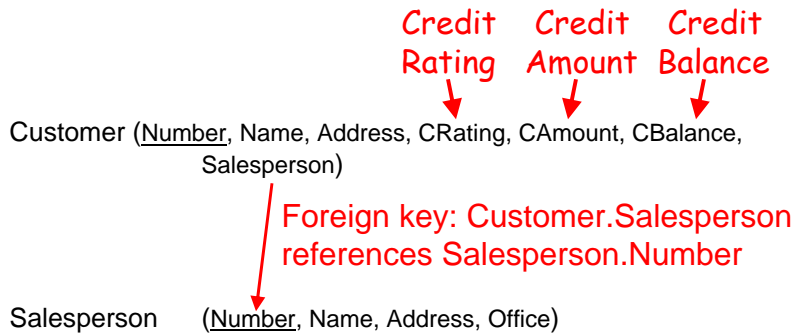


## Sample Database

We use the following database for some SQL queries:



## Extensions to the SELECT clause

Consider the following two queries:

```
SELECT DISTINCT Name
FROM Customer;
```

```
SELECT Name
FROM Customer;
```

The first query eliminates duplicate rows from the answer.

The second query does not eliminate duplicate rows.

The query writer gets to choose whether duplicates are eliminated!

## Challenge Question

Under what conditions are the following two queries equivalent?

```
SELECT DISTINCT A
FROM Table1;
```

```
SELECT A
FROM Table1;
```

## More Extensions to the SELECT clause

These queries are equivalent except for the column names in the query answer. "Important-Customer" used in the 2nd query.

```
SELECT Name, Address
FROM Customer
WHERE CRating = 10;
```

```
SELECT Name AS Important-Customer, Address
FROM Customer
WHERE CRating = 10;
```

The query writer can assign new names to the columns in a query answer with the AS <new column name> clause.

## More Extensions to the SELECT Clause

We can use aggregate operators in the SELECT clause: **COUNT**, **SUM**, **MIN**, **MAX**, and **AVG**

```
SELECT MIN(CBalance), MAX(CBalance), AVG (CBalance)
FROM Customer;
```

```
SELECT MIN(CBalance), MAX(CBalance), AVG (CBalance)
FROM Customer
WHERE age > 35;
```

If **one aggregate operator appears** in the SELECT clause, **then ALL** of the entries in the select clause **must be an aggregate operator** (unless the query includes a GROUP BY clause (covered later)).

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006  
Some slides adapted from R. Ramakrishnan, with permission

Slide 5

## Does this make sense?

```
SELECT S.Name, S.Phone, AVG(S.Age)
FROM Student S
WHERE S.Major = "CS";
```

Consider the data:

Student	ID	Name	Phone	Age	Major
	1	Joe	123	24	CS
	2	Mary	456	28	CS
	3	Arun	789	32	CS
	4	John	999	18	English

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006  
Some slides adapted from R. Ramakrishnan, with permission

Slide 6

## More Extensions to the SELECT Clause

What is the difference between these two queries?

```
SELECT COUNT(Name)      SELECT COUNT(DISTINCT Name)
FROM Customer;          FROM Customer;
```

## Questions

What is the implication of using **DISTINCT** when computing the **SUM** or **AVG** of an attribute?

What is the implication of using **DISTINCT** when computing the **MIN** or **MAX** of an attribute?

## Questions

What is the implication of using DISTINCT when computing the SUM or AVG of an attribute?

**Answer:** The SUM or AVG will be computed only on distinct values (compared to the normal case - without DISTINCT - where the SUM and AVG are computed on all values)

What is the implication of using DISTINCT when computing the MIN or MAX of an attribute?

**Answer:** There is no difference. The maximum or minimum value is the same whether or not duplicates are eliminated.

## More Extensions to the SELECT clause

The SELECT clause list can also include simple arithmetic expressions using +, -, \*, and /.

```
SELECT (CAmount - CBalance) AS AvailableCredit, Name  
FROM Customer  
WHERE CAmount > 0;
```

This query computes the available credit, for those Customers who have credit.

## Describing a Relational Database Mathematically

- A relation is a **set** – of **tuples** defined on **attributes**
- As in the original definition of the relational model [Codd 1970]
- Query operators are adapted from **set theory**
- **Relational algebra** is a mathematically defined query language

## Relational Algebra as a Query Language for Relational Databases

Note: we don't normally use relational algebra directly....there aren't any products (in wide use) that support relational algebra as a user language.

But...it's often used internally in a DBMS.

