SE240: Introduction to Database Systems

Lecture 04:

Relational Algebra and Calculus

Outline

- **Relational Algebra**
  - **Unary Relational Operations**
  - Binary Relational Operations

Relational Query Languages

- Query Languages (QL): Allow manipulation and retrieval of data from a database
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for much optimization
- Query Languages != Programming Languages!
  - QLs not expected to be “Turing complete”
  - QLs not intended to be used for complex calculations
  - QLs support easy, efficient access to large data sets

Formal Relational Query Languages

- Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
  - *Relational Algebra*: More operational (procedural), very useful for representing execution plans
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it: Non-operational, declarative

Relational Algebra

- *Relational algebra* is the basic set of *operations* for the relational model
- These operations enable a user to specify basic retrieval requests (*queries in terms of operators*) 算子
- Every operator in relational algebra accepts (one or two) *relation* instances as arguments and returns a *relation* instance as the result (*makes the algebra "closed"*)
- Relational algebra is a procedural query language
  - Defines a step-by-step procedure for computing the answer

Relational Algebra

- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol: $\boldsymbol{\sigma}$ (sigma))
    - PROJECT (symbol: π (pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE (or MINUS, –)
    - CARTESIAN PRODUCT (**x**) $A \times B = \{(a,b) \mid a \in A \text{ and } b \in B\}$
  - Binary Relational Operations
    - JOIN (several variations of JOIN exist)
    - DIVISION
  - Additional Relational Operations
    - OUTER JOINS
    - AGGREGATE FUNCTIONS
- Each operation returns a relation, and operations can be composed!

SELECT (σ)

- The SELECT operation (denoted by **σ**(sigma)) is used to select a *subset* of the tuples from a relation based on a ***selection condition***

manually given

  - The selection condition acts as a ***filter***
  - Keeps only those tuples that satisfy the qualifying condition
- The SELECT operation is denoted by $\sigma_{\text{<selection condition>}}(R)$ where
  - The symbol **σ** (sigma) is used to denote the *select* operator
  - The *selection condition* is a Boolean (conditional) expression specified on the attributes of relation R
  - Tuples that make the condition $\boxed{\textbf{true}}$ are selected
  - Tuples that make the condition $\boxed{\textbf{false}}$ are filtered out

SELECT (σ)

- <selection condition> is a Boolean combination (an expression using logical connectives $\wedge$ and $\vee$) of terms:
  - Condition have the form: **Term** *op* **Term** operators
    - **Term** is an attribute name or a constant
    - *op* is one of $<, \leq, =, \neq, >, \geq$
  - Different conditions can be linked together with a boolean expression
    - **($C1 \wedge C2$)**, **($C1 \vee C2$)**, **($\neg C1$)** are conditions where C1 and C2 are conditions
      - $\wedge$ means AND
      - $\vee$ means OR
      - $\neg$ means NOT

Example: SELECT (σ)

Relation R

a	b	c	d
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{a=b \wedge d>5}(R)$

a	b	c	d
α	α	1	7
β	β	23	10

Do not forget to list the attribute name
at 1st row of the table

Example: SELECT (σ)

- Employee

department_number

f_name	l_name	id	sex	salary	superid	dno
Joseph	Chan	999999	M	2950	654321	4
Victor	Wong	001100	M	3000	888555	5
Carrie	Kwan	898989	F	2600	654321	4
Joyce	Fong	345345	F	1200	777888	4

- Find all employees who works in department 4 and whose salary is greater than 2500

$$\sigma_{dno=4 \wedge salary>2500}(Employee)$$

f_name	l_name	id	sex	salary	superid	dno
Joseph	Chan	999999	M	2950	654321	4
Carrie	Kwan	898989	F	2600	654321	4

SELECT (σ) Properties

- The SELECT operation $\sigma_{<\text{selection condition}>}(R)$ produces a relation S that has the same schema (same attributes) as R
- SELECT σ is commutative: $\sigma_{<c_1>}(\sigma_{<c_2>}(R)) = \sigma_{<c_1 \wedge c_2>}(R)$
  - $\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(R)) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition1}>}(R))$
- Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
  (con1, con2, con3) = (con1, con3, con2)
  = (con2, con1, con3) = (con2, con3, con1)
  = (con3, con1, con2) = (con3, con2, con1)
  - $\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(R)) = \sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(\sigma_{<\text{cond1}>}(R)))$
- A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
  - $\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(R)) = \sigma_{<\text{cond1}> \wedge <\text{cond2}> \wedge <\text{cond3}>}(R)))$
- The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input relation R $\leq$

PROJECT (π, Π)

- PROJECT operation is denoted by π (pi), this operation keeps certain *columns* (attributes) from a relation and discards the other columns
  - PROJECT creates a vertical partitioning
- The general form of the *project* operation is:

$$\pi_{\text{<attribute list>}}(R)$$

  - π (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R
- The project operation *removes any duplicate tuples*
  - This is because the result of the *project* operation must be a *set of tuples*

