# Relational Model and Relational Algebra

# The Relational Model

- The **relational model** describes the *logical view* of the data by using tables
  - The logical view does not describe how the data is stored
  - There are different ways of storing relations on disks
  - Efficiency is the major consideration when determining how to store relations on disks

# The Relational Algebra

- **Relational algebra (RA)** is a (mathematical) *query language* for the relational model
- A query language is used for writing questions about the data
- Typically, a query language is high level, namely, it describes what we want but not how to compute it efficiently

# SQL

- **SQL** is the concrete query language used in relational database management systems (RDBMS)
- Some of the differences between RA and SQL are subtle
  - The result of a relational-algebra expression is always a set, whereas the result of an SQL query could have duplicates

# Why do we need to understand RA?

- Real queries are written in **SQL**, but are translated by the query processor into relational algebra
- Why?
  - SQL is declarative, RA provides (high level) operations for execution
  - Optimization is easier in RA, since we can take advantage of (provable) expression equivalences

# What you should know

1. How to calculate the result of a relational-algebra expression over a set of relations
2. How to write queries in relational algebra
3. How to determine whether two relational-algebra expressions are equivalent
4. How to push selection and projection
5. How to prove that the basic rel.-algebra operators are independent of one another

# The Relational Data Model

# The Relational Model

- Intuitively, a relation is a table
- Formally, a relation has two parts: schema and instance
- **Schema: Name($att_1$:$dom_1$, …, $att_k$:$dom_k$)**
  - Example: Students(sid: number, sname: string, year: int)
  - In these slides, we usually omit the domain (i.e., type)
- **Instance:** A **set** of tuples (rows) with arity and types that match the schema
  - Can be empty!
  - No duplicates!

# מונחים בעברית

- record, row, tuple
- ח-יה, שורה, רשומה
- All of the above are different names for the same thing

# Example Table

**Schema: Students(sid, sname, year)**

**sname is a column name or an attribute**

**Students**

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

**a row or tuple**

# Some Notes

- The order of the rows is not important

| sname | sid | year |
|---|---|---|
| smith | 241 | 4 |
| white | 240 | 3 |

=

| sname | sid | year |
|---|---|---|
| white | 240 | 3 |
| smith | 241 | 4 |

- The order of the columns is important when
  - Some columns do not have a name, or
  - Two or more columns have the same name
- **No null values**

# Relational Algebra

# Relational Algebra

- Relational algebra is a collection of operators on ***relations***
- Operators may be ***unary*** or ***binary***
- The output of an operator is a relation
  - That is, the algebra is “closed”
  - Therefore, operators can be composed one on another
- Note: the name of the output relation and some of its attribute names may be undefined

# Basic Operators

- Relational Algebra has 5 basic operators:
  - Projection (הטלה)
  - Selection (בחירה)
  - Union (איחוד)
  - Set difference (הפרש)
  - Cartesian product (מכפלה קרטזית)

**These 5 operators are Independent! (How can this be proven?)**

- Other operators can be defined using these: intersection (חיתוך), join (צירוף), division (חילוק)
- A useful syntactic operator: renaming (שינוי שם)

# An Example of Relations

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

T

| tid | tname | dept |
|---|---|---|
| 10 | cohen | math |
| 20 | levy | cs |

S = Students
T = Teachers
R = Studies

# הטלה

Projection

# Projection

- Projection is unary (i.e., it is applied to a single relation)
- Denoted by $\pi_{A_1,\ldots,A_n}$
  - $A_1,\ldots,A_n$ are attributes
- Projection returns a new relation that contains only the columns $A_1,\ldots,A_n$ from the original relation
- The output relation does not have a name

# Projection: Example

- The projection (in this case) returns the pairs of teacher id and course, such that the teacher teaches the course

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

$\pi_{\text{tid,course}}$ **R**

| tid | course |
|---|---|
| 20 | os |
| 10 | calculus |
| 20 | db |

Conceptually, projection is applied to each tuple individually

**Fewer tuples in result. Why?**

# Think about it

- You can mention the same column more than once in the projection, e.g., $\pi_{sid,\ sid}\, R$
- When computing a projection on a relation with *n* tuples,
  - What is the minimum number of tuples in the result?
  - What is the maximum number of tuples in the result?

# בחירה

Selection

# Selection

- Unary operator
- Written $\sigma_C$
  - C is a Boolean condition over a single tuple
- Returns the tuples that satisfy C
- Just like projection, you can think of selection as operating on each tuple individually

# Selection: Example

- Return the tuples for teacher number 20

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

$\sigma_{\mathbf{tid=20}}$ **R**

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 20 | 202 | db |
| 20 | 240 | db |

# Combining Selection and Projection

- What does this compute?

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

$\pi_{\text{course}}(\sigma_{\text{tid=20}}\ \text{R})$

**Can we change the order of these two operations?**

# Another Example

- How would you find the names of the third year students?

$$\pi_{\text{sname}}(\sigma_{\text{year=3}}\ R)$$

S

?

