KNU CSE
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

THE RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

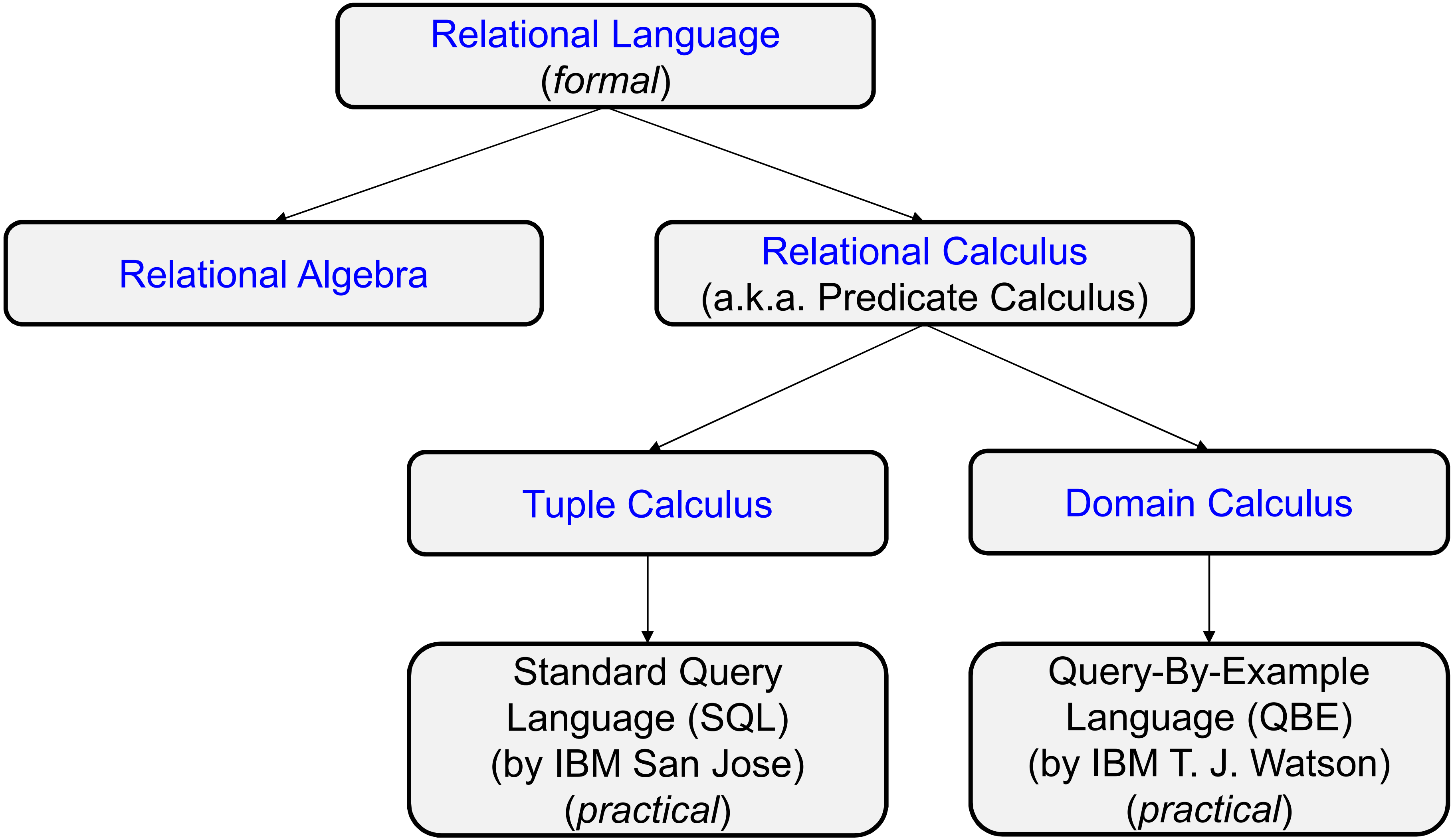
Chapter 8

- 관계 대수 및 관계 해석

Chapter Outline

- Background
- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations
  - Examples of Queries in Relational Algebra
- Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus (if time)

Background


Relational Language
(*formal*)
Relational Algebra
Relational Calculus
(a.k.a. Predicate Calculus)
Tuple Calculus
Domain Calculus
Standard Query Language (SQL)
(by IBM San Jose)
(*practical*)
Query-By-Example Language (QBE)
(by IBM T. J. Watson)
(*practical*)

Background – Relational Algebra

- What is ***relational algebra***?
  - The basic set of operations for the relational model
  - Enables a user to specify basic retrieval requests, or ***queries***.
- The result of an operation: a ***new relation***
  - The relation may have been formed from one or more ***input*** relations.
  - This property makes the algebra "closed" (all objects in relational algebra are relations). (관계 대수의 대상이 릴레이션이고 연산 결과도 릴레이션이므로 관계 대수는 릴레이션들에서만 적용되는 효과를 가지고 옴.)
  - Producing new relations can be further manipulated using operations of the "same" algebra.
- A sequence of relational algebra operations forms a ***relational algebra expression***.
  - The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request).

Background – Relational Algebra (Cont'd)

- Why is relational algebra *important*? Three reasons.

  1) It provides a **formal foundation** for relational model operations.

  2) It is used as a basis for implementing and **optimizing** queries in the query processing and optimization modules that are integral parts of RDBMSs.

  3) Some of its concepts are **incorporated into** the SQL standard query language for RDBMSs.

  - The core operations and functions in **the internal modules** of most relational systems are based on relational algebra operations.

Background – Relational Algebra (Cont'd)

- Its classic operations can be divided into two groups.
  - G1) includes set operations from mathematical set theory.
    - Remind that each relation is a set of tuples in the *formal* relational model.
    - Ex) `UNION`(∪), `INTERSECTION`(∩), `SET DIFFERENCE`(–), `CROSS (CARTESIAN) PRODUCT`(x)
  - G2) consists of operations for relational databases.
    - **Unary operations**: `SELECT`(symbol: σ (sigma)), `PROJECT`(symbol: π (pi)), `RENAME`(symbol: ρ(rho))
    - **Binary operations**: `JOIN` and `DIVISION`
- Besides, some *additional* operations were added.
  - **Aggregate functions** computing summary of data (`SUM`, `COUNT`, `AVG`, `MIN`, `MAX`), **OUTER JOIN**s, and OUTER UNIONs (skipped this semester)
    - These operations were added due to their importance to many DB applications.

Background – Relational Calculus

- ***Relational calculus*** provides a higher-level *declarative* language for specifying relational queries.
  - Also another formal relational language and has a firm basis in mathematical logic (called *predicate calculus*).
  - In a relational calculus expression, there is **NO** *order of operations* to specify how to retrieve the query results (like SQL), that is, “only what information” the result should be contain.