## Select operator ( $\sigma$ ) in relational algebra

Consider the following relation ... (with its extent)

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Consider the query:  $\sigma_{\text{Balance} < 3000}$  Account

## Select operator example

$\sigma_{\text{Balance} < 3000}$  Account

The select predicate is evaluated for each tuple

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
<del>103</del>	<del>J. Smith</del>	<del>5000.00</del>	<del>savings</del>
104	M. Jones	1000.00	checking
<del>105</del>	<del>H. Martin</del>	<del>10,000.00</del>	<del>checking</del>

## Select operator in relational algebra

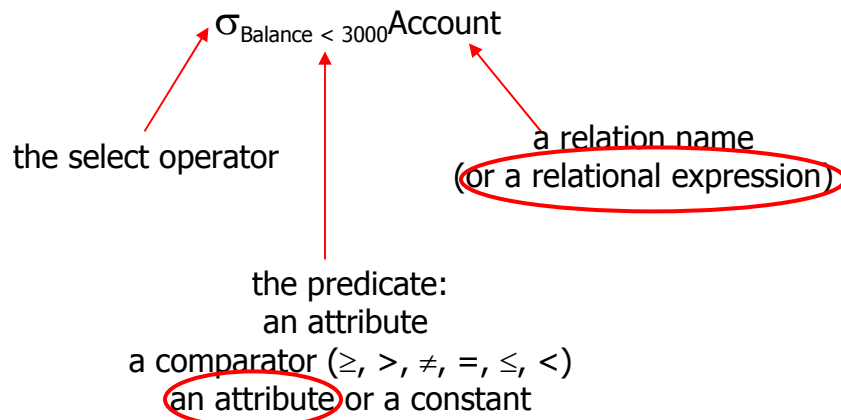
For this query:  $\sigma_{\text{Balance} < 3000}$ Account

The query answer is:

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
104	M. Jones	1000.00	checking

## Select operator in relational algebra

Always applied to a single relation – a unary operator



## Examples using the select operator

$\sigma_{\text{Balance} < 3000}$ Account

$\sigma_{\text{Number} = 103}$ Account

$\sigma_{\text{Owner} = \text{Type}}$ Account *Attribute compared to attribute*

$\sigma_{\text{Type} = \text{"checking"}}$ ( $\sigma_{\text{Balance} < 3000}$ Account) *relational expression*

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006 Slide 17  
Some slides adapted from R. Ramakrishnan, with permission

## Project operator ( $\pi$ ) in relational algebra

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Consider the query:  $\pi_{\text{Number, Owner}}$ Account

list of attributes (to retain) 

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006 Slide 18  
Some slides adapted from R. Ramakrishnan, with permission

## Project operator ( $\pi$ ) in relational algebra

Always applied to single relation – a unary operator

Consider the query:  $\pi_{\text{Number, Owner}} \text{Account}$

query answer is:

Number	Owner
101	J. Smith
102	W. Wei
103	J. Smith
104	M. Jones
105	H. Martin

## Project operator ( $\pi$ ) – another example

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

Consider the query:  $\pi_{\text{Owner}} \text{Account}$

list of attributes (to retain) 

## Example (cont.)

Consider the query:  $\pi_{\text{Owner}} \text{Account}$

Query answer is:

Owner
W. Wei
J. Smith
M. Jones
H. Martin

In relational algebra (as defined mathematically), relations are always sets. Thus the query answer is a set. J. Smith appears just once in the query answer.

## Select and Project can be combined

Account

Number	Owner	Balance	Type
101	J. Smith	1000.00	checking
102	W. Wei	2000.00	checking
103	J. Smith	5000.00	savings
104	M. Jones	1000.00	checking
105	H. Martin	10,000.00	checking

$$\pi_{\text{Owner}}(\sigma_{\text{Balance} < 3000} \text{Account})$$

$$\sigma_{\text{Balance} < 3000}(\pi_{\text{Owner, Balance}} \text{Account})$$

$$\pi_{\text{Owner}}(\sigma_{\text{Balance} < 3000}(\pi_{\text{Owner, Balance}} \text{Account}))$$

Which of these queries are equivalent?

## Cross Product

an operator from set theory  
(that has been modified slightly, by Codd)

Suppose..  $A = \{a, b, c\}$   $B = \{1, 2\}$

then in *set theory*, the cross product is defined as:

$$A \times B = \{(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2)\}$$

$A \times B$  is a set consisting of pairs (2-tuples) where each pair consists of an element from A and an element from B

## Practice Question

Suppose..  $A = \{a, b, c\}$   $B = \{1, 2\}$

What is  $B \times B$ ?

## Challenge Question

Suppose..  $A = \{a, b, c\}$   $B = \{1, 2\}$

is  $A \times B$  equivalent to  $B \times A$ ?

