Chapter 6: Formal Relational Query Languages

Database System Concepts, 6th Ed.

- Chapter 1: Introduction
- **Part 1: Relational databases**
  - Chapter 2: Introduction to the Relational Model
  - Chapter 3: Introduction to SQL
  - Chapter 4: Intermediate SQL
  - Chapter 5: Advanced SQL
  - Chapter 6: Formal Relational Query Languages
- **Part 2: Database Design**
  - Chapter 7: Database Design: The E-R Approach
  - Chapter 8: Relational Database Design
  - Chapter 9: Application Design
- **Part 3: Data storage and querying**
  - Chapter 10: Storage and File Structure
  - Chapter 11: Indexing and Hashing
  - Chapter 12: Query Processing
  - Chapter 13: Query Optimization
- **Part 4: Transaction management**
  - Chapter 14: Transactions
  - Chapter 15: Concurrency control
  - Chapter 16: Recovery System
- **Part 5: System Architecture**
  - Chapter 17: Database System Architectures
  - Chapter 18: Parallel Databases
  - Chapter 19: Distributed Databases
- **Part 6: Data Warehousing, Mining, and IR**
  - Chapter 20: Data Mining
  - Chapter 21: Information Retrieval
- **Part 7: Specialty Databases**
  - Chapter 22: Object-Based Databases
  - Chapter 23: XML
- **Part 8: Advanced Topics**
  - Chapter 24: Advanced Application Development
  - Chapter 25: Advanced Data Types
  - Chapter 26: Advanced Transaction Processing
- **Part 9: Case studies**
  - Chapter 27: PostgreSQL
  - Chapter 28: Oracle
  - Chapter 29: IBM DB2 Universal Database
  - Chapter 30: Microsoft SQL Server
- **Online Appendices**
  - Appendix A: Detailed University Schema
  - Appendix B: Advanced Relational Database Model
  - Appendix C: Other Relational Query Languages
  - Appendix D: Network Model
  - Appendix E: Hierarchical Model

Chapter 6: Formal Relational Query Languages

- 6.1 Relational Algebra
- 6.2 Tuple Relational Calculus
- 6.3 Domain Relational Calculus

Formal Theory behind SQL!!!

Relational Algebra

- Procedural language (Language Specifying What to Retrieve, not How to Retrieve)
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.

Select Operation σ – Example

- 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

Select Operation σ

- Notation: $\sigma_p(r)$ (where *p* is called the **selection predicate)**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$ // tuple relational calculus

Where *p* is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

Each **term** is one of: (<attribute>) *op* (<attribute> or <constant>)

where *op* is one of: =, $\neq$, >, $\geq$. <. $\leq$

- Example of selection:

$$\sigma_{dept\_name=\text{“}Physics\text{”}}(instructor)$$

Project Operation $\prod$ – Example

- Relation *r*:

A	*B*	*C*
α	10	1
α	20	1
β	30	1
β	40	2

- $\prod_{A,C}(r)$

A	*C*
α	1
α	1
β	1
β	2

=

A	*C*
α	1
β	1
β	2

Project Operation Π

- Notation: $\Pi_{A_1, A_2, \ldots, A_k}(r)$

  where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\Pi_{ID,\ name,\ salary}(instructor)$$

Figure 6.01: The Instructor relation

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 6.02: The result of applying a selection predicate dept_name = "Physics"

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Figure 6.03: The result of applying projection with ID, name, salary to Instructor relation

ID	name	salary
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Figure 6.07: The Teaches relation

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Figure 6.08: The result of instructor X teaches

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2009
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2010
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

Figure 6.09: The Result of applying a selection predicate dept_name = "Physics" to instructor X teaches

inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
22222	Einstein	Physics	95000	10101	CS-437	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
22222	Einstein	Physics	95000	32343	HIS-351	1	Spring	2010
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
33456	Gold	Physics	87000	10101	CS-437	1	Fall	2009
33456	Gold	Physics	87000	10101	CS-315	1	Spring	2010
33456	Gold	Physics	87000	12121	FIN-201	1	Spring	2010
33456	Gold	Physics	87000	15151	MU-199	1	Spring	2010
33456	Gold	Physics	87000	22222	PHY-101	1	Fall	2009
33456	Gold	Physics	87000	32343	HIS-351	1	Spring	2010
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

Figure 6.10:

name	*course_id*
Einstein	PHY-101

Union Operation ∪ – Example

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Union Operation ∪

- Notation: $r \cup s$
- Defined as:

  $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$ // tuple relatioanal calculus

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Set difference (–) of two relations

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* – *s*:

A	B
α	1
β	1

Set Difference (–) Operation

- Notation $r - s$
- Defined as:

  $r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$ // tuple relational calculus

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible

- Eg: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Figure 6.04: The section relation

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

Figure 6.05: Courses offered in either Fall 2009, Spring 2010 or both semesters

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Figure 6.06: Courses offered in either Fall 2009, but not in Spring 2010 semesters

course_id
CS-347
PHY-101

Figure 6.13: Courses offered in both the Fall 2009 and Spring 2010 semesters

course_id
CS-101

Cartesian-Product (X) Operation – Example

- Relations *r, s*:

A	*B*
α	1
β	2

r

C	*D*	*E*
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	*B*	*C*	*D*	*E*
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$

A	*B*	*C*	*D*	*E*
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Cartesian-Product Operation

- Notation r **x** s
- Defined as:

  $r \times s = \{t\ q \mid t \in r$ **and** $q \in s\}$ // tuple relational calculus