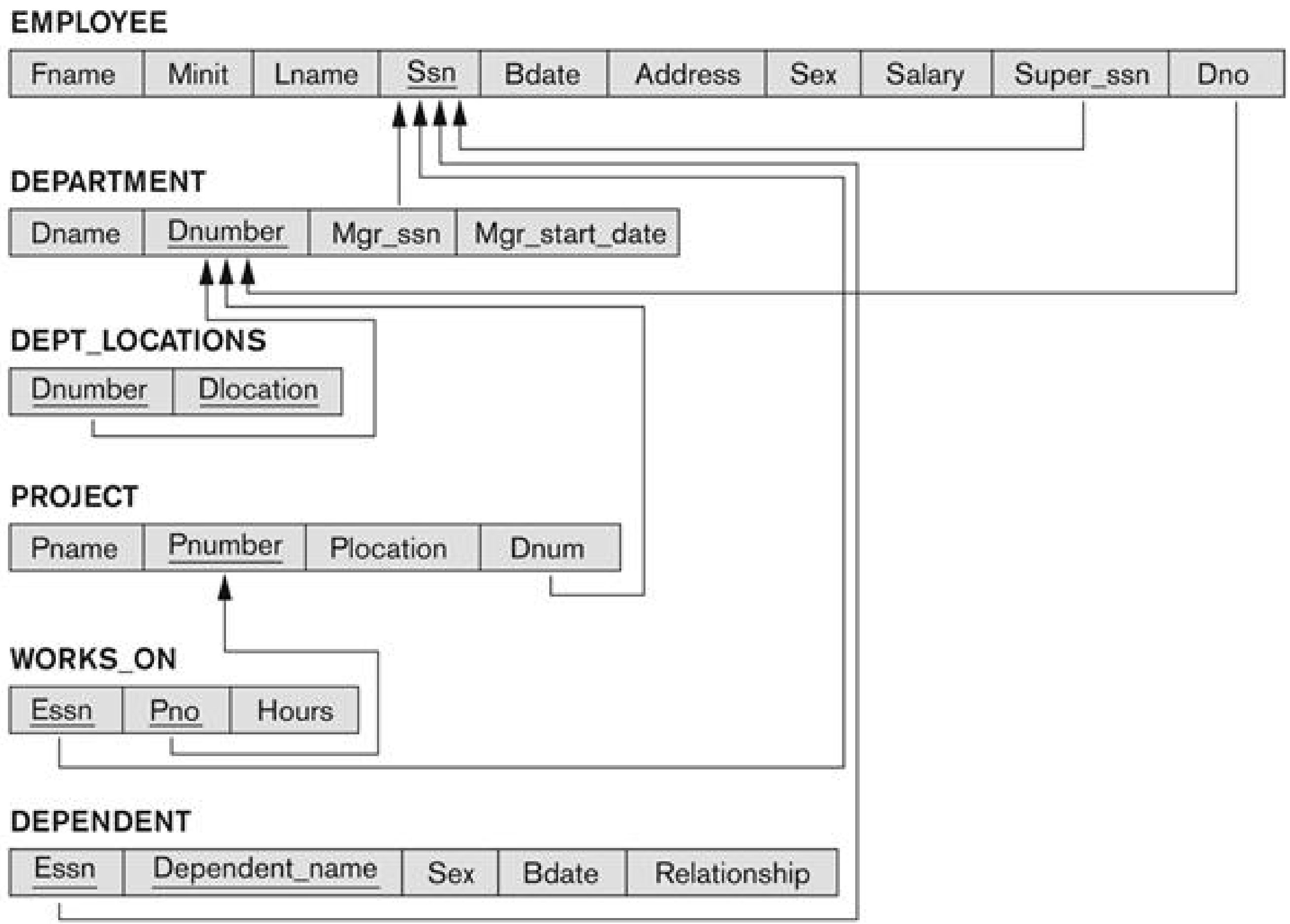
    => The main difference from relational algebra
  - It has two variations:
    - 1) *tuple relational calculus*: presents to the SQL for RDBMSs some of its foundations.
      - Use variables ranging over <u>tuples</u> (rows)
    - 2) *domain relational calculus:* another form of relational calculus
      - Use variables ranging over the <u>domains</u> (values) of attributes (columns).

UNARY RELATIONAL OPERATIONS: `SELECT` AND `PROJECT`

Chapter 8.1

Recall: the `COMPANY` Database

- All examples discussed refer to the `COMPANY` database below.



One Possible DB State for COMPANY

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

One Possible DB State for COMPANY (Cont'd)

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Unary Relational operation: `SELECT`

- Denoted by **σ** (sigma) (in Greek)
- Used to select a subset of the tuples from a relation based on a **selection condition**.
  - The selection condition acts as a filter.
  - Keeps only those tuples satisfying the qualifying condition.
    - Tuples satisfying the condition are selected whereas the others are not selected, or **filtered out**.
- Examples
  - "Selects the `EMPLOYEE` tuples whose department number is 4."

$$\sigma_{\texttt{Dno=4}}(\texttt{EMPLOYEE})$$

  - "Select the `EMPLOYEE` tuples whose salary is greater than $30,000."

$$\sigma_{\texttt{Salary>30000}}(\texttt{EMPLOYEE})$$

Unary Relational operation: SELECT (Cont'd)

- In general, the select operation is denoted by

$$\sigma_{<selection\ condition>}(R)\text{, where}$$

  - the symbol σ represents the "select operator", and
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R.
    - <*selection condition*> consists of a number of "clauses" of the form:

      <*attribute name*> <*comparison op*> <*constant value*>, where

      <*attribute name*>: the name of an attribute of R,

      <*comparison op*>: one of $\{=, <, \leq, >, \geq\}$, and

      <*constant value*>: a constant value from the attribute domain.
    - The clauses can be connected by: *and*, *or*, and *not* to form a condition.
  - tuples that make the (selection) condition true are **selected**
    - appear in the result of the (select) operation,
  - tuples that make the (selection) condition false are **filtered out**
    - discarded from the result of the (select) operation.

Unary Relational operation: SELECT (Cont'd)

- Several properties:

1) The SELECT operation $\sigma_{<selection\ condition>}(R)$ produces a relation S that has the same schema (same attributes) as R

2) σ is *commutative*:

$$\sigma_{<condition1>}(\sigma_{<condition2>}(R)) = \sigma_{<condition2>}(\sigma_{<condition1>}(R))$$

3) Because of commutativity property, a cascade (sequence) of σ operations may be applied in any order:

$$\sigma_{<cond1>}(\sigma_{<cond2>}(\sigma_{<cond3>}(R))) = \sigma_{<cond2>}(\sigma_{<cond3>}(\sigma_{<cond1>}(R)))$$

4) A cascade of σ operations may be replaced by a single σ with a conjunction of all the selection conditions:

$$\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<condn>}(R))\ldots)) = \sigma_{<cond1>\text{AND}<cond2>\text{AND}\ldots\text{AND}<condn>}(R)$$

5) The number of tuples in the result: smaller or equal to the number of tuples in the input relation R, denoted by $|\sigma_C(R)| \leq |R|$.

- But the number of attributes of the relation from σ: equal to the degree of R.

6) *Unary*(단항?); applied to a *single relation*, to *each tuple individually*.