PROJECT π_L(R)

- The projection operator π allows us to extract attributes from a relation

  delete unrelated list

  - Deletes attributes that are not in projection list L
  - Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation

Relation R

a	b	c
α	10	1
α	20	1
β	30	1
β	40	2

unrelated

a	c
α	1
α	1
β	1
β	2

$\pi_{a,c}$ (**R**) →

Eliminate Duplicated rows

a	c
α	1
β	1
β	2

Example: PROJECT $\pi_L(R)$

1. Find the employee names and department number of all employees (means the other attribute lists should be discarded)
2. Find the sex and department number of all employees

f_name	l_name	id	sex	salary	superid	dno
Joseph	Chan	999999	M	2950	654321	4
Victor	Wong	001100	M	3000	888555	5
Carrie	Kwan	898989	F	2600	654321	4
Joyce	Fong	345345	F	1200	777888	4

$\pi_{lname,fname,dno}(Employee)$

f_name	l_name	dno
Joseph	Chan	4
Victor	Wong	5
Carrie	Kwan	4
Joyce	Fong	4

$\pi_{sex,dno}(Employee)$

sex	dno
M	4
M	5
F	4

PROJECT (π, Π) Properties

- The number of tuples in the result of projection $\pi_{<list>}(R)$ is *always less or equal to* the number of tuples in R

  primary key?

  - If the list of attributes includes a *key* of R, then the number of tuples in the result of PROJECT is *equal* to the number of tuples in R
- PROJECT is *not* commutative
  - $\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$ as long as <list2> contains the attributes in <list1>

Relational Algebra Expressions

- We may want to apply several relational algebra operations one after the other
  - Either we can write the operations as a single relational algebra expression by nesting the operations, or
  - We can apply one operation at a time and create intermediate result relations.
- In the latter case, we must give *names* to the relations that hold the intermediate results
- For example: $\pi_{lname,fname}(\sigma_{dno=5}(\text{EMPLOYEE}))$, we can apply one operation at a time:
  - assignment operator
  - $\text{DEP5\_EMPS} \leftarrow \sigma_{dno=5}(\text{EMPLOYEE})$
  - $\text{RESULT} \leftarrow \pi_{lname,fname}(\text{DEP5\_EMPS})$

Example: Compositing Operations

- Substituting an expression where a relation is expected
- The result of an relational-algebra expression is always a relation.

f_name	l_name	id	sex	salary	superid	dno
Joseph	Chan	999999	M	29500	654321	4
Victor	Wong	001100	M	30000	888555	5
Carrie	Kwan	898989	F	26000	654321	4
Joyce	Fong	345345	F	12000	777888	4

$$\pi_{lname,fname}(\sigma_{dno=4 \wedge salary>25000}(Employee))$$

OR

$$R1 \leftarrow \sigma_{dno=4 \wedge salary>25000}(Employee)$$
$$\pi_{lname,fname}(R1)$$

f_name	l_name
Joseph	Chan
Carrie	Kwan

RENAME (ρ)

- The RENAME operator is denoted by ρ (rho)
- Allows to name and therefore to refer to the result of relational algebra expression.
- Allows to refer to a relation by more than one name (e.g., if the same relation is used twice in a relational algebra expression)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
  - Useful when a query requires multiple operations
  - Necessary in some cases (see JOIN operation later)

RENAME (ρ)

- The general RENAME operation ρ can be expressed by:
  - $\rho(R(A_1 \rightarrow B_1, A_2 \rightarrow B_2, \ldots, A_n \rightarrow B_n), E)$ or $\rho(R, E)$, with $E(A_1, \ldots, A_n)$, changes both:
    - the relation name to R, *and*
    - the column (attribute) names $\boxed{\text{to}}$ $B_1, \ldots, B_n$
    - the position number *i*-th can also be used
    - the resulting relation of $\rho(R, E)$ is R
    - $R \leftarrow E$ is same as $\rho(R, E)$

Example: RENAME (ρ)

- Renaming by attribute name:
  - $\rho(C(sid \rightarrow identity), E)$
  - We may rename more attributes:
  - $\rho(C(sid \rightarrow identity, child \rightarrow dependent), E)$
- Renaming by attribute position:
  - $\rho(C(3 \rightarrow identity), E)$
  - Result is C
  - The 3rd attribute in E is renamed as "*identity*" in C

Example: RENAME (ρ)

$$\rho(R(A_1 \rightarrow B_1, A_2 \rightarrow B_2, \ldots, A_n \rightarrow B_n), E)$$

- The new relation R has the same instance as E, but its schema has attribute B_i instead of attribute A_i
- Necessary if we need to perform a cartesian product or join of a table with itself

ρ(Staff(Name $\rightarrow$ Family_Name, Salary $\rightarrow$ Gross_salary), Employee)