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

| sname |
|---|
| white |
| Jones |

# What types of Conditions can be used?

- The condition is made up of comparisons that are connected using logical operators (and, or)
- Comparisons are between two attributes or between an attribute and a constant
  - *attribute1* op *attribute2*
  - *attribute1* op *constant*
- ***Important! Conditions are evaluated a single tuple at a time. Cannot state global conditional on a table***

# Types of Comparisons

- We can use any of the comparisons:

$$<, \quad \leq, \quad >, \quad \geq, \quad =, \quad \neq$$

- When comparing an attribute with a string, the string is written in single quotes

$$\sigma_{tname='cohen'}(T)$$

# Example

- Find id's of students named jones who are in their first year or third year of studies

$$\pi_{sid}(\sigma_{sname='jones'\wedge(year=1\vee year=3)}(S))$$

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

| sid |
|---|
| 202 |

# Once Again

- What is in the result of the query when we have *this* instance of S?

$$\pi_{sid}(\sigma_{sname='jones'\wedge(year=1\vee year=3)}(S))$$

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | jones | 1 |

# A Variation

- What is in the result (and meaning) of *this* query?

$$\pi_{sid}(\sigma_{sname='jones'\wedge(year=1\wedge year=3)}(S))$$

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | jones | 1 |

**Can we express: Find the names for which there are some students with that name in year 1 and in year 3?**

# What does this return?

$$\pi_{sid}(\sigma_{course \neq 'db'}(R))$$

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

# Think about it

- When computing a selection on a relation with *n* tuples,
  - What is the minimum number of tuples in the result?
  - What is the maximum number of tuples in the result?
- Must duplicates be eliminated when processing a selection?

# איחוד

Union

# Union

- Binary operator
- Written R ∪ S
- Union can only be performed between ***compatible*** relations
  - same number of attributes
  - *corresponding* attributes have the same name and type
    - names (but not types) can be changed to meet this requirement
- Result contains all tuples that are in at least one of the relations
- Schema attributes of the result are those of R (which are the same as those of S)

# Example

$S$

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

$G$

| sid | sname | year |
|---|---|---|
| 701 | katz | 1 |
| 202 | jones | 3 |
| 820 | sapir | 2 |

$S \bigcup G$

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 701 | katz | 1 |
| 820 | sapir | 2 |
| 450 | adams | 1 |

# Think about it

- Suppose that R has n tuples, and S has m tuples
  - What is the minimum number of tuples in R ∪ S ?
  - What is the maximum number of tuples in R ∪ S ?
- Must duplicates be eliminated when processing a union?

# הפרש

Set Difference

# Set Difference

- Binary operator
- Can only be performed between relations that are compatible
- Written **R − S**
- Result contains the tuples from R, not in S
- Schema attributes of the result are those of R (which are the same as those of S)

# Example

$S$

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

$G$

| sid | sname | year |
|---|---|---|
| 701 | katz | 1 |
| 202 | jones | 3 |
| 820 | sapir | 2 |

$S - G$

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 450 | adams | 1 |

# Find ID's of Students Who DO NOT Study 'db'

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

# Questions

- Suppose that R has n tuples, and S has m tuples
  - What is the minimum number of tuples in R−S ?
  - What is the maximum number of tuples in R−S ?
- Must duplicates be eliminated when processing a set difference?

# מכפלה קרטזית

Cartesian Product

(Cross Product)

# Cartesian Product

- Binary Operator
- Written **R** $\times$ **S**
- Result of a Cartesian product of two relations is a new relation that contains a tuple for each pair of tuples from the two input relation
- Column names that appear in both R and S are qualified with the relation name, e.g., R.A and S.A
- Number of tuples in the result is always the product of the number of tuples in R and in S

# Recall: An Example of Relations

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

T

| tid | tname | dept |
|---|---|---|
| 10 | cohen | math |
| 20 | levy | cs |

S = Students
T = Teachers
R = Studies

$R \times T$

| R.tid (tid) | sid | course | T.tid (tid) | tname | dept |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| 10 | 450 | calculus | 10 | cohen | math |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| 20 | 202 | os | 20 | levy | cs |
| 10 | 450 | calculus | 20 | levy | cs |
| 20 | 202 | db | 20 | levy | cs |
| 20 | 240 | db | 20 | levy | cs |

# When to Use Cartesian Product

- In order to get a meaningful result, we usually write queries that have both a Cartesian product and a selection.
  - Example: Find the names of the courses taught by Levy
  - More about this soon, when we discuss joins
- What happens when a Cartesian product is applied and one of the relations is empty?