Unary Relational operation: SELECT (Cont'd)

- Example:

$$\sigma_{(\text{Dno}=4 \textbf{ AND } \text{Salary}>25000) \textbf{ OR } (\text{Dno}=5 \textbf{ AND } \text{Salary}>30000)}(\text{EMPLOYEE})$$

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

- $\sigma_{\texttt{Dno=4 AND Salary>25000}}(\text{EMPLOYEE})$ // In relational algebra

```
SELECT *
FROM     EMPLOYEE
WHERE Dno=4 AND Salary>25000;
```

// A SQL Query

// σ => typically specified in the WHERE clause

Unary Relational operation: `PROJECT`

- Denoted by π (pi) (in Greek)
- Selects certain *columns* from a table and discards the other columns of that table.
  - The operation *projects* the relation over these attributes only.
  - Its result can be visualized as a *vertical partition* of the relation into two relations: one with the needed and another with the discarded
- Example: To list each employee's last name, first name, and salary, the following can be specified:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

⇩

```sql
SELECT Lname, Fname, Salary
FROM EMPLOYEE;
```

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

Unary Relational operation: `PROJECT` (Cont'd)

- In general, the project operation is denoted by

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

  - the symbol π represents the "project operator".
  - <*attribute list*>: the desired list of attributes from relational `R`
    - Again, note that R is a relational algebra expression whose result is a relation, which in the simplest case is a relation.
- π *removes any duplicate tuples*.
  - In the formal relational model, the result of π **MUST** be a set of tuples.
    - C.f. Mathematically a set cannot have duplicate elements.
  - This where the formal model is different from its practical model, for instance, SQL. Why? SQL?

Unary Relational operation: `PROJECT` (Cont'd)

- Several properties:
  1) The number of tuples in the result of $\pi_{<attribute\ list>}(R)$: $\leq |R|$
     - Why? But if the attribute list includes a *key* of R, then the degree of the result of $\pi_{<attribute\ list>}(R) = |R|$.
  2) π is **NOT** *commutative* (ex. *attr. list1* = {`Lname`, `Salary`}, *attr. list2*= {`Ssn`, `Salary`}):

$$\pi_{<attr.\ list1>}(\pi_{<attr.\ list2>}(R)) \neq \pi_{<attr.\ list2>}(\pi_{<attr.\ list1>}(R))$$

     - Note: The leftmost attribute list determines what the result of π.
     - If $<attribute\ list2> \supseteq <attribute\ list1>$ (ex. {`Sex`, `Salary`} ⊇ {`Salary`}), then

$$\pi_{<attr.\ list1>}(\pi_{<attr.\ list2>}(R)) = \pi_{<attr.\ list1>}(R)$$

- Example:

$\pi_{\texttt{Sex, Salary}}$ (`EMPLOYEE`)

```
SELECT DISTINCT Sex, Salary
FROM EMPLOYEE
```

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Applying Sequences of Operations: *Single Expression* vs. *Sequence of Operations*

- We may apply relational algebra operations one after the other. Some ways to express this:

  1) *Using an in-line expression*: nest the operations and write them as a **single relational algebra expression**.

$$\pi_{\texttt{Fname,Lname,Salarly}}\ (\sigma_{\ \texttt{DNO=5}}(\texttt{EMPLOYEE}))$$

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Applying Sequences of Operations: Single Expression vs. Sequence of Operations (Cont'd)

2) *Using the assignment operation*: Explicitly show the sequences of operations, giving a **name** to each intermediate relation, and using the assignment symbol (←).

$\texttt{DEP5\_EMPS} \leftarrow \sigma_{\texttt{Dno=5}}(\texttt{EMPLOYEE})$

$\texttt{RESULT}^{*} \leftarrow \pi_{\texttt{Fname,Lname,Salary}}(\texttt{DEP5\_EMPS})$

* **RESULT** will have the same attribute names as **DEPT5_EMPS**.

or, $\texttt{TEMP} \leftarrow \sigma_{\texttt{Dno=5}}(\texttt{EMPLOYEE})$

$\mathbf{R}\,(\texttt{First\_name,Last\_name,Salary}) \leftarrow \pi_{\texttt{Fname,Lname,Salary}}(\texttt{TEMP})$

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

```sql
SELECT E.Fname AS First_name,
       E.Lname AS Last_name,
       E.Salary AS Salary
FROM   EMPLOYEE AS E
WHERE  E.Dno = 5;
```

Unary Relational Operations: RENAME

- Denoted by ρ (rho) (in Greek)
- Works for renaming the attributes of a relation or the relation name or both: useful for complex queries involving JOIN
- ρ can be expressed by any of the following terms:
  - $\rho_{S(B1,\ B2,\ \ldots,\ Bn)}(R)$ changes:
    - the column (attribute) names to $B_1, B_2, \ldots, B_n$
  - $\rho(R)$ changes:
    - the relation name only to S
  - $\rho_{S(B1,\ B2,\ \ldots,\ Bn)}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to $B_1, B_2, \ldots, B_n$

RELATIONAL ALGEBRA OPERATIONS FROM SET THEORY

Chapter 8.2

UNION

- Denoted by $\cup$
- Binary operation
  - The result of $R \cup S$: a relation that includes all tuples 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 (or UNION) compatible**".
    - Suppose $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$.
    - For both to be **type-compatible**, i) the same degree of n and ii) $\mathrm{dom}(A_i) = \mathrm{dom}(B_i)$ for $1 \leq i \leq n$.
      - In other words, same attribute counts, same attribute domains
  - The type compatibility applies to other set operators: `INTERSECTION` and `SET DIFFERENCE` (`MINUS`).

UNION: Example

- “Retrieve the SSNs of all employees whose either
  - *Work in department 5* or
  - *Directly supervise an employee who works in department 5.”*

$\textbf{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$

$\textbf{RESULT1} \leftarrow \pi_{\text{SSN}}(\textbf{DEP5\_EMPS})$

$\textbf{RESULT2}(\text{SSN}) \leftarrow \pi_{\text{SUPERSSN}}(\textbf{DEP5\_EMPS})$ -- renaming: SUPERSSN to SSN

$\textbf{RESULT} \leftarrow \textbf{RESULT1} \cup \textbf{RESULT2}$

Ssn
123456789
333445555
666884444
453453453

$\cup$

Ssn
333445555
888665555

$=$

Ssn
123456789
333445555
666884444
453453453
888665555

The union operation produces the tuples that are in either **RESULT1** or **RESULT2** or both.

INTERSECTION

- Denoted by $\cap$
- Binary operation
  - The result of $R \cap S$: a relation that includes all tuples in both R and S.
    - R and S must be in **type-compatibility**.
  - Again, the attribute names in the result will be the same as the attribute names in R (or S)

SET DIFFERENCE (EXCEPT)

- Denoted by $-$
- Binary operation
  - The result of $R - S$: a relation that includes all tuples that are in R but not in S.
    - R and S must be in **type-compatibility**.
  - Again, the attribute names in the result will be the same as the attribute names in R (or S)

Example to Illustrate the Result of `UNION`, `INTERSECT`, and `DIFFERENCE`

STUDENT (S)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR (I)

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

$[S \cup I]$

Fn	Ln
Susan	Yao
Ramesh	Shah

$[S \cap I]$

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

$[S - I]$

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

$[I - S]$

`DIFFERENCE` is not commutative.