Employee

Name	Salary	Emp_No
Clark	150000	1006
Gates	5000000	1005
Jones	50000	1001
Peters	45000	1002
Phillips	25000	1004
Rowe	35000	1003
Warnock	500000	1007

Staff

Family_Name	Gross_Salary	Emp_No
Clark	150000	1006
Gates	5000000	1005
Jones	50000	1001
Peters	45000	1002
Phillips	25000	1004
Rowe	35000	1003
Warnock	500000	1007

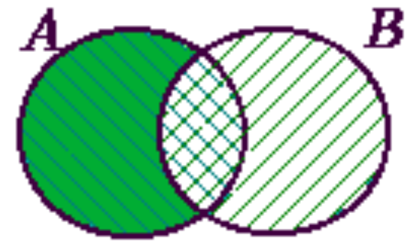
Outline

- **Relational Algebra**
  - Unary Relational Operations
  - **Binary Relational Operations**
  - Additional Relational Operations
- Introduction to Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus

Operations from Set Theory

- Type Compatibility of operands is required for the binary set operation UNION $\cup$, (also for INTERSECTION $\cap$, and SET DIFFERENCE –)
- $R1(A_1, A_2, ..., A_n)$ and $R2(B_1, B_2, ..., B_n)$ are type compatible if:
  - They have *the same number of attributes*, and
  - The domains of corresponding attributes are *type compatible* (i.e. $dom(A_i) = dom(B_i)$ for $i = 1, 2, ..., n$)
- The resulting relation for $R1 \cup R2$ (also for $R1 \cap R2$, or $R1 – R2$) has the same attribute names as the first operand relation R1 (by convention)

SET DIFFERENCE (–)



- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S
- The two operand relations R and S must be "type compatible"
- Example: **Plane_1 – Plane_2**

Maker1	Model_No1
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340
MD	DC10
MD	DC9

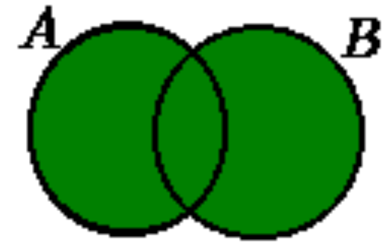
–

Maker2	Model_No2
Boeing	B727
Boeing	B747
Boeing	B757
MD	DC10
MD	DC9

=

Maker1	Model_No1
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340

UNION (∪)



- Binary operation UNION, denoted by ∪
- The result of R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S
- Duplicate tuples are eliminated
- The two operand relations R and S must be "type compatible"
- Example: **Plane_1 ∪ Plane_2**

Maker1	Model_No1
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340
MD	DC10
MD	DC9

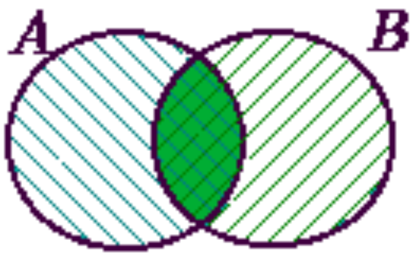
∪

Maker2	Model_No2
Boeing	B727
Boeing	B747
Boeing	B757
MD	DC10
MD	DC9

=

Maker1	Model_No1
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340
Boeing	B727
Boeing	B747
Boeing	B757
MD	DC10
MD	DC9

INTERSECTION (∩)



- The result of the operation $R \cap S$, is a relation that includes all tuples that are in both R and S
- Intersection is not considered a basic operation, as it can be derived from the basic operations: $R \cap S = R - (R - S)$
- The two operand relations R and S must be "type compatible"
- Example: $\mathbf{Plane_1} \cap \mathbf{Plane_2}$

Maker1	Model_No1
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340
MD	DC10
MD	DC9

∩

Maker2	Model_No2
Boeing	B727
Boeing	B747
Boeing	B757
MD	DC10
MD	DC9

=

Maker1	Model_No1
MD	DC9
MD	DC10

CARTESIAN PRODUCT (×)

- CARTESIAN (or CROSS) PRODUCT Operation is used to combine tuples from two relations in a combinatorial fashion
  - Denoted by $R(A_1, A_2, \ldots, A_n) \times S(B_1, B_2, \ldots, B_m)$
  - Result is a relation Q with degree n + m attributes:
    - $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$, in that order
  - The resulting relation state has one tuple for each combination of tuples - one from R and one from S
  - Hence, if R has n_R tuples rows (denoted as $|R| = n_R$), and S has n_S tuples, then $R \times S$ will have $n_R \cdot n_S$ tuples
- Generally, CROSS PRODUCT is not a meaningful operation
  - Can become meaningful when followed by other operations

Example: CARTESIAN PRODUCT (×)

- Combines each row of one table with every row of another table
- Can_fly × Plane

Emp_No	Model_No
1001	B727
1001	B747
1001	DC10
1002	A320
1002	A340
1002	B757
1002	DC9
1003	A310
1003	DC9