## Cross product in Set Theory

Suppose..  $A = \{a, b, c\}$   $B = \{1, 2\}$   $C = \{x, y\}$  then in *set theory*, the cross product is defined as:

$A \times B = \{(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2)\}$

and  $(A \times B) \times C =$

$\{((a,1),x), ((b,1),x), ((c,1),x), ((a,2),x), ((b,2),x),$   
 $((c,2),x), ((a,1),y), ((b,1),y), ((c,1),y), ((a,2),y),$   
 $((b,2),y), ((c,2),y)\}$

## Relational Algebra vs. Set Theory (cross product) (cont.)

Given  $A = \{a, b, c\}$   $B = \{1, 2\}$   $C = \{x, y\}$  with the cross product  $(A \times B) \times C$   
in *set theory* =

$\{((a,1),x), ((b,1),x), ((c,1),x), ((a,2),x), ((b,2),x), ((c,2),x),$   
 $((a,1),y), ((b,1),y), ((c,1),y), ((a,2),y), ((b,2),y), ((c,2),y))\}$

we simplify it in *relational algebra* to:

$\{(a,1,x), (b,1,x), (c,1,x), (a,2,x), (b,2,x), (c,2,x),$   
 $(a,1,y), (b,1,y), (c,1,y), (a,2,y), (b,2,y), (c,2,y)\}$

by eliminating parentheses...“flattening” the tuples.

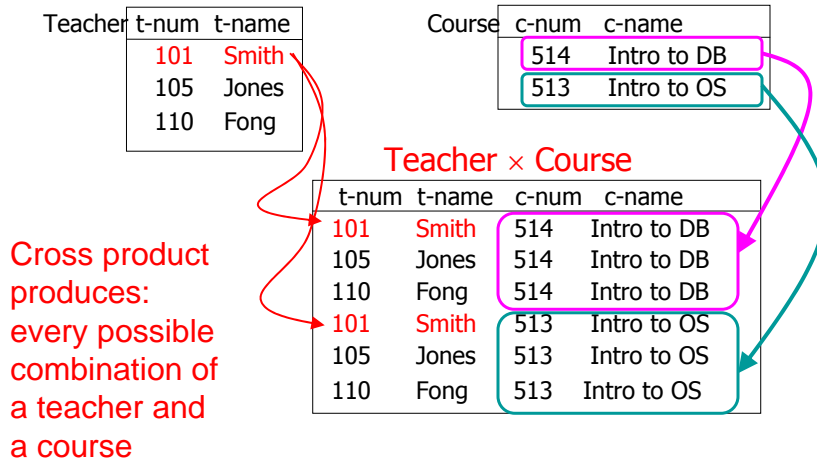
## New Example Database – To Show how the Cross Product Operator works

Imagine that we have these two relations in a university database. We'll use these simple relations in the next example.

Teacher (t-num, t-name)

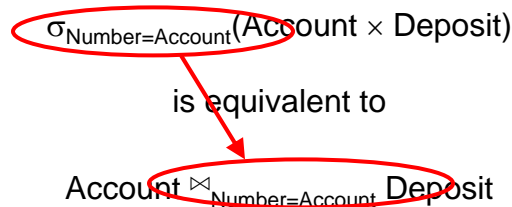
Course (c-num, c-name)

## × cross product operator



## ⋈ join operator

Account	Number	Owner	Balance	Type
Deposit	Account	Transaction-id	Date	Amount
Check	Account	Check-number	Date	Amount



## Join operator – for convenience

- **Note: Join** ( $\text{Relation1} \bowtie_{A1=A2} \text{Relation2}$ , where A1 is an attribute of Relation1 and A2 is an attribute of Relation2)

is defined as  $\sigma_{A1=A2}(\text{Relation1} \times \text{Relation2})$

- Thus...we don't really "need" the join operator. Any query with a join that we want to express, we can always use  $\times$  and  $\sigma$ . But ... join is used very frequently so it has been defined as an operator, for our convenience.

## Join .. With all six comparators

- Deposit  $\bowtie_{\text{Account}=\text{Account}}$  Check
- Deposit  $\bowtie_{\text{Date} < \text{Date}}$  Check
- Deposit  $\bowtie_{\text{Amount} \geq \text{Amount}}$  Check etc.
- Join is sometimes called "theta-join" or " $\theta$ -join" where the  $\theta$  represents any of the 6 comparators
- The most common join (with equality) is called equi-join

## Definition of a Relation

- The instance of a relation is always a *set* of tuples. This means there are no duplicates.