# שינוי שם

Renaming

# Renaming

- Attribute conflicts may sometimes occur in a relational-algebra expression (e.g., when using Cartesian product)
  - When else?
- Renaming can also give a name to (the result of) a sub-expression, which can be used to break down long expressions

# Renaming Syntax

- The expression $\rho_{R(A1,\ldots,An)}(E)$ takes the relational-algebra expression *E*, and returns a relation called *R*
- *R* contains the same tuples as *E* and the same number of attributes, except that they are renamed as $A_1,\ldots, A_n$
- Every column of E must be a given a name by $\rho$

# Recall: An Example of Relations

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

T

| tid | tname | dept |
|---|---|---|
| 10 | cohen | math |
| 20 | levy | cs |

S = Students
T = Teachers
R = Studies

$R \times T$

**Both columns have the same name!**
**Renaming is implicit (i.e., R.tid and T.tid)**

Does implicit renaming always solve the problem?

| (tid) | sid | course | (tid) | tname | dept |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| 10 | 450 | calculus | 10 | cohen | math |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| 20 | 202 | os | 20 | levy | cs |
| 10 | 450 | calculus | 20 | levy | cs |
| 20 | 202 | db | 20 | levy | cs |
| 20 | 240 | db | 20 | levy | cs |

$\rho_{C(tid1,sid,course,tid2,tname,dpmnt)}(R \times T)$

Renaming can be done explicitly by the operator $\rho$

| **tid1** | sid | course | **tid2** | tname | **dpmnt** |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| 10 | 450 | calculus | 10 | cohen | math |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| 20 | 202 | os | 20 | levy | cs |
| 10 | 450 | calculus | 20 | levy | cs |
| 20 | 202 | db | 20 | levy | cs |
| 20 | 240 | db | 20 | levy | cs |

$$\sigma_{tid1=tid2}(\rho_{C(tid1,sid,course,tid2,tname,dpmnt)}(R \times T))$$

| **tid1** | sid | course | **tid2** | tname | **dpmnt** |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| **10** | **450** | **calculus** | **10** | **cohen** | **math** |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| **20** | **202** | **os** | **20** | **levy** | **cs** |
| 10 | 450 | calculus | 20 | levy | cs |
| **20** | **202** | **db** | **20** | **levy** | **cs** |
| **20** | **240** | **db** | **20** | **levy** | **cs** |

The result consists of the purple tuples

# Find the names of the courses taught by Levy

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

T

| tid | tname | dept |
|---|---|---|
| 10 | cohen | math |
| 20 | levy | cs |

S = Students
T = Teachers
R = Studies

# Additional Operators

# Additional Operators

- We discussed the 5 basic operators of relational algebra
- Additional operators can be defined in terms of these
- Defining these operators is a good idea for two reasons:
  - Allows us to write simpler expressions
  - Allows some specific optimizations

# צירוף

Join

# Joins

- The result of a Cartesian product is usually not meaningful
- In order to derive an interesting result, selection is usually composed onto the Cartesian product (as in the previous example)
- The join is a “shortcut” for writing such expressions
- We will see three types of joins: conditional join, equijoin and natural join

# Conditional Join צירוף עם תנאי

- A conditional join has the format

$$R \bowtie_C S$$

  where *C* is a condition as in select, except that all the comparisons are between an attribute of R and an attribute of S

- This expression is equivalent to:

$$\sigma_C(R \times S)$$

- Conditional join is also called *theta-join*
- *Give examples that show legal and illegal C*

$$\rho_{R1(rtid,sid,course)}(R) \bowtie_{rtid<tid} T$$

| rtid | sid | course | tid | tname | dept |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| 10 | 450 | calculus | 10 | cohen | math |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| 20 | 202 | os | 20 | levy | cs |
| 10 | 450 | calculus | 20 | levy | cs |
| 20 | 202 | db | 20 | levy | cs |
| 20 | 240 | db | 20 | levy | cs |

**Which rows are in the result?**

# Alternatively, we can write $R \bowtie_{R.tid<T.tid} T$

| R.tid | sid | course | T.tid | tname | dept |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| 10 | 450 | calculus | 10 | cohen | math |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| 20 | 202 | os | 20 | levy | cs |
| 10 | 450 | calculus | 20 | levy | cs |
| 20 | 202 | db | 20 | levy | cs |
| 20 | 240 | db | 20 | levy | cs |

**Which rows are in the result?**

# צירוף שוויון Equijoin

- This is a special case of conditional join, where the condition is a conjunction of equalities between attributes
- For such cases, it is not necessary to have both equal columns in the result and the second one is dropped automatically
- Thus, translating an Equijoin to an expression using only the basic operators requires Cartesian product, selection and projection
- Give examples showing what is an equijoin and what is not

$$R \bowtie_{tid=tid2} \rho_{T1(tid2,tname,dept)}(T)$$

| tid | sid | course | tid2 | tname | dept |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| 10 | 450 | calculus | 10 | cohen | math |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| 20 | 202 | os | 20 | levy | cs |
| 10 | 450 | calculus | 20 | levy | cs |
| 20 | 202 | db | 20 | levy | cs |
| 20 | 240 | db | 20 | levy | cs |

**What is in the result?**

$$R \bowtie_{tid=tid2} \rho_{T1(tid2,tname,dept)}(T)$$

| tid | sid | course | tid2 | tname | dept |
|---|---|---|---|---|---|
| 20 | 202 | os | 10 | cohen | math |
| 10 | 450 | calculus | 10 | cohen | math |
| 20 | 202 | db | 10 | cohen | math |
| 20 | 240 | db | 10 | cohen | math |
| 20 | 202 | os | 20 | levy | cs |
| 10 | 450 | calculus | 20 | levy | cs |
| 20 | 202 | db | 20 | levy | cs |
| 20 | 240 | db | 20 | levy | cs |