×

Maker	Model_No
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340
Boeing	B727
Boeing	B747
Boeing	B757
MD	DC10
MD	DC9

=

Emp_No	Model_No	Maker	Model_No
1001	B727	Airbus	A310
1001	B727	Airbus	A320
1001	B727	Airbus	A330
1001	B727	Airbus	A340
1001	B727	Boeing	B727
1001	B727	Boeing	B747
1001	B727	Boeing	B757
1001	B727	MD	DC10
1001	B727	MD	DC9
1001	B747	Airbus	A310
1001	B747	Airbus	A320
1001	B747	Airbus	A330
1001	B747	Airbus	A340
1001	B747	Boeing	B727
1001	B747	Boeing	B747
1001	B747	Boeing	B757
1001	B747	MD	DC10
1001	B747	MD	DC9
1001	B727	Airbus	A310
1001	B727	Airbus	A320
…	…	…	…

81 tuples!!!

CARTESIAN PRODUCT (×)

- $R \times S$ returns a relation instance whose schema contains all the attributes of R followed by all the attributes of S
- To keep only combinations of tuples meaningful, typically, we add a SELECT operation after Cartesian product

R

a	b
α	1
β	2

S

c	d	e
α	10	+
β	10	+
β	20	-
γ	10	-

R × S

a	b	c	d	e
α	1	α	10	+
α	1	β	10	+
α	1	β	20	-
α	1	γ	10	-
β	2	α	10	+
β	2	β	10	+
β	2	β	20	-
β	2	γ	10	-

$\sigma_{a=c}(R \times S)$

a	b	c	d	e
α	1	α	10	+
β	2	β	10	+
β	2	β	20	-

JOIN ($\bowtie_{\text{<join condition>}}$)

- The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations
- A special operation, called JOIN combines this sequence into a single operation
- This operation is *very important* for any relational database with more than a single relation, because it allows us *combine related tuples* from various relations
- The general form of a join operation on two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is:

$$R \bowtie_{\text{<join condition>}} S = \sigma_{\text{<join condition>}}(R \times S)$$

where R and S can be any relations that result from general *relational algebra expressions*

EQUIJOIN

- EQUIJOIN Operation is the most common use of join involv~~es~~ing join conditions with equality comparisons only
- Such a join, where the only comparison operator used is =, is called an EQUIJOIN
  - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple
  - The JOIN seen in the previous example was an EQUIJOIN

Example: EQUIJOIN

- A condition join where the condition contains only equalities.
- The conditions of the join have the form R.A = S.B, where we want to join R with S

R

a	b	c
α	1	α
β	5	γ
γ	4	β
α	1	γ
δ	2	β

S

b	e	f
1	X	α
3	X	β
1	X	γ
2	Y	δ
3	Y	ϵ

$R \bowtie_{R.b=S.b} S$

a	b	c	e	f
α	1	α	X	α
α	1	α	X	γ
α	1	γ	X	α
α	1	γ	X	γ
δ	2	β	Y	δ

Some Properties of JOIN

- JOIN

$$R(A_1, A_2, \ldots, A_n) \bowtie_{R.A_i=S.B_j} S(B_1, B_2, \ldots, B_m)$$

- Result is a relation Q has *n + m* attributes:
  - $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$, in that order
- The resulting relation state has one tuple for each combination of tuples - r from R and s from S, but *only if they satisfy the join condition* $r[A_i]=s[B_j]$ ($r.A_i = s.B_j$)
- Hence, if R has n_R tuples, and S has n_S tuples, then the join result will generally have *less than* $n_R \cdot n_S$ tuples
- Only related tuples (based on the join condition) will appear in the result

NATURAL JOIN ($\bowtie$)

- NATURAL JOIN ($\bowtie$), is a further special case of the join operation, which is an EQUIJOIN in which equalities are specified on *all fields having the same name* in R and S
    - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations
    - If this is not the case, a renaming operation is applied first
    - We can simply omit the join condition

R

a	b	c	d
α	1	α	X
β	2	γ	X
γ	4	β	Y
α	1	γ	Y
δ	2	β	Y

S

b	d	e
1	X	α
3	X	β
1	X	γ
2	Y	δ
3	Y	ε

R $\bowtie$ S

a	b	c	d	e
α	1	α	X	α
α	1	α	X	γ
δ	2	β	Y	δ

Example: NATURAL JOIN ($\bowtie$)

- Can_fly $\bowtie$ Plane

Emp_No	Model_No
1001	B727
1001	B747
1001	DC10
1002	A320
1002	A340
1002	B757
1002	DC9
1003	A310
1003	DC9

$\bowtie$

Maker	Model_No
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340
Boeing	B727
Boeing	B747
Boeing	B757
MD	DC10
MD	DC9