- Suppose we have a relation defined as:

Person(name, salary, num, status) with domains defined as:

**Codd's definition of a table**

- name-values = {all possible strings of 30 characters}
- sal-values = {real numbers between 0 and 100,000}
- status-values = {"f", "p"}
- num-values = {integers between 0 and 9999}

any instance of the relation is always a subset ( $\subseteq$ ) of:

Name-values  $\times$  Sal-values  $\times$  Num-values  $\times$  Status-values

## Question

- How could you convert the basic SQL statement (SELECT...FROM...WHERE...) statement to relational algebra (assuming that we wanted to eliminate all duplicates in the answer)?

```
SELECT    DISTINCT x, y, z
FROM      Table1, Table2
WHERE     condition
```

## SPJ

- SELECT DISTINCT x, y, z  
FROM Table1, Table2  
WHERE condition

Is the same as:

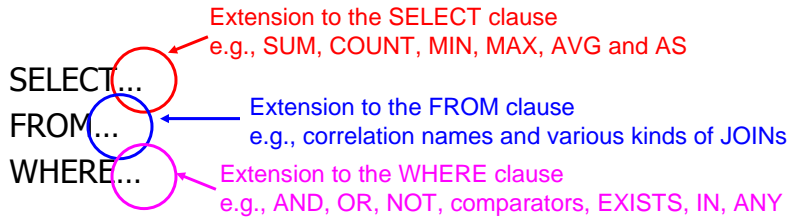
$\pi_{x, y, z} (\sigma_{\text{condition}} (\text{Table1} \times \text{Table2}))$

The basic SELECT ... FROM ... WHERE ..  
is sometimes described as equivalent to the SPJ subset of  
relational algebra (S = select, P = project, J = join)

## Back to SQL

remember: each product may differ  
in which features of SQL it supports

## SQL ... Additional Features



(SELECT...FROM...WHERE...)  
**UNION**  
(SELECT...FROM...WHERE...)

Operators that expect two or more complete SQL queries as operands  
e.g., UNION and INTERSECT

**ORDER BY...**

**GROUP BY...**

**HAVING ...**

Several additional clauses  
e.g., ORDER BY, GROUP BY, and HAVING

## Recall

Customer (Number, Name, Address, CRating, CAmount, CBalance,  
Salesperson)



Salesperson (Number, Name, Address, Office)

## SQL ... Extensions

SELECT...  
FROM... ← Extension to the FROM clause  
WHERE... e.g., correlation names and various kinds of JOINS

## Extensions to the FROM clause - Joins

There are a number of join types that can be expressed in the WHERE clause:

- inner join (the regular join)
- cross join
- natural join
- left outer join
- right outer join
- full outer join
- union join

## Extensions to the FROM clause - Joins

There are a number of join types that can be expressed in the WHERE clause:

- inner join (the regular join) ←
  - cross join ←
  - natural join ←
  - left outer join
  - right outer join
  - full outer join
- ... these joins can be expressed in a basic SELECT..FROM..WHERE query.

## Extensions to the FROM clause - Joins

There are a number of join types that can be expressed in the WHERE clause:

- inner join (the regular join)
  - cross join
  - natural join
  - left outer join ←
  - right outer join ←
  - full outer join ←
- There are new operators ... these joins can only be expressed in a complex SQL query involving the union operator.

We'll discuss them a little later.

## join clause in FROM (with ON)

Join condition in the FROM clause (vs. the WHERE clause)

These two queries are equivalent:

```
SELECT C.Name, S.Name
FROM Customer C JOIN Salesperson S ON C.Salesperson = S.Number
WHERE C.CRating < 6;
```

```
SELECT C.Name, S.Name
FROM Customer C, Salesperson S
WHERE C.Salesperson = S.Number AND C.CRating < 6;
```

## Join queries: SQL and equivalent relational algebra

```
SELECT C.Name, S.Name
FROM Customer C JOIN Salesperson S ON C.Salesperson = S.Number
WHERE C.CRating < 6;
```

$$\pi_{\text{Cust.Name, Salesp.Name}}(\sigma_{\text{C.CRating} < 6}(\text{Customer} \bowtie_{\text{Salesperson=Number}} \text{Salesperson}))$$

```
SELECT C.Name, S.Name
FROM Customer C, Salesperson S
WHERE C.Salesperson = S.Number AND C.CRating < 6;
```

$$\pi_{\text{Cust.Name, Salesp.Name}}(\sigma_{\text{C.CRating} < 6 \text{ AND Salesperson=Number}}(\text{Customer} \times \text{Salesperson}))$$

## JOIN with USING Clause

(when join attributes in the 2 tables have the same name)

