constraints.

Relational Algebra

- Formalism for creating new relations from existing ones
- Its place in the big picture:



What is an “Algebra”

- Mathematical system consisting of:
 - *Operands* --- variables or values from which new values can be constructed.
 - *Operators* --- symbols denoting procedures that construct new values from given values.

Some basic concepts...

- Tuple
 - An ordered list of elements: (1), ("a", "b", "c"),...
- Set
 - A collection of distinct objects: {1, 2, 5, 7}
 - Operations on set such as union, intersection, ...
- Bag
 - A collect of non-distinct objects (i.e. an object may appear more then once: { 1, 2, 2, 2, 5, 5, 5, 7, 7} \rightarrow {(1,1), (3,2), (3, 4), (2, 7)})
- Relation
 - A named set of tuples

Why Relations?

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, the most important database language today.

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
 - The result is an algebra that can be used as a *query language* for relations.

Why Bother?

- Different implementations may yield different results
- What a clear definition of what the result of a query is over a data set
- Mathematical model defines “semantics”
 - Declarative semantics vs. implementation defined semantics

Relational Algebra at a Glance

- Operators: relations as input, new relation as output
- Five basic RA operations:
 - Basic Set Operations
 - union, difference
 - Selection: σ
 - Projection: π
 - Cartesian Product: \times (sometimes denoted as $*$)
- When our relations have conflicting attribute names:
 - Renaming: ρ
- Derived operations:
 - Intersection
 - Joins (theta join, equi-join, natural, semi-join, etc.)

Five Basic RA Operations

Our Running Example

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Relational Schemas

- *Relation schema* = relation name and attribute list.
 - Optionally: types of attributes.
 - Example: `Beers(name, manf)` or `Beers(name: string, manf: string)`

Set Operations

- Union, difference
- Both are binary operations

Set Operations: Union

- Union: all tuples in R1 **or** R2
- Notation: $R1 \cup R2$
- R1, R2 must have the same schema
- $R1 \cup R2$ has the same schema as R1, R2
- Example:
 - ActiveEmployees \cup RetiredEmployees

Set Operations: Difference

- Difference: all tuples in R1 and **not** in R2
- Notation: $R1 - R2$
- R1, R2 must have the same schema
- $R1 - R2$ has the same schema as R1, R2
- Example
 - AllEmployees - RetiredEmployees

Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- **Unary operation**
- c is a condition: boolean expression built from $=$, $<$, $>$, and, or, not
- Output schema: same as input schema
- Find all employees with salary more than \$40,000:
 - $\sigma_{Salary > 40000}(\text{Employee})$

Example: Selection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\sigma_{\text{bar}=\text{"Joe's"}}(\text{Sells})$:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Projection

- $R1 := \pi_L(R2)$
 - L is a list of attributes from the schema of $R2$.
 - $R1$ is constructed by looking at each tuple of $R2$, extracting the attributes on list L , in the order specified, and creating from those components a tuple for $R1$.
 - Eliminate duplicate tuples, if any.

Example: Projection

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\pi_{\text{beer}, \text{price}}$ (Sells):

Duplicate removed!

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Extended Projection

- Using the same Π_L operator, we allow the list L to contain arbitrary expressions involving attributes:
 1. Arithmetic on attributes, e.g., $A+B \rightarrow C$.
 2. Duplicate occurrences of the same attribute.

Example: Extended Projection

R =

(A	B)
1	2
3	4

$\pi_{A+B \rightarrow C, A, A} (R) =$

C	A1	A2
3	1	1
7	3	3

Product

- $R3 := R1 \times R2$
 - Pair each tuple $t1$ of $R1$ with each tuple $t2$ of $R2$.
 - Concatenation $t1t2$ is a tuple of $R3$.
 - Schema of $R3$ is the attributes of $R1$ and then $R2$, in order.
 - But beware attribute A of the same name in $R1$ and $R2$: use $R1.A$ and $R2.A$.

Example: $R3 := R1 \times R2$

R1(

A,	B)
1	2
3	4

R2(

B,	C)
5	6
7	8
9	10

R3(

A,	R1.B,	R2.B,	C)
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

Renaming

- The ρ operator gives a new schema to a relation.
- $R1 := \rho_{R1(A1, \dots, An)}(R2)$ makes R1 be a relation with attributes $A1, \dots, An$ and the same tuples as R2.
- Simplified notation: $R1(A1, \dots, An) := R2$.

Example: Renaming

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

$R(\text{bar}, \text{addr}) := \text{Bars}$

R(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

Derived RA Operations

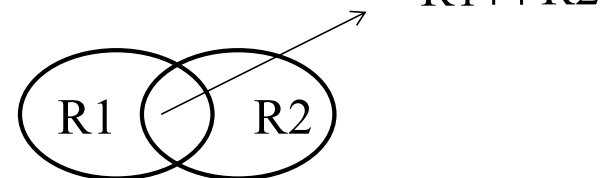
1) Intersection

2) Most importantly: Join

Set Operations: Intersection

- Intersection: all tuples both in R1 and in R2
- Notation: $R1 \cap R2$
- R1, R2 must have the same schema
- $R1 \cap R2$ has the same schema as R1, R2
- Example
 - UnionizedEmployees \cap RetiredEmployees
- Intersection is derived:

– $R1 \cap R2 = R1 - (R1 - R2)$ **why ?**



Joins

- Theta join
- Natural join
- Equi-join
- Semi-join
- Inner join
- Outer join
- etc.