**What is in the result?**

$$R \bowtie_{tid=tid2} \rho_{T1(tid2,tname,dept)}(T)$$

| tid | sid | course | tname | dept |
|---|---|---|---|---|
| 10 | 450 | calculus | cohen | math |
| 20 | 202 | os | levy | cs |
| 20 | 202 | db | levy | cs |
| 20 | 240 | db | levy | cs |

# צירוף טבעי Natural Join

- This is a special equijoin, where we require equality between every pair of attributes that have the same name in the two relations
- No condition is written

$$R \bowtie S$$

# צירוף טבעי Natural Join

- Example: $R \bowtie_{tid=tid2} \rho_{T1(tid2,tname,dept)}(T)$ is actually the same as $R \bowtie T$

- Question: What is in $R \bowtie T$ if R and T have no common attributes?
- Question: What is in $R \bowtie T$ if R and T have exactly the same attributes?

# Queries with Natural Join

# Names of Teachers Who Teach the Student with id 202

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

T

| tid | tname | dept |
|---|---|---|
| 10 | cohen | math |
| 20 | levy | cs |

$$\pi_{tname}\,(\sigma_{sid='202'}\,(R \bowtie T))$$

$$\pi_{tname}\,(\sigma_{sid='202'}\,(R) \bowtie T)$$

## What are the Years of Students Taught by Levy?

R

| tid | sid | course |
|---|---|---|
| 20 | 202 | os |
| 10 | 450 | calculus |
| 20 | 202 | db |
| 20 | 240 | db |

S

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

T

| tid | tname | dept |
|---|---|---|
| 10 | cohen | math |
| 20 | levy | cs |

Join is commutative and associative

$$\pi_{year}\,(\sigma_{tname='levy'}\,(S \bowtie R \bowtie T))$$

$$\pi_{year}\,(S \bowtie R \bowtie \sigma_{tname='levy'}(T))$$

# Example of Natural Join on More than One Column

# Equality Between Every Pair of Columns with the Same Attribute

S

| A | B | C |
|---|---|---|
| 202 | 20 | os |
| 450 | 10 | calc |
| 202 | 20 | db |
| 240 | 20 | db |

R

| A | C | D |
|---|---|---|
| 240 | calc | 3 |
| 202 | db | 2 |
| 450 | os | 1 |

$R \bowtie S$

| A | B | C | D |
|---|---|---|---|
| 202 | 20 | db | 2 |

# What is the Result Now?

S

| A | B | C |
|---|---|---|
| 202 | 20 | os |
| 450 | 10 | calc |
| 202 | 20 | db |
| 240 | 20 | db |

R

| A | C | D |
|---|---|---|
| 240 | os | 3 |
| 202 | db | 2 |
| 450 | calc | 1 |

$R \bowtie S$

| A | B | C | D |
|---|---|---|---|
| 450 | 10 | calc | 1 |
| 202 | 20 | db | 2 |

# חילוק

Division

# Division

- Division is useful for writing “for all” queries
  - But sometimes it is better not to use it for “for all”
- Examples:
  - Find students who studied **all** of Prof. Cohen’s courses
  - Find lecturers who have taught **all** the students
- We will use ÷ to denote division
  - Sometimes, */* is used for this operation

# Division

- Can perform R ÷ S if all the attributes appearing in S also appear in R
- The result is a relation with all the attributes appearing in R and not in S
- $R(X_1,\ldots,X_k,Y_1,\ldots,Y_m)$, $S(Y_1,\ldots,Y_m)$:
  - Which attributes are in the result of R ÷ S ?

# Division: Example

- Consider:
  - $A(X_1,\ldots,X_k,Y_1,\ldots,Y_m)$
  - $B(Y_1,\ldots,Y_m)$

- Then $A \div B =$

$$\{\langle x_1,\ldots,x_k\rangle \mid (\forall \langle y_1,\ldots,y_m\rangle \in B) \langle x_1,\ldots,x_k,y_1,\ldots,y_m\rangle \in A\}$$

# Suppliers from A who supply All Parts from B

| sno | pno |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S1 | P3 |
| S1 | P4 |
| S2 | P1 |
| S2 | P2 |
| S3 | P2 |
| S4 | P2 |
| S4 | P4 |

A

÷

| pno |
|---|
| P2 |

B

=

| sno |
|---|
| |

# Suppliers from A who supply All Parts from B

| sno | pno |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S1 | P3 |
| S1 | P4 |
| S2 | P1 |
| S2 | P2 |
| S3 | P2 |
| S4 | P2 |
| S4 | P4 |

A

÷

| pno |
|---|
| P2 |
| P4 |

B

=

| sno |
|---|
| |

# Suppliers from A who supply All Parts from B

| sno | pno |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S1 | P3 |
| S1 | P4 |
| S2 | P1 |
| S2 | P2 |
| S3 | P2 |
| S4 | P2 |
| S4 | P4 |