Course (CNumber, CName, Description)  
 Teacher (TNumber, TName, Phone)  
 Offering (CNumber, TNumber, Time, Days, Room)

These two queries are equivalent:

```
SELECT Course.CNumber, Course.CName, Room
FROM Course JOIN Offering USING (CNumber);
```

```
SELECT C.Number, C.Name, O.Room
FROM Course C JOIN Offering O
ON C.CNumber = O.CNumber;
```

Note: USING clause doesn't need (and can't have) a correlation name

## Basic Join $\equiv$ INNER JOIN

For the INNER JOIN,

```
SELECT C.Name, S.Name
FROM Customer C INNER JOIN Salesperson S ON
C.Salesperson = S.Number;
```

the query answer includes all "matches"  
 but does not include:

- a Customer who doesn't have a Salesperson ....
- nor
- a Salesperson who is not assigned to any customers.

## Basic Join $\equiv$ INNER JOIN – the Default

This INNER JOIN query:

```
SELECT  C.Name, S.Name
FROM    Customer C INNER JOIN Salesperson S ON
        C.Salesperson = S.Number;
```

is the same as:

```
SELECT  C.Name, S.Name
FROM    Customer C JOIN Salesperson S ON
        C.Salesperson = S.Number;
```

## CROSS JOIN

A “CROSS JOIN” is a cross product

The following queries are equivalent:

```
SELECT  *
FROM    Customer, Salesperson;
```

```
SELECT  *
FROM    Customer CROSS JOIN Salesperson;
```

## NATURAL JOIN

NATURAL JOIN ... like a "macro" that joins tables with an equality check for all attributes **with the same name**.

Consider the following database

Course (CNumber, CName, Description)

Teacher (TNumber, TName, Phone)

Offering (CNumber, TNumber, Time, Days, Room)

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006 Slide 49  
Some slides adapted from R. Ramakrishnan, with permission

## Equi-join vs. Natural Join

equi-join: Account  $\bowtie$  Number=Account Deposit

When the join is based on equality, then we always have two identical columns of values in the answer.

Number	Owner	Balance	Type	Account	Trans-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
105	H. Martin	10,000.00	checking	105	4	11/02/00	10000.00

The **natural join** eliminates the duplicate column for joins based on equality. (This terminology is slightly different from the book.) Note: natural join requires attributes with the same name in the two relations.

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006 Slide 50  
Some slides adapted from R. Ramakrishnan, with permission

## Natural Join Drops Columns (automatically)

With any join based on equality, **there will always be pairs of identical columns** (one for each column that is joined).

The NATURAL JOIN **eliminates one** of the duplicate columns.

## Natural Join

List course name and teacher name for all course offerings. This query can be expressed with the NATURAL JOIN or with an INNER JOIN.

These two queries are equivalent.

```
SELECT CName, TName
FROM   Course NATURAL JOIN Offering
       NATURAL JOIN Teacher;
```

```
SELECT CName, TName
FROM   Course C, Offering O, Teacher T
WHERE  C.CNumber = O.CNumber AND
       O.TNumber = T.TNumber;
```

## Natural Join

Will the columns with the same name appear in the query answer twice? Or once?

Products may differ ... regarding when they drop duplicate columns ...

```
SELECT *
FROM   Course NATURAL JOIN Offering
       NATURAL JOIN Teacher;
```

```
SELECT *
FROM   Course C, Offering O, Teacher T
WHERE  C.CNumber = O.CNumber AND
       O.TNumber = T.TNumber;
```

## Relational Algebra Operators

There are eight operators

- $\pi$  project ←
  - $\sigma$  select ←
  - $\cup$  union
  - $\cap$  intersection
  - $-$  difference
  - $\times$  cross product ←
  - $\bowtie$  join ←
  - $\div$  division
  - renaming (to provide names for the relation & attributes of answer)
- Four operators  
that you've already  
seen

## Relational Algebra Operators

There are eight operators

- $\pi$  project ←
- $\sigma$  select ←
- $\cup$  union ← Three operators from set theory
- $\cap$  intersection ←
- $-$  difference ←
- $\times$  cross product ←
- $\bowtie$  join ←
- $\div$  division
- renaming (to provide names for the relation & attributes of answer)

## Intersection, Union, Difference

- $\cup$  UNION
- $\cap$  INTERSECTION
- $-$  SET DIFFERENCE

These operators can only be used with relations that are "union-compatible".

## Union Compatible

- Two relations are *union-compatible* if they have the same degree (i.e., the same number of attributes) and the corresponding attributes are defined on the same domains.
- Suppose we have these relations:

Checking-Account (c-num, c-owner, c-balance)

Savings-Account (s-num, s-owner, s-balance)

These are *union-compatible* relations.