Some Prosperities of Set Operations

- Both union and intersection are *commutative*.
  - $R \cup S = S \cup R$, and $R \cap S = S \cap R$
- Both union and intersection are *associative* operations.
  - They can be treated as *n*-ary operations applicable to *any* number of relations.
    - $R \cup (S \cup T) = (R \cup S) \cup T$
    - $(R \cap S) \cap T = R \cap (S \cap T)$
- The difference operation is not commutative, as we have seen before. In general,
  - $(R - S) \neq (S - R)$

CARTESIAN (or CROSS) PRODUCT

- Used to combine tuples from two relations in a combinatorial fashion.
- Denoted by $R(A_1, A_2, \ldots, A_n)$ x $S(B_1, B_2, \ldots, B_m)$
  - R and S do **NOT** have to be "type compatible".
- Result: $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$
  - The degree of relation Q: $n + m$ (attributes)
  - The state of Q has one tuple for each combination of tuples—one from R and one from S.
  - The cardinality (number of tuples) of Q: $|R|$ x $|S|$
    - $|R|$: number of tuples in R, $|S|$: number of tuples in S
- Especially useful when a selection is applied after Cartesian product of two relations.
  - But be careful, as mostly this operation is meaningless and expensive.

CARTESIAN PRODUCT: Example

- "Retrieve a list of names of each female employee's dependents."

1: **FEMALE_EMPS** $\leftarrow \sigma_{\text{SEX='F'}}$(EMPLOYEE)

2: **EMPNAMES** $\leftarrow \pi_{\text{Fname, Lname, Ssn}}$ (**FEMALE_EMPS**)

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

CARTESIAN PRODUCT: Example (Cont'd)

3: **EMP_DEPENDENTS** ← **EMPNAMES** **x** DEPENDENT

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

In SQL, CARTESIAN PRODUCT is realized by using the CROSS JOIN option in joined tables that appear in the FROM clause.

```
SELECT  *
FROM    EMPNAMES
        CROSS JOIN
        DEPENDENT;
```

CARTESIAN PRODUCT: Example (Cont'd)

4: ACTUAL_DEPS $\leftarrow \sigma_{\text{Ssn=Essn}}$(EMP_DEPENDENTS)

5: RESULT $\leftarrow \pi_{\text{Fname,Lname,Dependent\_Name}}$(ACTUAL_DEPS)

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	. . .
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	. . .

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

In reality, this is equivalent to:

ACTUAL_DEPS $\leftarrow$ EMPNAMES $\bowtie_{\text{Ssn=Essn}}$ DEPENDENT

BINARY RELATIONAL OPERATIONS: JOIN AND DIVISION

Chapter 8.3

JOIN

- Binary operations; Denoted by $\bowtie$
- Used to combine related tuples from two relations into single "longer" tuples
- VERY important, in a sense that it allows us to process relationships among relations
- Ex) "Retrieve the name of the manager of each department."

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE}$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	. . .	Fname	Minit	Lname	Ssn	. . .
Research	5	333445555	. . .	Franklin	T	Wong	333445555	. . .
Administration	4	987654321	. . .	Jennifer	S	Wallace	987654321	. . .
Headquarters	1	888665555	. . .	James	E	Borg	888665555	. . .

JOIN: Some Properties

- Given two relations, $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_m)$, the general form of a JOIN operation can be expressed by:

$$R \bowtie_{<join\ condition>} S.$$

  - Result: $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$
    - The degree of relation Q: $n + m$ (attributes)
    - The state of Q has one tuple for each combination of tuples—one from R and one from S, but only if they satisfy the join condition:

$$\boldsymbol{r[A_i] = s[B_j].}$$

    - The cardinality (number of tuples) of $Q \leq (|R| \times |S|)$
      - $|R|$: number of tuples in R, $|S|$: number of tuples in S
    - Difference from Cartesian Product: Only related tuples (based on the join condition) will appear in Q.

JOIN: Some Properties (Cont'd)

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

  $$R \bowtie_{\theta} S,$$

  where θ (called *theta*) (in Greek) is the join condition.

- θ can be any general Boolean expression on the attributes of R and S; e.g.,

  $$\theta \Rightarrow (R.A_i < S.B_j \text{ AND } (R.A_k = S.B_l \text{ OR } R.A_p < S.B_q))$$

- Most join conditions involve one or more equality condition, called *equijoin*: "AND"ed together; e.g.,

  $$\theta \Rightarrow (R.A_i = S.B_j \text{ AND } (R.A_k = S.B_l \text{ OR } R.A_p = S.B_q))$$

Variations of JOIN: EQUIJOIN

- EQUIJOIN operation: the most common use of join
- Involves join conditions with **equality comparisons** (=) only
  - So such a join is called an EQUIJOIN.
- In the result of EQUIJOIN, we always have one or more pairs of attributes that have *identical values* in every tuple.
  - But the names of the attributes need not be identical.

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

Variations of JOIN: NATURAL JOIN

- Denoted by *
- Created to remove the 2nd attribute in an EQUIJOIN condition
  - As one of each pair of attributes with identical values is superflous.
- As we studied last time, the two join attributes must have the same name in both relations.
  - If not, you first need to rename them to be consistent before applying natural join. E.g., NATURAL JOIN on PROJECT & DEPARTMENT

$$\text{DEPT} \leftarrow \rho_{(\text{Dname, } \mathbf{Dnum}\text{, Mgr\_ssn, Mgr\_start\_date})}(\text{DEPARTMENT})$$

$$\text{PROJ\_DEPT} \leftarrow \text{PROJECT} * \text{DEPT}$$

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

Join Selectivity

- A property of each join condition
- Defined as the expected size of the join result divided by the maximum size ($|R|$ x $|S|$); expressed as percentage
  - How about join selectivity on the following join?

$$\texttt{DEPARTMENT} \bowtie_{\texttt{Mgr\_ssn=Ssn}} \texttt{EMPLOYEE}$$

  - How about join selectivity on Cartesian Product of two arbitrary relations?

- The higher, the lower in the result size; thus, better in query optimization thanks to reduced I/O

Inner Joins, *n*-way Joins, Implementation

- Inner joins
  - A type of “match-and-combine” operation: all discussed so far
  - Defined formally as a combination of `CROSS PRODUCT` and `SELECTION`.
- *n*-way joins
  - Joins involving multiple tables: e.g., a three-way join:

$$((\texttt{PROJECT} \bowtie_{\texttt{Dnum=Dnumber}} \texttt{DEPARTMENT}) \bowtie_{\texttt{Mgr\_Ssn=Ssn}} \texttt{EMPLOYEE})$$

    - Combines each project with its controlling department tuple into a single tuple, and then
    - Combines that tuple with an employee tuple that is the department manager.
- Implementation in SQL:
  - <*join condition*> in `WHERE`, a nested relation via `IN`, joined tables, …

Complete Set of Relational Operations

- A *complete set*: the set of relational algebra operations, $\{\sigma, \pi, \cup, -, \rho, \times\}$
- Why a complete set?
  - Because any of the other original relational algebra operations can be expressed as a sequence of operations from this set—a combination of these six operations.
- For example,
  - $R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$
  - $R \bowtie_{<condition>} S \equiv \sigma_{<\text{condition}>} (R \times S)$
  - $R * S \equiv \sigma_{<\text{condition}>}\pi(\rho(R) \times \rho(S)))$

DIVISION: Illustration of $T(Y) = R(Z) \div S(X)$

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

÷

S

A
a1
a2
a3

=

T

B
b1
b4

- Let $X = \{A\}$, $Y = \{B\}$, and $Z = \{A, B\}$.