A

÷

| pno |
|---|
| P1 |
| P2 |
| P4 |

B

=

| sno |
|---|
| |

# Translating to Basic Operators

- To find the suppliers who supply all the parts

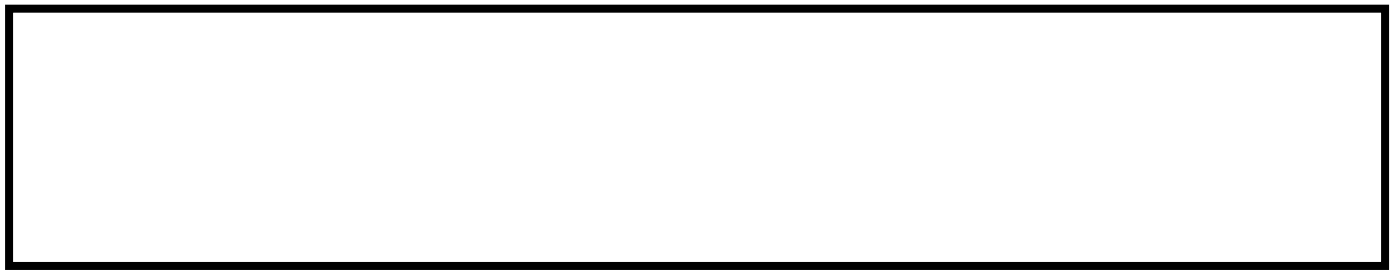
$$(A) \div (B)$$

- Division can be expressed using other relational algebra operators. How?

$$\pi_{sno} A - \pi_{sno}((\pi_{sno} A \times B) - A)$$

# Teachers who taught all the students

- To find the teachers who taught all the students:



# חיתוך

Intersection

# Intersection

- Binary operator
- Written **R ∩ S**
- Performed between compatible relations
- Result contains the tuples that appear in both R and S

# Example

$S \cap G$

**What is in the result?**

$S$

| sid | sname | year |
|---|---|---|
| 240 | white | 3 |
| 202 | jones | 3 |
| 450 | adams | 1 |

$G$

| sid | sname | year |
|---|---|---|
| 701 | katz | 1 |
| 202 | jones | 3 |
| 820 | sapir | 2 |

# Think about it

- How can we define ∩ using the other operators?
- Suppose that R has n tuples and S has m tuples
  - What is the minimum number of tuples in R ∩ S ?
  - What is the maximum number of tuples in R ∩ S ?
- Is duplicate elimination needed?

# Practicing Relational Algebra

- Suppliers(sid, sname, address)
- Parts(pid,pname,color)
- Catalog(sid,pid,cost)

The attributes of a key are underlined

1. Names of suppliers who supply some red parts
2. Sid's of suppliers who supply a red part and a green part
3. Sid's of suppliers who do not supply a red part
4. Sid's of suppliers who supply every red part

# Equivalences Among RA Expressions

# What Does Equivalence Mean?

- We say that expressions E1 and E2 are *equivalent* if for all instances (i.e., sets of tuples) of the relations mentioned in E1 and E2, these expressions always return the same result
- Simple example: Consider the relation R(A,B,C). The following three expressions are equivalent:

$$\pi_A R \qquad \pi_A \pi_{A,B} R \qquad \pi_A \pi_{A,C} R$$

- Can E1 and E2 be equivalent if they mention different relations?

# Why and How?

| | | |
|---|---|---|
| $E1 = \pi_A R$ | $E2 = \pi_A \pi_{A,B} R$ | $E3 = \pi_A \pi_{A,C} R$ |

- Why do we care that the expressions are equivalent?
  - Mostly, for optimizations. Which expression is the most efficient?
- How can we determine that the expressions are equivalent?
  - Prove containment in both directions
  - Other, advanced, methods also exist

# Examples and Notation

- Suppose that we have R(A,B) and S(A,B)
  - Is $\pi_A R \cup \pi_A S$ equivalent to $\pi_A (R \cup S)$ ?
  - Is $\pi_A R \cap \pi_A S$ equivalent to $\pi_A (R \cap S)$ ?
- Equivalence is denoted by ≡

$$\pi_A R \cup \pi_A S \equiv \pi_A (R \cup S)$$

  - We need to show that every tuple in the result of one side is also in the result of the other side (two directions to prove)

# Containment

- Sometimes there is only containment in one direction, denoted by $\subseteq$

$$\pi_A (R \cap S) \subseteq \pi_A R \cap \pi_A S$$

  - The proof is simple
- To show $\pi_A R \cap \pi_A S \not\subseteq \pi_A (R \cap S)$, we need to give a counterexample

# A Counterexample

| Boats1 | | |
|---|---|---|
| bid | bname | Color |
| 101 | Nancy | red |
| 105 | Matilda | Green |
| 103 | Gloria | red |

| Boats2 | | |
|---|---|---|
| bid | bname | Color |
| 107 | Kim | blue |
| 104 | Nancy | Green |
| 101 | Nancy | blue |