- Union, intersection, & difference all require union-compatible relations

## Relational Algebra Operators

$\cup$  union Checking-account  $\cup$  Savings-account

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00
	103	J. Smith	5000.00

## Relational Algebra Operators

$\cap$  intersection

Checking-account  $\cap$  Savings-account

What's the answer to this query?

$(\pi_{c-owner} \text{Checking-account}) \cap (\pi_{s-owner} \text{Savings-account})$

What's the answer to this new query?

## Relational Algebra Operators

Checking-account  $\cap$  Savings-account

What's the answer to this query?

It's empty. There are no tuples that are in both relations.

$(\pi_{c-owner} \text{Checking-account}) \cap (\pi_{s-owner} \text{Savings-account})$

What's the answer to this new query?

	c-owner
	J. Smith

## Relational Algebra Operators

— difference

Checking-account — Savings-account

Find all the tuples (rows) that are in the Checking-account relation that are not in the Savings-account relation.

$(\pi_{c-owner} \text{Checking-account}) - (\pi_{s-owner} \text{Savings-account})$

## SQL – for Union, Intersection, Difference

SELECT...  
FROM...  
WHERE...

(SELECT...FROM...WHERE...)  
**UNION**  
(SELECT...FROM...WHERE...)

Operators that expect two or more complete SQL queries as operands  
e.g., UNION and INTERSECT

## SQL UNION

```
(SELECT C.Name
FROM Customer C
WHERE C.Name LIKE "B%")
```

UNION

```
(SELECT S.Name
FROM Salesperson S
WHERE S.Name LIKE "B%");
```

Two complete queries - with the UNION operator in between.

## SQL INTERSECTION

```
(SELECT C.Name
FROM Customer C)
```

INTERSECT

```
(SELECT S.Name
FROM Salesperson S);
```

Two complete queries - with the INTERSECT operator in between.

## SQL EXCEPT (or MINUS) (set difference)

<pre>(SELECT S.Number FROM Salesperson S)</pre>	<p>Two complete queries - with the EXCEPT operator in between.</p>
<pre>EXCEPT (SELECT C.Salesperson FROM Customer C);</pre>	

## Sets vs. Bags (also called Multi-sets)

Consider  $S1=\{a,b\}$ ,  $S2=\{a,b,c,d\}$ ,  $S3=\{d,b\}$

- What is  $S1 \cup S2$ ?
- What is  $S3 - S1$ ?
- What is  $S2 \cap S3$ ?

Now consider bags  $B1=\{a,a,b,b,b\}$ ,  $B2=\{a,b,b,c,c,d,d\}$ ,  
 $B3=\{b,b,b,b,d,d,d\}$

- What is  $B1 \cup B2$ ?
- What is  $B3 - B1$ ?
- What is  $B2 \cap B3$ ?

Is there any correspondence between set and bag results?

## Bags of Tuples

fund1(name amount)	fund2(name amount)
Ng 5	Ng 4
Ng 5	Ng 5
Apt 3	Apt 3
Apt 5	Apt 3
Apt 6	Apt 3
Fry 7	Fry 2

- take the union of these bags (UNION ALL)
- take the intersection (INTERSECT ALL)
- take the set difference (EXCEPT ALL)

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006  
Some slides adapted from R. Ramakrishnan, with permission

Slide 67

## Despite these differences...

- Relational operators (with appropriate additional operators beyond the original eight) are used to represent queries internally – in a DBMS
- A complete set of operators for bags – that includes the relational algebra operators – plus the additional operators – can be defined formally...and the equivalences can be proven

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006  
Some slides adapted from R. Ramakrishnan, with permission

Slide 68

## Why do we use Relational Algebra?

Because:

- It is mathematically defined (where relations are sets)
- We can prove that two relational algebra expressions are equivalent. For example:

$$\begin{aligned} \sigma_{\text{cond1}} (\sigma_{\text{cond2}} R) &\equiv \sigma_{\text{cond2}} (\sigma_{\text{cond1}} R) \equiv \sigma_{\text{cond1 AND cond2}} R \\ &\equiv (\sigma_{\text{cond1}} R) \cap (\sigma_{\text{cond2}} R) \end{aligned}$$

$$R1 \bowtie_{\text{cond}} R2 \equiv \sigma_{\text{cond}} (R1 \bowtie R2)$$

## "AND", "OR", and "NOT"

$$\sigma_{\text{cond1 OR cond2}} R \equiv (\sigma_{\text{cond1}} R) \cup (\sigma_{\text{cond2}} R)$$

$$\sigma_{\text{cond1 AND cond2}} R \equiv (\sigma_{\text{cond1}} R) \cap (\sigma_{\text{cond2}} R)$$

$$\sigma_{\text{cond1 AND NOT cond2}} R \equiv (\sigma_{\text{cond1}} R) - (\sigma_{\text{cond2}} R)$$

The WHERE clause (and the predicate for the  $\sigma$  operator) may contain AND, OR, as well as NOT.