- Tuples (values) with `b1` and `b4` appear in R in combination with all three tuples (`a1`, `a2`, `a3`) in S.

DIVISION (Cont'd)

- Denoted by ÷; useful for a special kind of query in database applications
- Applied to two relations and expressed as $R(Z) \div S(X)$, where
  - The attributes of S are a subset of the attributes of R; that is, $X \subseteq Z$.
- The tuples in the denominator relation S *restrict* the numerator relation R, by selecting those tuples in the result (subset of R) that match all values present in relation S.
  - Think of conditional probability…
- Let $Y = Z - X$ (and hence $Z = X \cup Y$);
  - That is, let Y be the set of attributes of R that are not attributes of S.
- The result of ÷: a relation $T(Y)$ that includes a tuple t, if tuples t_R appear in R with $t_R[Y] = t$ and with $t_R[X] = t_S$ *for every* tuple t_S in S.
  - Means that for a tuple to appear in the result (T) of the ÷ operation, the values in tuple t **MUST** appear in R in combination with every tuple in S.

DIVISION: Example

- "Retrieve the names of employees who work on ***all*** the projects that 'John Smith' works on.

$SMITH \leftarrow \sigma_{\text{Fname='John' AND Lname='Smith'}}(EMPLOYEE)$

$SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$

$SSN\_PNOS \leftarrow \pi_{Essn,Pno}(WORKS\_ON)$

$SSNS(Ssn) \leftarrow (SSN\_PNOS) \div SMITH\_PNOS$

$RESULT \leftarrow \pi_{Fname,Lname}(SSNS * EMPLOYEE)$

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

DIVISION: Properties

- ÷ can be expressed as a sequence of π, x, and $-$.

$T1 \leftarrow \pi_Y(R)$ -- get tuples with some attributes of R

$T2 \leftarrow \pi_Y((S \times T1) - R)$ -- leave some tuples all appearing in S but not in R

$T \leftarrow T1 - T2$ -- leave tuples of $T1$ appearing in R and all existing in S

- Defined for convenience for dealing with queries involving *universal quantification* or the *all* condition.
  - In SQL, no matching statement but implemented by "NOT EXISTS".
    - See slides 13-14 in Week 6.

Summary: Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}),(\text{<join attributes 2>})} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}),(\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$

Summary: Operations of Relational Algebra (Cont'd)

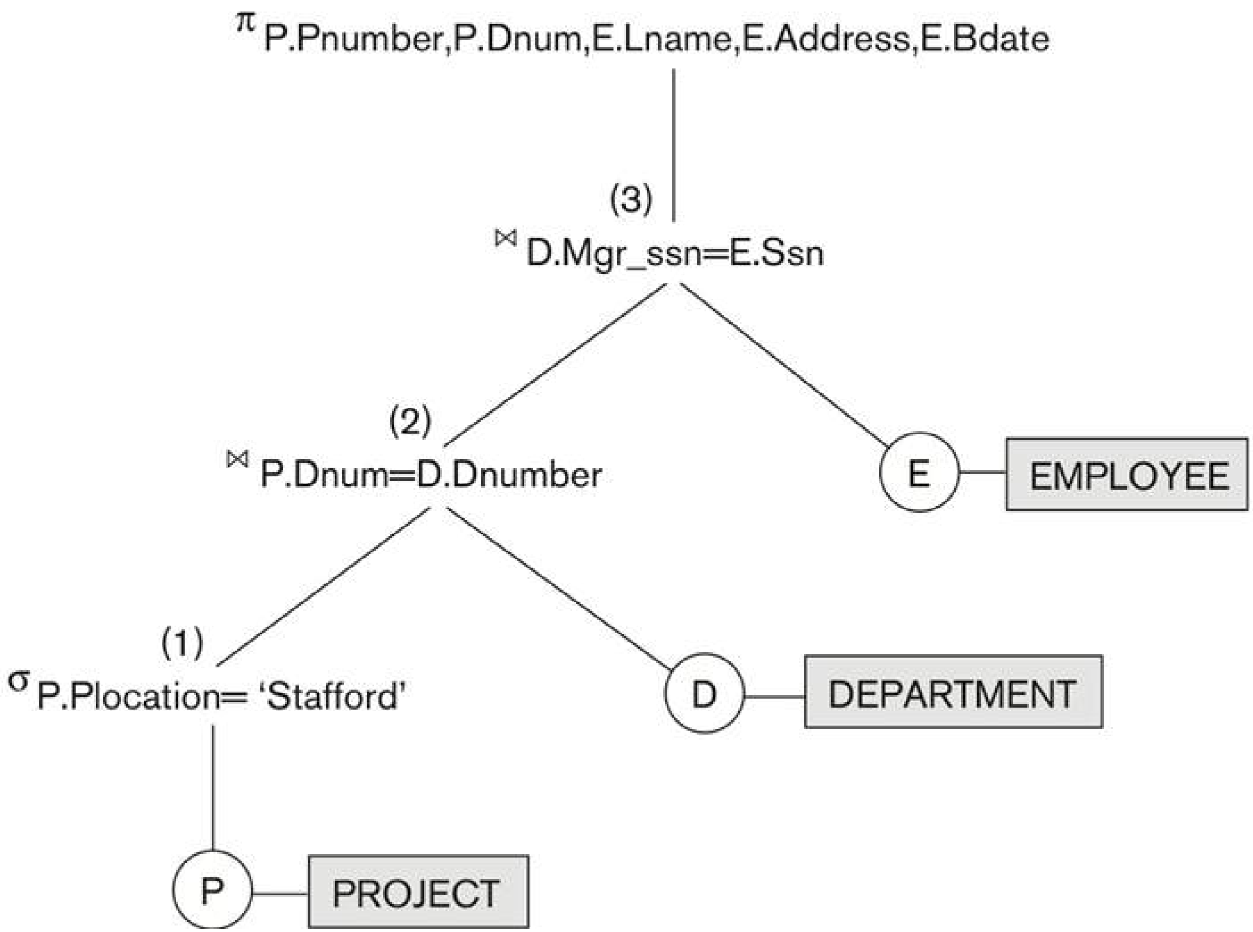
OPERATION	PURPOSE	NOTATION
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Query (Evaluation or Execution) Trees

- An internal data structure to represent a query
  - Corresponds to a relational algebra expression.
- Standard technique for estimating the work involved in
  - query execution
  - generation of intermediate results
  - query optimization: consists of rewriting the query or modifying the query tree into an equivalent tree.
- Non-terminal (non-leaf) tree nodes represent relational algebra operations:
  - selection, projection, join, renaming, division, …
- Terminal (leaf) tree nodes represent base relations (tables).