מה המשמעות והתוצאה של השאילתה הבאה?

$$\pi_{bname}(Boats1) \cap \pi_{bname}(Boats2)$$

ואם אנו מחליפים את סדר החיתוך וההטלה?

$$\pi_{bname}(Boats1 \cap Boats2)$$

# Equivalences for Optimizing Select-Project-Join Expressions

# How to Optimize?

- Make intermediate results as small as possible using the following principles
  - Apply selection and projection as soon as possible (see example)
  - Determine the best order of computing the joins (the join is commutative and associative – see next slide)

$$\pi_{year}\,(S \bowtie (R \bowtie \sigma_{tname='levy'}T))$$

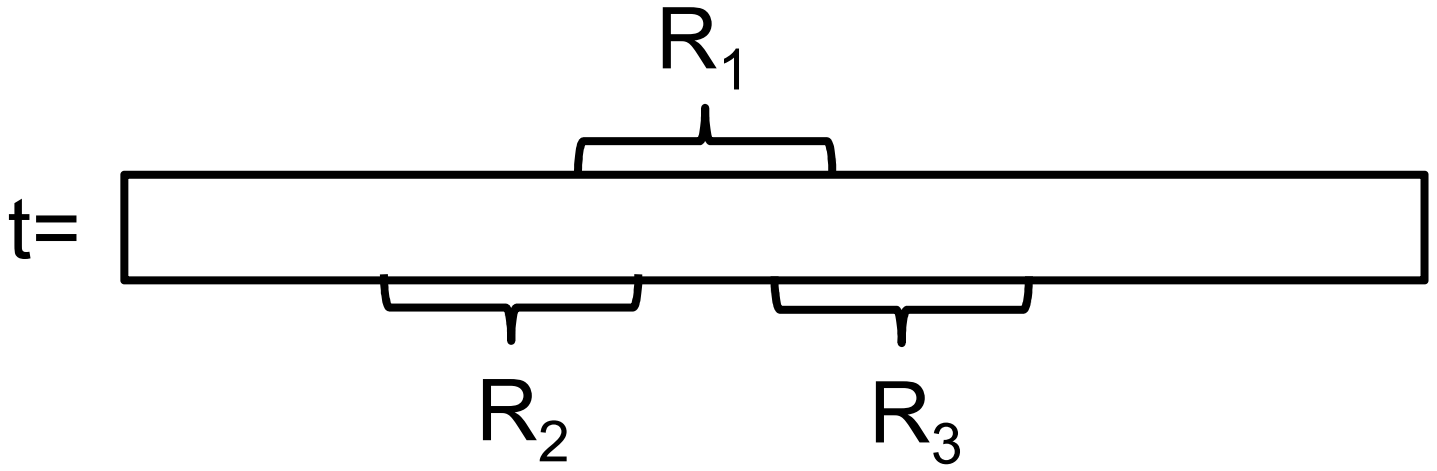
# Proof that the Natural Join is Associative and Commutative

- Implied by the following characterization
- A tuple t is in the result of $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$ if and only if
  - The attributes of t are those of $R_1,\ldots,R_n$, and
  - for all i, $t[R_i]$ is in the relation of $R_i$

Notation: We use $R_i$ to denote both the relation name and its set of attributes. We denote the projection of a tuple t on a set of attributes X as t[X].

# The Characterization of Join

- A tuple t of $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$ is over the attributes of $\bigcup_{i=1}^{n} R_i$




- $\forall i(\pi_{R_i}(t) \in R_i) \Leftrightarrow t \in \bowtie_{i=1}^{n} R_i$
- Note: $\pi_{R_i}(t)$ is also written as $t[R_i]$

# Explanation

- The characterization from the previous slide does not depend on either the order of the relations in the expression or the order of applying the join operators (i.e., the way of putting parentheses)
- Therefore, the join operator is commutative and associative

# Equivalence Rules for Projection

- $X_1$ is the subset of X that appears in R
- $X_2$ is the subset of X that appears in S
- Y is the set of all attributes common to R & S

$$\pi_X(R \bowtie S) \not\equiv \pi_{X_1}(R) \bowtie \pi_{X_2}(S)$$

Which $\subseteq$ ?