## Uses of Relational Algebra Equivalences

- To help query writers – they can write queries in several different ways
- To help query optimizers – they can choose among different ways to execute the query

and in both cases **we know for sure** that the two queries (the original and the replacement) are identical – that they will produce the same answer

## INNER JOIN vs OUTER JOIN

For the INNER JOIN,

```
SELECT  C.Name, S.Name
FROM    Customer C INNER JOIN Salesperson S ON
        C.Salesperson = S.Number;
```

the query answer does not include:

- a Customer who doesn't have a Salesperson ....
- nor
- a Salesperson who is not assigned to any customers.

## INNER JOIN vs OUTER JOIN

an INNER (regular) JOIN includes only that customers that have salespersons (only the matches)

```
SELECT    C.Name, S.Name
FROM      Customer C INNER JOIN Salesperson S ON
          C.Salesperson = S.Number;
```

- a LEFT OUTER JOIN will include all matches plus all Customers who don't have a Salesperson ....
- a RIGHT OUTER JOIN will include all matches plus all Salespersons who are not assigned to any customers
- a FULL OUTER JOIN will include all of these!

## LEFT OUTER JOIN

Assume for now that foreign key is not enforced

Customer

Number	Name	Address	CRating	CAmount	CBalance	Salesperson
1	smith	xxx	5	1,000	1,000	101
2	jones	yyy	7	5,000	4,000	101
3	wei	zzz	10	10,000	10,000	103

Salesperson

Number	Name	Address	Office
101	johnson	aaa	23
102	miller	bbb	26

**INNER JOIN** on C.Salesperson = S.Number gives us:  
 "smith" with "johnson" and "jones" with "johnson"

**LEFT OUTER JOIN** on C.Salesperson = S.Number gives us:  
 INNER JOIN plus "wei" with "<null>" salesperson

## RIGHT OUTER JOIN

Customer

Number	Name	Address	CRating	CAmount	CBalance	Salesperson
1	smith	xxx	5	1,000	1,000	101
2	jones	yyy	7	5,000	4,000	101
3	wei	zzz	10	10,000	10,000	103

Salesperson

Number	Name	Address	Office
101	johnson	aaa	23
102	miller	bbb	26

**INNER JOIN** on C.Salesperson = S.Number gives:

“smith” with “johnson” and “jones” with “johnson”

**RIGHT OUTER JOIN** on C.Salesperson = S.Number gives:

INNER JOIN plus “<null>” with “miller” salesperson

## FULL OUTER JOIN

Customer

Number	Name	Address	CRating	CAmount	CBalance	Salesperson
1	smith	xxx	5	1,000	1,000	101
2	jones	yyy	7	5,000	4,000	101
3	wei	zzz	10	10,000	10,000	103

Salesperson

Number	Name	Address	Office
101	johnson	aaa	23
102	miller	bbb	26

**INNER JOIN** on C.Salesperson = S.Number gives us:

“smith” with “johnson” and “jones” with “johnson”

**FULL OUTER JOIN** on C.Salesperson = S.Number gives us:

INNER JOIN union LEFT OUTER union RIGHT OUTER

## SQL ... Additional Clauses

SELECT...  
FROM...  
WHERE...

ORDER BY...            Several additional clauses  
GROUP BY...            e.g., ORDER BY, GROUP BY,  
HAVING ...              and HAVING

Note that “ORDER BY” simply sorts the query answer. It is considered an “implementation specific” feature of SQL

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006            Slide 77  
Some slides adapted from R. Ramakrishnan, with permission