=

Emp_No	Model_No	Maker
1003	A310	Airbus
1002	A320	Airbus
1002	A340	Airbus
1001	B727	Boeing
1001	B747	Boeing
1002	B757	Boeing
1001	DC10	MD
1002	DC9	MD
1003	DC9	MD

Example: Complex Query

- Let us try a more complex query
- Query: Retrieve for each ***female employee*** a list of the ***names*** of her ***dependents***

EMPLOYEE

f_name	l_name	id	bdate	addr	sex	salary	super_id	dno

DEPENDENT

eid	dependent-name	sex	bdate	relationship

Example:

EMPLOYEE

f_name	l_name	id	bdate	addr	sex	salary	super_id	dno

DEPENDENT

eid	dependent-name	sex	bdate	relationship

- Query: Retrieve for each

  female employee a list of the ***names*** of her ***dependents***

- Step1 (Female Emps): FemaleEmps $\leftarrow \sigma_{sex='F'}(Employee)$
- Step2 (Their IDs): EmpNames $\leftarrow \pi_{fname,lname,id}$(FemaleEmps)

FemaleEmps

f_name	l_name	id	bdate	address	sex	salary	superid	dno
Alicia	Chan	998877	2-Jul-70	231, Cai Road, HK	F	9500	654321	4
Jennifer	Wong	654321	20-June-60	342, Cheung Road, HK	F	30000	888555	4
Joyce	Fong	345345	19-Dec-80	23, Young Road, HK	F	12000	777888	5

Example:

EMPLOYEE

f_name	l_name	id	bdate	addr	sex	salary	super_id	dno

DEPENDENT

eid	dependent-name	sex	bdate	relationship

- Query: Retrieve for each

  female employee a list of the ***names*** of her ***dependents***.

- Step3 (Their dependents): EmpNames $\bowtie_{id=eid}$ Dependents

$= \sigma_{id=eid}(EmpNames \times Dependents)$

EmpNames

f_name	l_name	id
Alicia	Chan	998877
Jennifer	Wong	654321
Joyce	Fong	345345

Dependents

eid	dep_name	sex	bdate	relationship
334455	Alice	F	5-Apr-90	Daughter
334455	Theodore	M	3-Mar-92	Son
654321	Abner	M	29-Feb-94	Son
123456	Alice	F	2-Nov-97	Daughter

$$ActualDependents \leftarrow EmpNames \bowtie_{id=eid} Dependents$$

$$= \sigma_{id=eid}(EmpNames \times Dependents)$$

FEmpNamesID × Dependents

f_name	l_name	id	eid	dep_name	sex	bdate	relationship
Alicia	Chan	998877	334455	Alice	F	5-Apr-90	Daughter
Alicia	Chan	998877	334455	Theodore	M	3-Mar-92	Son
Alicia	Chan	998877	654321	Abner	M	29-Feb-94	Son
Alicia	Chan	998877	123456	Alice	F	2-Nov-97	Daughter
Jennifer	Wong	654321	334455	Alice	F	5-Apr-90	Daughter
Jennifer	Wong	654321	334455	Theodore	M	3-Mar-92	Son
Jennifer	Wong	654321	654321	Abner	M	29-Feb-94	Son
Jennifer	Wong	654321	123456	Alice	F	2-Nov-97	Daughter
Joyce	Fong	345345	334455	Alice	F	5-Apr-90	Daughter
Joyce	Fong	345345	334455	Theodore	M	3-Mar-92	Son
Joyce	Fong	345345	654321	Abner	M	29-Feb-94	Son
Joyce	Fong	345345	123456	Alice	F	2-Nov-97	Daughter

EMPLOYEE

f_name	l_name	id	bdate	addr	sex	salary	super_id	dno

DEPENDENT

eid	dependent-name	sex	bdate	relationship

Example:

- Query: Retrieve for each

  female employee a list of the ***names*** of her ***dependents***.

- Step4: $\pi_{fname,lname,dep\_name}(ActualDependents)$

ActualDependents

f_name	l_name	id	eid	dep_name	sex	bdate	relationship
Jennifer	Wong	654321	654321	Abner	M	29-Feb-94	Son

Result

f_name	l_name	dep_name
Jennifer	Wong	Abner

Example:

EMPLOYEE

f_name	l_name	id	bdate	addr	sex	salary	super_id	dno

DEPENDENT

eid	dependent-name	sex	bdate	relationship

- Query: Retrieve for each

  female employee a list of the ***names*** of her ***dependents***.

- Step1: FemaleEmps $\leftarrow \sigma_{sex='F'}$ (*Employee*)
- Step2: EmpNames $\leftarrow \pi_{fname,lname,id}$(FemaleEmps)
- Step3: *ActualDependents* $\leftarrow$ *EmpNames* $\bowtie_{id=eid}$ *Dependents*
- Step4: $\pi_{fname,lname,dep\_name}$(*ActualDependents*)

Result

f_name	l_name	dep_name
Jennifer	Wong	Abner

Example:

EMPLOYEE

f_name	l_name	id	bdate	addr	sex	salary	super_id	dno

DEPENDENT

eid	dependent-name	sex	bdate	relationship

- Query: Retrieve for each

  female employee a list of the ***names*** of her ***dependents***.