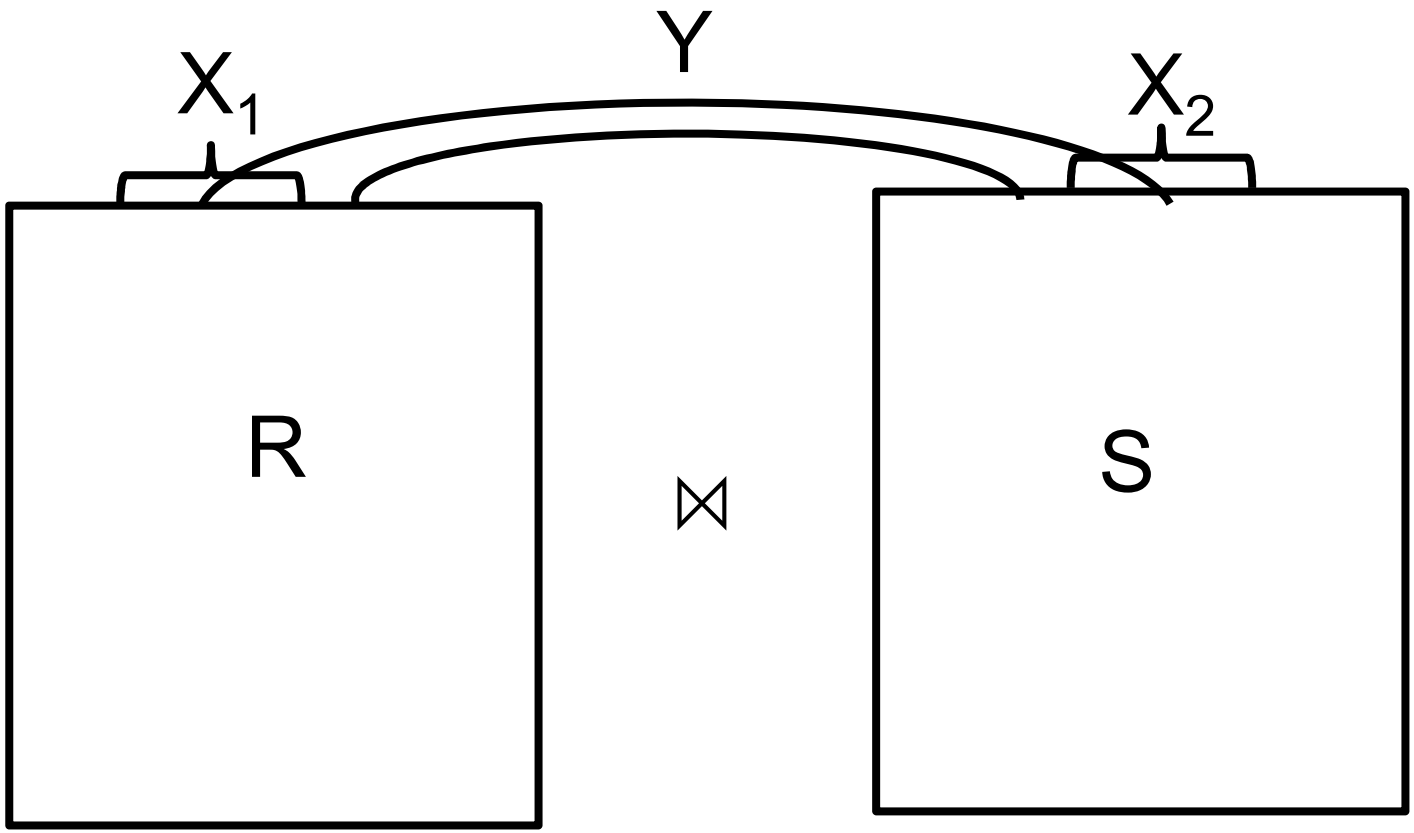
$$\pi_X(R \bowtie S) \not\equiv \pi_{X_1Y}(R) \bowtie \pi_{X_2Y}(S)$$

$$\pi_X(R \bowtie S) \equiv \pi_X(\pi_{X_1Y}(R) \bowtie \pi_{X_2Y}(S))$$

Notation: For sets of attributes X and Y, we use XY to denote the union of X and Y

$$\pi_X(R \bowtie S) \equiv \pi_X(\pi_{X_1Y}(R) \bowtie \pi_{X_2Y}(S))$$

$$X_1 = R \cap X,\ Y = R \cap S,\ X_2 = S \cap X$$



# Explanation

- We must keep all the attributes of Y until after the join is done
- We must also keep all the attributes of X, because they are needed for the final result
- Therefore, before the join we project R and S on $X_1Y$ and $X_2Y$, respectively, and after the join we project on X

# Equivalence Rules for Selection

- $\sigma_{C_1 \wedge C_2}(E) \equiv \sigma_{C_1}(\sigma_{C_2}(E)) \equiv \sigma_{C_2}(\sigma_{C_1}(E))$
- If all attributes of C are in R,

$$\sigma_C(R \bowtie S) \equiv \sigma_C(R) \bowtie S$$

- If all attributes of C are in S,

$$\sigma_C(R \bowtie S) \equiv R \bowtie \sigma_C(S)$$

- If all attributes of C are in both R and S,

$$\sigma_C(R \bowtie S) \equiv \sigma_C(R) \bowtie \sigma_C(S)$$

Is there a similar rule for logical OR?