Store	Beverage	Size	Price
Plaid Pantry	Whole Milk	16oz	1.15
Plaid Pantry	2% Milk	16oz	1.05
Plaid Pantry	Whole Milk	64oz	3.20
Plaid Pantry	Pepsi	12oz	.70
Plaid Pantry	Diet Pepsi	12oz	.70
Plaid Pantry	Pepsi	20 oz	.95
7-11	Whole Milk	8oz	.65
7-11	Chocolate Milk	8oz	.65
7-11	Whole Milk	16oz	1.10
7-11	2% Milk	16oz	1.00
7-11	Coke	12oz	.65
7-11	Diet Coke	12oz	.65
7-11	Coke	20oz	1.10
7-11	Diet Coke	20oz	1.10
7-11	Diet Caffeine-Free Cherry Low-Fizz Vitamin-Fortified Coke	20oz	1.10
Circle K	Whole Milk	8oz	.60
Circle K	Whole Milk	16oz	1.20
Circle K	Whole Milk	32oz	2.30
Circle K	Whole Milk	128oz	4.10
Circle K	Coke	20oz	1.15
Circle K	Diet Coke	20oz	1.15
Circle K	Coke	32oz	2.10
Circle K	Diet Coke	32oz	2.05
Safeway	Skim Milk	16oz	1.20
Safeway	Skim Milk	32oz	2.00
Safeway	2% Milk	16oz	1.25
Safeway	2% Milk	64oz	2.75
Safeway	Diet Pepsi	12oz	.60
Safeway	Diet Coke	12oz	.60

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006            Slide 78  
Some slides adapted from R. Ramakrishnan, with permission

## GROUP BY, HAVING

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson;
```

```
SELECT Salesperson
FROM Customer
GROUP BY Salesperson
HAVING Count(*) > 1;
```

Any form of SQL query (e.g., with or without subqueries) can have the answer "grouped". When a query answer is grouped, **there is one output row - for each group.**

## GROUP BY, HAVING

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson;
```

```
SELECT Salesperson
FROM Customer
GROUP BY Salesperson
HAVING Count(*) > 1;
```

The **HAVING** clause is a **predicate evaluated against each group**. A group participates in the query answer if it satisfies the **HAVING** predicate

## Group by example

Input data:

Customer

(Number, Name, Address, ... Salesperson)

101	Mary	...	5
102	John	...	8
103	Quan	...	NULL
106	Susan	...	5
107	David	...	5
109	Mike	...	2
110	Ying	...	8

```
SELECT Salesperson
FROM Customer
GROUP BY Salesperson;
```

## Example: Group by

Input data:

Customer

(Number, Name, Address, ... Salesperson)

101	Mary	...	5
102	John	...	8
103	Quan...		NULL
106	Susan	...	5
107	David	...	5
109	Mike	...	2
110	Ying	...	8

Intermediate result:  
4 groups

103	Quan...		Null
101	Mary ...		5
106	Susan	...	5
107	David	...	5
102	John	...	8
110	Ying	...	8
109	Mike	...	2

### Example: Group by

Intermediate result:  
4 groups

Query answer:  
one row per group!

103	Quan...	Null	
101	Mary ...		5
106	Susan	...	5
107	David	...	5
102	John	...	8
110	Ying	...	8
109	Mike	...	2

Salesperson
Null
5
8
2

```
SELECT Salesperson
FROM Customer
GROUP BY Salesperson;
```

### Example: Group by with Having

Intermediate result:  
4 groups

Query answer:  
one row per group  
that satisfies the  
HAVING clause!

<del>103</del>	<del>Quan...</del>	<del>Null</del>	
101	Mary ...		5
106	Susan	...	5
107	David	...	5
102	John	...	8
110	Ying	...	8
<del>109</del>	<del>Mike</del>	<del>...</del>	<del>2</del>

Salesperson
5
8

```
SELECT Salesperson
FROM Customer
GROUP BY Salesperson
HAVING Count(*) > 1;
```

## GROUP BY, HAVING

```
SELECT Salesperson, COUNT(*)
FROM Customer
GROUP BY Salesperson;
```

```
SELECT Salesperson
FROM Customer
GROUP BY Salesperson
HAVING Count(*) > 1;
```

The only attributes that can appear in a “grouped” query answer are **the grouping attribute** or **aggregate operators** (that are applied to the group).

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006  
Some slides adapted from R. Ramakrishnan, with permission

Slide 85

## Group By - Example

The following query is **not legal**

```
SELECT Name
FROM Customer
GROUP BY Salesperson;
```

...because there can be many names for each group!

---

CS386 Introduction to Database Systems, © Lois Delcambre, David Maier 1999-2006  
Some slides adapted from R. Ramakrishnan, with permission

Slide 86

## Group By – Order of Clauses

- The WHERE clause is evaluated *before* the GROUP BY
- The HAVING clause is evaluated *after* the GROUP BY

## Question

```
SELECT    Salesperson, AVG(CBalance)
FROM      Customer
GROUP BY  Salesperson
HAVING    AVG(CBalance) > 200;
```

```
SELECT    Salesperson, AVG(CBalance)
FROM      Customer
WHERE     CBalance > 200
GROUP BY  Salesperson;
```

Are these queries equivalent (give the same answer)?