- Another Solution:

$$\pi_{fname,lname,id}\,(\sigma_{sex='F'}\,(Employee \bowtie_{id=eid} Dependents))$$

Result

f_name	l_name	dep_name
Jennifer	Wong	Abner

FemaleEmps

f_name	l_name	id	bdate	address	sex	salary	superid	dno
Alicia	Chan	998877	2-Jul-70	231, Cai Road, HK	F	9500	654321	4
Jennifer	Wong	654321	20-June-60	342, Cheung Road, HK	F	30000	888555	4
Joyce	Fong	345345	19-Dec-80	23, Young Road, HK	F	12000	777888	5

EmpNames

f_name	l_name	id
Alicia	Chan	998877
Jennifer	Wong	654321
Joyce	Fong	345345

$\bowtie$

Dependents

eid	dep_name	sex	bdate	relationship
334455	Alice	F	5-Apr-90	Daughter
334455	Theodore	M	3-Mar-92	Son
654321	Abner	M	29-Feb-94	Son
123456	Alice	F	2-Nov-97	Daughter

ActualDependents

f_name	l_name	id	eid	dep_name	sex	bdate	relationship
Jennifer	Wong	654321	654321	Abner	M	29-Feb-94	Son

Result

f_name	l_name	dep_name
Jennifer	Wong	Abner

Condition Join (θ-Join)

- The general case of JOIN operation is called a θ-join:

$$R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$$

- The join condition is called *theta* (θ).
- *Theta* (θ) can be any general boolean expression on the attributes of R and S; for example:
  - $R.A_i \neq S.A_i$
  - $R.A_i < S.A_i \vee (R.A_i = S.A_i \wedge R.A_j < S.A_j)$ ..

Condition Join (θ-Join)

- The general case of JOIN operation is called a θ-join:

$$R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$$

- Simple Example:

S

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.5

R

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$$S \bowtie_{S.sid<R.sid} R$$

S.sid	sname	rating	age	R.sid	bid	day
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

Example: θ-Join

- We have a Flight table that records the Flight Number, Origin, Destination, Departure Time and Arrival Time.
- We join this table with itself (*self-join*) using the condition:
- Flight1.Dest = Flight2.Origin $\wedge$ Flight1.Arr_Time < Flight2.Dept_Time
- What should we get?

Flight1

Num	Origin	Dest	Dep_Time	Arr_Time
334	ORD	MIA	12:00	14:14
335	MIA	ORD	15:00	17:14
336	ORD	MIA	18:00	20:14
337	MIA	ORD	20:30	23:53
394	DFW	MIA	19:00	21:30
395	MIA	DFW	21:00	23:43

$\overset{\dots}{\bowtie}$

Flight2

Num	Origin	Dest	Dep_Time	Arr_Time
334	ORD	MIA	12:00	14:14
335	MIA	ORD	15:00	17:14
336	ORD	MIA	18:00	20:14
337	MIA	ORD	20:30	23:53
394	DFW	MIA	19:00	21:30
395	MIA	DFW	21:00	23:43

Example: θ-Join

- We have a Flight table that records the Flight Number, Origin, Destination, Departure Time and Arrival Time.

$$\text{Flight1.Dest} = \text{Flight2.Origin} \wedge \text{Flight1.Arr\_Time} < \text{Flight2.Dept\_Time}$$

Flight1. Num	Flight1. Origin	Flight1 .Dest	Flight1.De p_Time	Flight1.Ar r_Time	Flight2_ 1.Num	Flight2.Or igin	Flight2. Dest	Flight2.Dep _Time	Flight2.Arr_ Time
334	ORD	MIA	12:00	14:14	335	MIA	ORD	15:00	17:14
335	MIA	ORD	15:00	17:14	336	ORD	MIA	18:00	20:14
336	ORD	MIA	18:00	20:14	337	MIA	ORD	20:30	23:53
334	ORD	MIA	12:00	14:14	337	MIA	ORD	20:30	23:53
336	ORD	MIA	18:00	20:14	395	MIA	DFW	21:00	23:43
334	ORD	MIA	12:00	14:14	395	MIA	DFW	21:00	23:43

What happens if we add the condition $\text{Flight1.Origin} \neq \text{Flight2.Dest}$

Example: θ-Join

EMPLOYEE

f_name	l_name	id	bdate	addr	sex	salary	super_id	dno

- Another Example.
- Query: Find the names of employees with the *highest* salary.
- Solution:

$$\pi_{lname,fname}(Employee) -$$

$$\pi_{Employee.lname,\ Employee.fname}($$

$$Employee \bowtie_{Employee.salary<F.salary} \rho(F, Employee))$$

- $Employee \bowtie_{Employee.salary>F.salary} \rho(F, Employee))$

Why this does not work?