If possible, better to push selection into the two operands rather than just one

# Equivalence Rule for Selection Followed by Projection

- Assuming that $\sigma_C\pi_X(E)$ is well defined (namely, the attributes of C are all in X), then

$$\sigma_C\pi_X(E) \equiv \pi_X\sigma_C(E)$$

We can always push selections through projections

# Pushing Selections and Projections

- Repeatedly split each selection with $\wedge$ using the equivalence $\sigma_{C_1 \wedge C_2}(E) \equiv \sigma_{C_1}(\sigma_{C_2}(E))$
- Repeatedly do the following:
  - Push selections through projections
  - Push selections into every operand of a natural join if possible (i.e., if the operand contains all the attributes of the selection)
- Push projections inside joins (using Slide 98):

$$\pi_X(R \bowtie S) \equiv \pi_X(\pi_{X_1 Y}(R) \bowtie \pi_{X_2 Y}(S))$$

# Example

- $\pi_A \sigma_{B<4} \sigma_{C>5} \pi_{A,B,C,D}(R(A,B,C) \bowtie S(C,D) \bowtie T(D,E))$
- Push selections

  $\pi_A \pi_{A,B,C,D}(\sigma_{B<4} \sigma_{C>5} R(A,B,C) \bowtie \sigma_{C>5} S(C,D) \bowtie T(D,E))$
- Collapse consecutive projections into one

  $\pi_A(\sigma_{B<4} \sigma_{C>5} R(A,B,C) \bowtie \sigma_{C>5} S(C,D) \bowtie T(D,E))$
- Choose which join will be done first (based on size estimation of intermediate results)

  $\pi_A((\sigma_{B<4} \sigma_{C>5} R(A,B,C) \bowtie \sigma_{C>5} S(C,D)) \bowtie T(D,E))$

# Example (continued)

- So far, we have

$$\pi_A((\sigma_{B<4}\sigma_{C>5}R(A,B,C) \bowtie \sigma_{C>5}S(C,D)) \bowtie T(D,E))$$

- Apply projections as early as possible

$$\pi_A(\pi_{A,D}(\pi_{A,C}\sigma_{B<4}\sigma_{C>5}R(A,B,C) \bowtie \sigma_{C>5}S(C,D)) \bowtie \pi_D T(D,E))$$

  - Can do it by applying the rule mentioned at the bottom of Slide 103, but there is an alternative intuitive rule:

After each selection and join, project only on the attributes that are needed later

# Comment

- A sequence of selections and projections is evaluated in one step
- Therefore, something like $\pi_A\sigma_{B<4}\sigma_{C>5}\pi_{A,B,C,D}$ can be written just as $\pi_A\sigma_{B<4}\sigma_{C>5}$
  - The whole sequence is evaluated efficiently in one scan
- Given a sequence of projections and selections, it is sufficient to write it as some selections followed by a single projection

# Another Comment

Determining join order and pushing selections & projections should be done together, in general

- If a selection is just a single comparison (not necessarily =) between an attribute and a constant, then
  - It can be pushed to the base relations prior to determining the join order
- In general, however, we need to know the join order before pushing selections
  - Same for projections

# Converting Equijoins to Natural Joins

- An expression with selections, projections and equijoins can always be converted into an equivalent expression with selections, projections and natural joins
  - First, rename attributes so that no attribute appears in more than one relation
  - Next, if two attributes are equated by an equijoin, then give them the same name

# Independence of RA Operators

# Independence

- The 5 basic operators:
  - Projection (הטלה)
  - Selection (בחירה)
  - Cartesian product (מכפלה קרטזית)
  - Union (איחוד)
  - Set difference (הפרש)

  are independent
- This means that there are queries that cannot be expressed if one of the operators is removed

# Independence of Cartesian Product and Projection

- We can show independence of an operator if we find a property P that it has whereas the other four operators do not satisfy P
- What is a property of projection that the other four operators do not have?
- What is a property of Cartesian product that the other four operators do not have?

# Proving Independence of Union

- There does not seem to be a global property of union that the other operators do not have
- To prove independence of union, we will consider a property with respect to some specific relations

# Independence of Union

- Consider the relation schemas R(A) and S(A) with the following instances:
  - R has only the tuple <0>
  - S has only the tuple <1>
- The result of R ∪ S has two tuples
- Every expression E over R and S that uses only selection, projection, Cartesian product and difference returns a relation with at most one tuple
- Proof: Induction on the number of operators in E

# Monotonic Queries

- A query is *monotonic* if adding tuples to the relations may only add tuples to the result
- Monotonicity is a *semantic* property, namely, it depends on the meaning of the query and not on how we write it in relational algebra
  - We can determine monotonicity just from the description of the query in a natural language

# Monotonic Queries (cont'd)

- Which operators are monotonic and which are not?
  - An operator is non-monotonic if adding tuples to some of its operands can remove tuples from the result
- What is the conclusion about independence?
- Which queries among those discussed in the exercise session are monotonic and which are not?
- Which operators are essential for expressing non-monotonic queries?

# Independence of Selection

- Consider the relation R(A) with the tuples <0> and <1>
- Consider the expression $\sigma_{A=0}(R)$
- Find some property P and prove that
  - $\sigma_{A=0}(R)$ does not have property P
  - every expression over R and the operators $\pi$, $\cup$, $\times$ and $-$ returns a relation that satisfies P