Example of a Query Tree

$$\pi_{\texttt{Pnumber,Dnum,Lname,Address,Bdate}}(((\sigma_{\texttt{Plocation='Stafford'}}(\texttt{PROJECT})) \bowtie_{\texttt{Dnum=Dnumber}}(\texttt{DEPARTMENT})) \bowtie_{\texttt{Mgr\_Ssn=Ssn}}(\texttt{EMPLOYEE}))$$



ADDITIONAL RELATIONAL OPERATIONS

Chapter 8.4

Aggregate Functions

- Cannot be expressed in the basic relational algebra.
- Concerns collections of values from the database.
  - E.g., Retrieving the average or total salary of all employees or the total number of employee tuples
- Common functions applied to collections of numeric values include:
  - `SUM`, `AVERAGE`, `MAXIMUM`, and `MINIMUM`.
- `COUNT`: used for counting tuples or values

Aggregate Functions (Cont'd)

- Denoted by $\mathcal{F}$ ($\mathfrak{I}$) (pronounced "script F")
- In general form,

  $_{<grouping\ attributes>}\mathcal{F}_{<function\ list>}\ (R)$, where

  - <*grouping attributes*>: a list of attributes of relation R
  - <*function list*>: a list of <*function*> <*attribute*> pairs

- $\mathcal{F}_{\texttt{MAX Salary}}$ (EMPLOYEE) retrieves the maximum Salary value from the EMPLOYEE relation
- $\mathcal{F}_{\texttt{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
- $\mathcal{F}_{\texttt{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary value from the EMPLOYEE relation
- $\mathcal{F}_{\texttt{COUNT Ssn, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary

Using Grouping with Aggregation

- Grouping can be combined with aggregate functions.
  - "For each department, retrieve the `Dno`, count `Ssn`, and average `Salary` and rename …"

$$\rho_{R\,(\texttt{Dno},\texttt{No\_of\_employees},\texttt{Average\_sal})}\left(_{\texttt{Dno}}\,\Im\,_{\texttt{COUNT Ssn, AVERAGE Salary}}(\texttt{EMPLOYEE})\right)$$

R

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

Count_ssn	Average_salary
8	35125

- Duplicates are **NOT** eliminated.
- The result of aggregation is not a scalar number but a relation.
  - This makes the relational algebra a "closed" mathematical system.

Recursive Closure Operation

- Another type of operation that cannot be specified in the basic relational algebra
  - Applied to a recursive relationship
- Ex) "Retrieve all SUPERVISEEs of an `EMPLOYEE` *e* at all levels."
  - All employees *e'* directly supervised by *e*;
  - All employees *e''* directly supervised by *e'*;
  - All employees *e''* directly supervised by *e'''*; and so on.
- It is *difficult* to specify all supervisees at "all" levels.
  - Although it's not hard to specify all employees supervised by *e* at "a specific level".
- Let's find all supervisees by "`James Borg`" at all levels.
  - In practice, you need a looping mechanism unless the max level is known.

Recursive Closure Operation (Cont'd)

$\text{BORG\_SSN} \leftarrow \pi_{\text{Ssn}}(\sigma_{\text{Fname='James' AND Lname='Borg'}}(\text{EMPLOYEE}))$

$\text{SUPERVISION(Ssn1,Ssn2)} \leftarrow \pi_{\text{Ssn, Super\_ssn}}(\text{EMPLOYEE})$

$\text{RESULT1(Ssn1)} \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}}(\text{BORG\_SSN}))$

SUPERVISION

(Borg's Ssn is 888665555)

Ssn1 (Ssn)	Ssn2 (Super_ssn)
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by Borg's subordinates)

RESULT

Ssn
123456789
999887777
666884444
453453453
987987987
333445555
987654321

$\text{RESULT2(Ssn1)} \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}}(\text{RESULT1}))$

$\text{RESULT} \leftarrow \text{RESULT2} \cup \text{RESULT1}$

The OUTER JOIN Operation

- Can be used when we want to keep
  - all the tuples in R, or
  - all those in S, or
  - all those in both relations in the join result.
    - Regardless of whether or not they have matching tuples in the other relation.
- Needed when combining matching rows and not losing tuples with no matching values. Thus, no information loss.
- C.f., In NATURAL JOIN and EQUIJOIN, tuples without a matching (or *related*) tuple are excluded from the join result.
    - Such joins are thus called INNER JOIN.
  - Tuples with NULL in the join attributes are also eliminated.
  - This amounts to loss of information.

The `OUTER JOIN` Operation (Cont'd)

- Left outer join:
  - Keeps every tuple in the first (or left) relation R in $R ⟕ S$.
    - If no matching tuples in S, then the attributes of S in the join result are filled (or "padded") with NULL values.

$$\textbf{TEMP} \leftarrow \pi_{\texttt{Ssn}}(\texttt{EMPLOYEE} ⟕_{\texttt{Ssn=Mgr\_Ssn}} (\texttt{DEPARTMENT}))$$
$$\textbf{RESULT} \leftarrow \pi_{\texttt{Fname,Minit,Lname,Dname}}(\textbf{TEMP})$$

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

The `OUTER JOIN` Operation (Cont’d)

- Right outer join:
  - Keeps every tuple in the second (or right) relation S in $R ⟖ S$.
    - If no matching tuples in R, then the attributes of R in the join result are filled (or “padded”) with `NULL` values.

- Full outer join:
  - Keeps all tuples both in relation R in $R ⟗ S$.
    - When no matching tuples are found, then the attributes of R and S in the join result are filled (or “padded”) with `NULL` values as needed.

EXAMPLES OF QUERIES IN RELATIONAL ALGEBRA

Chapter 8.5

Examples in Procedural Form

- Q1: Retrieve the name and address of all employees who work for the 'Research' department.

$$\textbf{RESEARCH\_DEPT} \leftarrow \sigma_{\texttt{Dname='Research'}}(\text{DEPARTMENT})$$
$$\textbf{RESEARCH\_EMPS} \leftarrow (\textbf{RESEARCH\_DEPT} \bowtie_{\texttt{Dnumber=Dno}}(\text{EMPLOYEE})$$
$$\textbf{RESULT} \leftarrow \pi_{\texttt{Fname, Lname, Address}}(\textbf{RESEARCH\_EMPS})$$

- Q6: Retrieve the names of employees who have no dependents.

$$\textbf{ALL\_EMPS} \leftarrow \pi_{\texttt{Ssn}}(\text{EMPLOYEE})$$
$$\textbf{EMPS\_WITH\_DEPS(SSN)} \leftarrow \pi_{\texttt{Ssn}}(\text{DEPENDENT})$$
$$\textbf{EMPS\_WITHOUT\_DEPS(SSN)} \leftarrow \textbf{ALL\_EMPS-EMPS\_WITH\_DEPS}$$
$$\textbf{RESULT} \leftarrow \pi_{\texttt{Fname, Lname, Address}}(\textbf{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$$

Examples in Single Expressions

- Q1: Retrieve the name and address of all employees who work for the 'Research' department.

$$\pi_{\text{Fname},\text{Lname},\text{Address}}(\sigma_{\text{Dname}='\text{Research}'}(\text{DEPARTMENT} \bowtie_{\text{Dnumber}=\text{Dno}}(\text{EMPLOYEE})))$$

- Q6: Retrieve the names of employees who have no dependents.

$$\pi_{\text{Fname, Lname, Address}}((\pi_{\text{Ssn}}(\text{EMPLOYEE}) - \rho_{\text{Ssn}}(\pi_{\text{Ssn}}(\text{DEPENDENT}))*\text{EMPLOYEE})$$