Extra example: S(eid, ename, sex, age), find the student who is youngest.

① $\rho(S_1, S), \rho(S_2, S)$ ② $S_1 \bowtie_{S_1.age > S_2.age} S_2$ ③ $S_3 \leftarrow S_1 \bowtie_{S_1.age > S_2.age} S_2$ ④ $S_4 \leftarrow (S_1 \times S_2) - S_3$

Complete Set of Relational Operations

- The set of operations including SELECT (σ), PROJECT (π), RENAME (ρ), UNION ($\cup$), DIFFERENCE ($-$), and CARTESIAN PRODUCT ($\times$) is called a *complete set* because any other relational algebra expression can be expressed by a combination of these five operations.
- For example:
  - $R \cap S = R \cap S = R - (R - S)$
  - $R \bowtie_{\text{<join condition>}} S = \sigma_{\text{<join condition>}}(R \times S)$

DIVISION

U
Y

=

R	
Y	X

/

S
X

- The DIVISION is applied to two relations, R and S
  - $R_{\{Y, X\}} / S_{\{X\}}$, where {Y} denote the attribute name set Y
  - The result of DIVISION is a relation $U_{\{Y\}}$ that includes tuples t_U in U if tuples $t_R = <t_U, t_S>$ appear in R with $t_R[Y] = t_U$, and with $t_R[X] = t_S$ *for every tuple* t_S in S
  - For a tuple t to appear in the result U of the DIVISION, the values in t must appear in R in combination with *every* tuple in S
  - $r / s := \{ t \mid t \in \pi_{R\text{-}S}(r) \wedge \forall u \in s : <t, u> \in r \}$

DIVISION

- The division operation is useful for expressing certain kinds of queries, for example, “find the names of student who have passed all courses.”
- Consider two relations A and B
  - $A_{\{x,\ y\}}$ has exactly two attributes x and y
  - $B_{\{y\}}$ has just one attribute y, with the same domain as in A
  - A/B contains all x tuples, such that for every y tuple in B there is a <x,y> tuple in A

A(x,y)/B(y)

A

x	y
s1	p2
s1	p3
s1	p4
s2	p2
s3	p2
s4	p2
s4	p4

/

B

y
p2
p4

=

A/B

x
s1
s4

DIVISION

A

x	y
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ε	1
ε	2

B

y
1
2

A / B

x
α
ε

- The division operation A/B is the set of all x values (in the form of unary records)
- Such that for every y value in a record of B, there is a record <x,y> in A

DIVISION

A

x	y
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ε	1
ε	2

B

y
1
2

A / B

x
α
ε

- Another way to understand division:
- For each x value in A, consider the set of y values that appear in records of A with that x value
- If this set contains all y values in B, the x value is in the result of A/B

DIVISION

A

x	y
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ε	1
ε	2

B

y
1
2

A / B

x
α
ε

- An analogy with integer division:
- For integers A and B, A/B is the largest integer Q such that $Q \times B \leq A$
- For relation instances A and B, A/B is the largest relation instance Q such that $Q \times B \subseteq A$

- The division can be written in terms of basic operations:

A

x	y
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ε	1
ε	2

B

y
1
2

A/B:

$$\text{Temp1} \leftarrow \pi_{\text{attri(A)-attri(B)}}(A)$$

$$\text{Temp2} \leftarrow \pi_{\text{attri(A)-attri(B)}}((\text{Temp1} \times B) - A)$$

$$A/B \leftarrow \text{Temp1} - \text{Temp2}$$

Temp1

x
α
β
γ
δ
ε

$(Temp1 \times B) - A$

x	y
β	2
γ	2
δ	2

Temp2

A / B

x
α
ε

Example: DIVISION

- Find all student IDs (sids) of the students who took ***all*** courses in table Course

Take / Course

Take

sid	cid
1	231
1	170
1	111
1	001
2	231
2	001
3	170
4	231
4	170

/

Course

cid
231
170

=

sid
1
4

Example: DIVISION

- Find all student IDs (sids) of the students who took ***all*** courses provided by ***CES*** department

$$\text{Take} \;/\; \pi_{cid}(\sigma_{dept\,=\,\text{“CES”}}(\text{Course}))$$

Take

sid	cid
1	231
1	170
1	111
1	001
2	231
2	001
3	170
4	231
4	170

Course

cid	dept
231	CSE
170	CSE
001	LANG
111	ECE
123	ECE

Example: Complex Query

- Query: Retrieve the names of employees who work on all the projects that 'John Sung' works on.