Theta-Join

- $R3 := R1 \bowtie_C R2$
 - Take the product $R1 \times R2$.
 - Then apply σ_C to the result.
- As for σ , C can be any boolean-valued condition.
 - Historic versions of this operator allowed only $A \theta B$, where θ is $=$, $<$, etc.; hence the name “theta-join.”

Example: Theta Join

Sells(

bar,	beer,	price)
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

Bars(

name,	addr)
Joe's	Maple St.
Sue's	River Rd.

BarInfo := Sells  Bars
Sells.bar = Bars.name

BarInfo(

bar,	beer,	price,	name,	addr)
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

Natural Join

- A useful join variant (*natural* join) connects two relations by:
 - Equating attributes of the same name, and
 - Projecting out one copy of each pair of equated attributes.
- Denoted $R3 := R1 \bowtie R2$.

Example: Natural Join

Sells(

bar,	beer,	price)
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

Bars(

bar,	addr)
Joe's	Maple St.
Sue's	River Rd.

BarInfo := Sells  Bars

Note: Bars.name has become Bars.bar to make the natural join “work.”

BarInfo(bar,beer, price, addr)

Joe's	Bud	2.50	Maple St.
Joe's	Miller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

Equi-join

- Most frequently used in practice:



$R1 \quad A=B \quad R2$

- Natural join is a particular case of equi-join
- A lot of research on how to do it efficiently

Relational Bag Algebra

Relational Algebra on Bags

- A *bag* is like a set, but an element may appear more than once.
 - *Multiset* is another name for “bag.”
- Example: $\{1,2,1,3\}$ is a bag. $\{1,2,3\}$ is also a bag that happens to be a set.
- Bags also resemble lists, but order in a bag is unimportant.
 - Example: $\{1,2,1\} = \{1,1,2\}$ as bags, but $[1,2,1] \neq [1,1,2]$ as lists.

Why Bags?

- SQL, the most important query language for relational databases is actually a bag language.
 - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like projection, are much more efficient on bags than sets.

Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, but **as a bag operator, we do not eliminate duplicates.**
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

R(A,	B)	S(B,	C)
	1	2			3	4	
	5	6			7	8	
	1	2					

SELECT _{A+B<5} (R) =	A	B
	1	2
	1	2

Example: Bag Projection

R(A,	B)	S(B,	C)
	1	2			3	4	
	5	6			7	8	
	1	2					

PROJECT _A (R) =	A
	1
	5
	1

Example: Bag Product

R(A,	B)	S(B,	C)
1		2		3		4	
5		6		7		8	
1		2					

R * S =	A	R.B	S.B	C
	1	2	3	4
	1	2	7	8
	5	6	3	4
	5	6	7	8
	1	2	3	4
	1	2	7	8

Example: Bag Theta-Join

R(A,	B)	S(B,	C)
1		2		3		4	
5		6		7		8	
1		2					

R JOIN _{R.B < S.B} S =	A	R.B	S.B	C
	1	2	3	4
	1	2	7	8
	5	6	7	8
	1	2	3	4
	1	2	7	8

Bag Union

- Union, intersection, and difference need new definitions for bags.
- An element appears in the union of two bags the **sum** of the number of times it appears in each bag.
- Example: $\{1,2,1\} \text{ UNION } \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

Bag Intersection

- An element appears in the intersection of two bags the **minimum** of the number of times it appears in either.
- Example: $\{1,2,1\} \text{ INTER } \{1,2,3\} = \{1,2\}$.

Bag Difference

- An element appears in the difference $A - B$ of bags as many times as it appears in A , minus the number of times it appears in B .
 - But never less than 0 time.
- Example: $\{1,2,1\} - \{1,2,2,3\} = \{1\}$.

Beware: Bag Laws \neq Set Laws

- Not all algebraic laws that hold for sets also hold for bags.
- For one example, the commutative law for union $(R \text{ UNION } S = S \text{ UNION } R)$ *does* hold for bags.
 - Since addition is commutative, adding the number of times x appears in R and S doesn't depend on the order of R and S .

An Example of Inequivalence

- Set union is *idempotent*, meaning that:
 - $S \text{ UNION } S = S$.
- However, for bags, if x appears n times in S , then it appears $2n$ times in $S \text{ UNION } S$.
- Thus $S \text{ UNION } S \neq S$ in general.

Finally: RA has Limitations!

- Cannot compute “transitive closure”

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred
- Cannot express in RA, need recursion !!!
- But note that new SQL standard permits recursion.