Examples (Cont'd)

- Q3: Find the names of employees who work on all the projects controlled by department number 5.

$$DEPT5\_PROJS \leftarrow \rho_{(Pno)}\ \pi_{Pnumber}(\sigma_{Dnum=5}\ (PROJECT))$$

$$EMP\_PROJ \leftarrow \rho_{(Ssn,Pno)}\ \pi_{Essn,Pno}\ (WORKS\_ON)$$

$$RESULT\_EMP\_SSNS \leftarrow EMP\_PROJ \div DEPT5\_PROJS$$

$$RESULT \leftarrow \pi_{Fname,\ Lname}(RESULT\_EMP\_SSNS * EMPLOYEE)$$

```
SELECT  E.Fname, E.Lname
FROM    EMPLOYEE E
WHERE   NOT EXISTS (SELECT    *
                    FROM      WORKS_ON B
                    WHERE     (B.Pno IN (SELECT P.Pnumber
                                         FROM   PROJECT P
                                         WHERE  P.Dnum = 5)
                              AND
                              NOT EXISTS (SELECT *
                                          FROM WORKS_ON C
                                          WHERE C.Essn = E.ssn
                                            AND C.Pno  = B.Pno)));
```

RELATIONAL CALCULUS - THE **TUPLE** RELATIONAL CALCULUS

Chapter 8.6

Relational Calculus

- Another formal language for the relational model
- Creates a new relation, specified in terms of variables ranging
  - over **ROWS** of the stored relations: called *tuple calculus*
  - over **COLUMNS** of the stored relations: called *domain calculus*
- A calculus expression doesn't specify *how* to retrieve the query result.
  - There is **NO** order of relational operations.
- It specifies only *what* information the result should contain.
- What about relational algebra?

Relational Calculus (Cont'd)

- Relational calculus is considered to be a **nonprocedural** or **declarative** language.
  - Provides the formal, mathematical basis on SQL.

- This differs from "relational algebra," where we write *a sequence of operations* to specify a retrieval request.
  - Hence, relational algebra can be considered as a **procedural** way of stating a query.
    - In spite of a possible single-line expression, a certain order among the operations is always explicitly specified.
    - The order of the operators in the expression may *significantly* affect the performance of executing a relational query.

Relational Calculus (Cont'd)

- Note that relational algebra and relational calculus have the identical **expressive power** as query language.
  - This led to the concept of a **relationally complete** language.
    - "A relational language L is considered relationally complete if we can express in L any query that can be expressed in relational calculus.
  - Most relational query languages are relationally complete.
    - SQL has *more expressive power* than relational calculus or relational algebra. Why?
- Why is relational calculus important? Two reasons.
  - First, it has a firm mathematical logic.
  - Second, the SQL for RDBMSs has its basic foundation in the tuple relational calculus.

Tuple Relational Calculus

- Based on specifying a number of "tuple variables"
- Each tuple usually "ranges over" a particular relation.
  - Means that the variable may take as its value "any" individual tuple from relation.
- A simple tuple relational calculus query is of the form:

$$\{t \mid \mathbf{COND}(t)\},$$

  where t is a tuple variable and **COND**(t) is a conditional expression involving t.
- The result of such a query: the set of all tuples satisfying **COND**(t).

Tuple Relational Calculus (Cont'd)

- "Find all employees whose salary is above $50,000."

$$\{t.\texttt{Fname},\ t.\texttt{Lname} \mid \texttt{EMPLOEE}(t)\ \text{AND}\ t.\texttt{Salary} > 50000\}$$

  - $\texttt{EMPLOEE}(t)$ specifies that the **range relation** of tuple variable t is EMPLOYEE.
    - If we don't specify a range relation, then the variable t will range over all possible tuples "in the universe" (as it's not restricted to any one relation).

  - The first name ($t.\texttt{Fname}$) and last name $(t.\texttt{Fname})$ (= $\pi_{\texttt{Fname,Lname}}$) of each EMPLOEE tuple t that satisfies the condition, $t.\texttt{Salary} > 50000$ (= $\sigma_{\text{SALARY} > 50000}$), will be included in the query result.

Tuple Relational Calculus (Cont'd)

- A general expression of the tuple relational calculus:

$$\{t_1.A_i,\ t_2.A_k,\ \ldots,\ t_n.A_m \mid \mathrm{COND}(t_1, t_2, \ldots\ t_n, t_{n+1}, \ldots, t_m)\}$$

  - Where $t_1, \ldots, t_{n+m}$: tuple variables,
    each A_i: an attribute of the relation on which t_i ranges,
    COND: a condition (or formula) of the tuple relational calculus.
  - A formula is made up of predicate calculus atoms, one of the following:
    - $R(t)$: relation R's tuple variable t; `TRUE` if t exists in R, and `FALSE`, otherwise.
    - $t_i.A$ **op** $t_j.B$: a comparison expression; **op**: one of $\{=, <, \leq, >, \geq\}$; t_i and t_j: tuple variables; A and B: an attribute of the relation on which t_i and t_j range respectively.
    - $t_i.A$ **op** c: another comparison expression; c: a constant value

Tuple Relational Calculus (Cont'd)

- A formula (Boolean conditions) (Cont'd)
  - Made up of one or more atoms connected via the logical operators (`AND`, `OR`, and `NOT`)
  - Defined recursively be Rules 1 and 2:
    - Rule 1: Every atom is a formula.
    - Rule 2: If F_1 and F_2 are formulas, then so are (F_1 AND F_2), (F_1 OR F_2), NOT(F_1), and NOT(F_2).
    - Rules 3 and 4 will be shown in the next slide.
  - The truth values (similar to general truth values on Boolean values):
    - (F_1 AND F_2): `TRUE` if both are `TRUE`; otherwise, `FALSE`.
    - (F_1 OR F_2): `FALSE` if both are `FALSE`; otherwise, `TRUE`.
    - NOT($F_{1(\text{or } 2)}$): `TRUE` if $F_{1(2)}$ is `FALSE`; `FALSE` if $F_{1(2)}$ is `TRUE`.

The Existential and Universal Quantifiers
(존재 정량자 & 전칭정량자)

- Two special symbols
  - ∃: called an *existential quantifier*, ∀: called an *universal quantifier*
- Concept related to the symbols: ***bound*** or ***free***
  - A tuple variable is bound if quantified (한정되면); that is,
    - If the variable appears in an $\forall\ t(F)$ or $(\exists\ t)$ clause, it's bound; otherwise, free.
      - F_1: d.`Dname = 'Research'` // d: free,
      - F_2: $(\exists\ t)$ d.`Dnumber = ` t.`Dno` // d: free, t: bound
      - F_3: $(\forall\ d)$ d.`Mgr_ssn = '333445555'` // d: bound
- Rules 3 and 4: If F is a formula, then so are $(\exists\ t)(F)$ and $\forall\ t(F)$, where t is a tuple variable.
  - Rule 3: $(\exists\ t)(F)$ is `TRUE` if it evaluates to `TRUE` for *some* (*at least one*) tuple assigned to free occurrences of t in F; otherwise, `FALSE`.
  - Rule 4: $(\forall\ t)(F)$ is `TRUE` if it evaluates to `TRUE` for *every* tuple (in the universe) assigned to free occurrences of t in F; otherwise, `FALSE`.