Employee

fname	lname	id	bdate	address	salary	sid	dno

Works_on

id	pno

Step1: Sung $\leftarrow \sigma_{\text{fname="John"} \wedge \text{lname="Sung"}}$(Employee)

Step2: Sung_pnos $\leftarrow \pi_{\text{pno}}$(Works_on $\bowtie$ Sung)

fname	lname	id	bdate	address	salary	sid	dno	pno

pno

Step3: Result_id $\leftarrow$ Works_on / Sung_pnos

Step4: Result $\leftarrow \pi_{\text{fname,lname}}$(Result_id $\bowtie$ Employee)

More Examples

Sailors (<u>sid</u>, sname, age)

Boats (<u>bid</u>, bname, color)

Reserves (<u>sid, bid, date</u>)

- Consider the above schemas, the primary key fields are underlined.

More Examples

Sailors (<u>sid</u>, sname, age)
Boats (<u>bid</u>, bname, color)
Reserves (<u>sid, bid, date</u>)

- Query 1: Find the names of sailors who have reserved boat with bid = 103

  – Solution 1:
  $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors})$

  More Efficient

  – Solution 2:
  $\rho(\text{Temp1}, \sigma_{bid=103} \text{Reserves})$
  $\rho(\text{Temp2}, \text{Temp1} \bowtie \text{Sailors})$
  $\pi_{sname}(\text{Temp2})$

  – Solution 3:
  $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors}))$

More Examples

Sailors (sid, sname, age)
Boats (bid, bname, color)
Reserves (sid, bid, date)

- Query 2: Find the names of sailors who have reserved at least a red boat
  - Solution 1:
    π_{sname} (($\sigma_{color='red'}$ Boats) $\bowtie$ Reserves $\bowtie$ Sailors)
  - Solution 2:
    π_{sname} (π_{sid} ((π_{bid} $\sigma_{color='red'}$ Boats) $\bowtie$ Reserves) $\bowtie$ Sailors)

More Examples

Sailors (sid, sname, age)
Boats (bid, bname, color)
Reserves (sid, bid, date)

- Query 3: Find the names of sailors who have reserved at least a red or a green boats
  - We can identify all red or green boats, then find sailors who have reserved one of these boats

— Solution 1:

ρ (Tempboats, ($\sigma_{\text{color='red'} \vee \text{color='green'}}$ Boats))

π_{sname} (Tempboats $\bowtie$ Reserves $\bowtie$ Sailors)

What happens if $\vee$ is replaced by $\wedge$ in this query?

— Solution 2:

π_{sname} (($\sigma_{\text{color='red'}}$ Boats) $\bowtie$ Reserves) $\bowtie$ Sailors)

$\cup\ \pi_{\text{sname}}$ (($\sigma_{\text{color='green'}}$ Boats) $\bowtie$ Reserves) $\bowtie$ Sailors)

More Examples

Sailors (<u>sid</u>, sname, age)
Boats (<u>bid</u>, bname, color)
Reserves (<u>sid, bid, date</u>)

- Query 4: Find the names of sailors who have reserved at least a red boat and at least a green boat
  - The previous Solution 1 would not work
  - We must identify sailors who have reserved red boats, sailors who have reserved green boats, then find the intersection
  - Note that sid is a key for Sailors

  – Solution:

  $\rho(\text{Tempred}, \pi_{sid}((\sigma_{color='red'} \text{Boats}) \bowtie \text{Reserves}))$

  $\rho(\text{Tempgreen}, \pi_{sid}((\sigma_{color='green'} \text{Boats}) \bowtie \text{Reserves}))$

  $\pi_{sname}((\text{Tempred} \cap \text{Tempgreen}) \bowtie \text{Sailors})$

More Examples

Sailors (sid, sname, age)
Boats (bid, bname, color)
Reserves (sid, bid, date)

- Query 5: Find the names of sailors who have reserved all boats
  - Solution:
    $\rho($ Tempsids, $(\pi_{sid,bid}$ Reserves $) / (\pi_{bid}$ Boats $)\,)$
    π_{sname} (Tempsids $\bowtie$ Sailors)

What if we simply do, Reserves / $(\pi_{bid}$ Boats) ?

date	sid	bid
2-3-2002	007	A
3-7-2002	007	B

Returns nothing,
When we expect 007

date	sid	bid
2-3-2002	007	A
2-3-2002	007	B

Returns (2-3-2002, 007)

More Examples

Sailors (<u>sid</u>, sname, age)
Boats (<u>bid</u>, bname, color)
Reserves (<u>sid, bid, date</u>)

- Query 6: Find sailors who have reserved all red boats:
  - Solution:
    $\rho($ Tempsids, $(\pi_{sid,bid}$ Reserves $) / (\pi_{bid} (\sigma_{color='red'}$ Boats $))$
    π_{sname} (Tempsids $\bowtie$ Sailors)

Summary

- The relational model has rigorously defined query languages that are simple and powerful
- Relational algebra is operational; useful as an internal representation for query evaluation steps
- Typically there are several ways of expressing a given query; a query optimizer should choose an efficient version