The Existential and Universal Quantifiers (Cont'd)

- $\exists$: called the *existential quantifier*
  - Why? Because ANY tuple that exists in "the universe of" tuples may make F $\texttt{TRUE}$ to make $(\exists\ t)(F)$ $\texttt{TRUE}$.

- $\forall$: called the *universal* (or "for all") *quantifier*
  - Why? Because EVERY tuple in "the universe of" tuples must make F $\texttt{TRUE}$ to make $(\forall\ t)(F)$ $\texttt{TRUE}$.

Sample Query Using ∃

- Q1: "List the name and address of all employees who work for the 'Research' department."

{*t*.Fname, *t*.Lname, *t*.Address | EMPLOYEE(*t*) AND (∃ *d*) (DEPARTMENT(*d*) AND *d*.Dname='Research' AND *d*.Dnumber=*t*.Dno)}

  - The only *free tuple variables* in a relational calculus expression should be those that appear to the left of the bar (|).
  - If a tuple *t* satisfies the conditions specified in the query, the attributes, Fname, Lname, Address are retrieved for each such tuple *t*.
    - EMPLOYEE(*t*) and DEPARTMENT(*d*) specify the range relations for *t* and *d*.
    - *d*.Dname='Research': selection condition, similar to σ in relational algebra
    - *d*.Dnumber=*t*.Dno: join condition, similar to an inner join in relational algebra

Sample Query Using ∀

- Q1: "List the name of employees who work on all the projects controlled by department number 5."

{e.Fname, e.Lname | EMPLOYEE(e) AND
((∀x)(NOT(PROJECT(x)) OR NOT(x.Dnum=5) OR ((∃w)(WORKS_ON(w) AND w.Essn = e.Ssn AND x.Pnumber = w.Pno))))}

{e.Fname, e.Lname | EMPLOYEE(e) AND **F'**}

F': ((∀x)(NOT(PROJECT(x)) OR **F_1**)) -- F_1 must be TRUE if F' is TRUE.

F_1: NOT(x.Dnum=5) OR **F_2** -- F_2 must be TRUE if F_1 is TRUE.

F_2: ((∃w)(WORKS_ON(w) AND w.Essn = e.Ssn AND x.Pnumber = w.Pno))

```
SELECT  E.Fname, E.Lname
FROM    EMPLOYEE E
WHERE   NOT EXISTS (SELECT     *
                    FROM       WORKS_ON B
                    WHERE      (B.Pno IN (SELECT P.Pnumber
                                          FROM   PROJECT P
                                          WHERE  P.Dnum = 5)
                               AND
                               NOT EXISTS (SELECT *
                                           FROM WORKS_ON C
                                           WHERE C.Essn = E.ssn
                                            AND C.Pno  = B.Pno)));
```

Sample Query Using ∀ (Cont'd)

- Q3: "List the name of employees who work on all the projects controlled by department number 5."

$\{e.\texttt{Fname}, e.\texttt{Lname} \mid \texttt{EMPLOYEE}(e)$ AND $\boldsymbol{F'}\}$

F': $((\forall x)(\text{NOT}(\texttt{PROJECT}(x))$ OR $\boldsymbol{F_1}))$ -- F_1 must be $\texttt{TRUE}$ if F' is $\texttt{TRUE}$.

F_1: NOT$(x.\texttt{Dnum}=5)$ OR $\boldsymbol{F_2}$ -- F_2 must be $\texttt{TRUE}$ if F_1 is $\texttt{TRUE}$.

F_2: $((\exists w)(\texttt{WORKS\_ON}(w)$ AND $w.\texttt{Essn} = e.\texttt{Ssn}$ AND $x.\texttt{Pnumber} = w.\texttt{Pno}))$

- For F' (=$((\forall x)\ F)$) to be $\texttt{TRUE}$, F must be $\texttt{TRUE}$ for all tuples (in the universe) that can be assigned to x.

- We're only interested only in F being $\texttt{TRUE}$ for all tuples of $\texttt{PROJECT}$ controlled by department 5. Thus, (NOT$(\texttt{PROJECT}(x))$ OR F_1): that is, for every tuple in $\texttt{PROJECT}$, F_1 must be $\texttt{TRUE}$ if F' is to be $\texttt{TRUE}$.

- We don't consider tuples in $\texttt{PROJECT}$ not controlled by department no. 5; that is, (NOT$(x.\texttt{Dnum}=5)$ OR F_2); for a tuple x in $\texttt{PROJECT}$, its $\texttt{Dnum}\neq 5$ or it must satisfy F_2.

- If F_2 is true, then a selected $\texttt{EMPLOYEE}$ tuple is held, such that the employee works on every $\texttt{PROJECT}$ tuple *that has not been eliminated yet*; such employee tuples are selected by the query.

Languages Based on Tuple Relational Calculus

- SQL: based on tuple calculus.
  - Uses the basic block structure to express the queries in tuple calculus

```
SELECT <list of attributes>
FROM   <list of relations>
WHERE  <conditions>
```

  - `SELECT`: mentions the attributes being projected
  - `FROM`: mentions the relations needed in the query
  - `WHERE`: mentions the selection as well as the join conditions

[Appendix] Languages Based on Tuple Relational Calculus (Cont'd)

- QUEL: another based on tuple calculus.
  - Actually uses the range variables as in tuple calculus
  - Syntax:
    - `RANGE OF` <*variable name*> `IS` <*relation name*>
    - `RETRIEVE` <*list of attributes from range variables*>
    - `WHERE` <*conditions*>
  - Proposed in the relational DBMS INGRES
    - INGRES: a research project at UC Berkeley, starting in the early 1970s and ending in 1985
      - Yielded a number of commercial database applications: Sybase, Microsoft SQL Server, NonStop SQL and a number of others [from Wikipedia]
      - Postgres (**Post** Ing**res**) later evolved into PostgreSQL.

RELATIONAL CALCULUS - THE DOMAIN RELATIONAL CALCULUS

Chapter 8.7

Domain Relational Calculus

- Another type of relational calculus
- Proposed after the Query-By-Example (QBE) language
  - Developed by IBM T. J. Watson Research Center, Yorktown Heights, NY
- Variables range over *single values from domains of attributes*.
  - Unlike tuple relational calculus
- An expression of domain relational calculus:

$$\{x_1, x_2, \dots, x_n \mid \mathrm{COND}\,(x_1, x_2, \dots x_n, x_{n+1}, \dots, x_m)\},$$

where $x_1, x_2, \dots x_n, x_{n+1}, \dots, x_{n+m}$: domain variables, and
COND: a condition or formula of the domain relational calculus.

Domain Relational Calculus: Example

- Q0: "List the birth date and address of the employee whose name is '`John B. Smith`'."

$\{u, v \mid (\exists\, q)\ (\exists\, r)\ (\exists\, s)\ (\exists\, t)\ (\exists\, w)\ (\exists\, x)\ (\exists\, y)\ (\exists\, z)\ (\texttt{EMPLOYEE}(qrstuvwxyz)$
AND $q = \texttt{'John'}$ AND $r = \texttt{'B'}$ AND $s = \texttt{'Smith'})\}$