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_x(E)$$

returns the expression *E* under the name *X*

- If a relational-algebra expression *E* has arity *n*, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

Example Query with Renaming ρ

- Find the largest salary in the university
  - Step 1: find instructor salaries that are less than some other instructor salary (i.e. not maximum)
    - using a copy of *instructor* under a new name *d*
    - $\Pi_{instructor.salary} (\sigma_{instructor.salary < d.salary} (instructor \times \rho_d (instructor)))$
    -
  - Step 2: Find the largest salary
    - $\Pi_{salary} (instructor) -$
      $\Pi_{instructor.salary} (\sigma_{instructor.salary < d,salary} (instructor \times \rho_d (instructor)))$

Figure 6.01: The Instructor relation

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 6.11: 누군가의 salary보다는 적은 salary

salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

Figure 6.12: The highest salary

salary
95000

Example Queries with selection commutativity

- Find the names of all instructors in the Physics department, along with the *course_id* of all courses they have taught

  - Query 1

$$\Pi_{instructor.ID,course\_id}(\sigma_{dept\_name=\text{“Physics”}}(\sigma_{instructor.ID=teaches.ID}(instructor \times teaches)))$$

  - Query 2

$$\Pi_{instructor.ID,course\_id}(\sigma_{instructor.ID=teaches.ID}(\sigma_{dept\_name=\text{“Physics”}}(instructor) \times teaches))$$

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\Pi_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Assignment
- Outer join

Set-Intersection (∩) Operation

- Notation: $r \cap s$
- Defined as: $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$ // tuple relational calculus
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

- Relation r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Natural-Join ($\bowtie$) Operation

- Notation: r $\bowtie$ s
- Let *r* and *s* be relations on schemas *R* and *S* respectively.
  Then, r $\bowtie$ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$
  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

    $$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$$

Natural Join Example

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural Join and Theta Join

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach

$$\Pi_{name,\ title}\left(\sigma_{dept\_name=\text{“Comp. Sci.”}}\left(instructor \bowtie teaches \bowtie course\right)\right)$$

- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$ is equivalent to $instructor \bowtie (teaches \bowtie course)$
- Natural join is commutative
  - $instruct \bowtie teaches$ is equivalent to $teaches \bowtie instructor$

- The **theta join** operation $r \bowtie_{\theta} s$ is defined as
  - $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$
  - Theta θ is a selection predicate

Figure 6.16

name	*title*
Brandt	Game Design
Brandt	Image Processing
Katz	Image Processing
Katz	Intro. to Computer Science
Srinivasan	Intro. to Computer Science
Srinivasan	Robotics
Srinivasan	Database System Concepts

Figure 6.14: Natural Join of the instructor relation and the teaches relation

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 6.15: Result of projection with name, course_id on Figure 6.14

name	*course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

Assignment (←) Operation

- The assignment operation (←) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.

- *Example:* R ⋈ S

$$temp1 \leftarrow R \times S$$

$$temp2 \leftarrow \sigma_{R.A1 = S.A1 \wedge R.A2 = S.A2 \ \ldots \ \wedge R.An = S.An}(temp1))$$

$$result = \prod_{R\ U\ S} (temp2)$$

  - The result to the right of the ← is assigned to the relation variable on the left of the ←.

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

- Outer join can be expressed using basic relational operations
  - e.g. r ⟕ s can be written as

    $(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$
- Outer Join
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Outer Join – Example [1/2]

- Relation *instructor*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches*

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

- Join : *instructor* ⋈ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join: *instructor* ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example [2/2]

- Relation *instructor*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches*

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

- Right Outer Join: *instructor* ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join: *instructor* ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Figure 6.17: Result of instructor left_outer_jon teaches

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
33456	Gold	Physics	87000	*null*	*null*	*null*	*null*
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
58583	Califieri	History	62000	*null*	*null*	*null*	*null*
76543	Singh	Finance	80000	*null*	*null*	*null*	*null*
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 6.18: Result of teaches right_outer_join instructor

ID	*course_id*	*sec_id*	*semester*	*year*	*name*	*dept_name*	*salary*
10101	CS-101	1	Fall	2009	Srinivasan	Comp. Sci.	65000
10101	CS-315	1	Spring	2010	Srinivasan	Comp. Sci.	65000
10101	CS-347	1	Fall	2009	Srinivasan	Comp. Sci.	65000
12121	FIN-201	1	Spring	2010	Wu	Finance	90000
15151	MU-199	1	Spring	2010	Mozart	Music	40000
22222	PHY-101	1	Fall	2009	Einstein	Physics	95000
32343	HIS-351	1	Spring	2010	El Said	History	60000
33456	*null*	*null*	*null*	*null*	Gold	Physics	87000
45565	CS-101	1	Spring	2010	Katz	Comp. Sci.	75000
45565	CS-319	1	Spring	2010	Katz	Comp. Sci.	75000
58583	*null*	*null*	*null*	*null*	Califieri	History	62000
76543	*null*	*null*	*null*	*null*	Singh	Finance	80000
76766	BIO-101	1	Summer	2009	Crick	Biology	72000
76766	BIO-301	1	Summer	2010	Crick	Biology	72000
83821	CS-190	1	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-190	2	Spring	2009	Brandt	Comp. Sci.	92000
83821	CS-319	2	Spring	2010	Brandt	Comp. Sci.	92000
98345	EE-181	1	Spring	2009	Kim	Elec. Eng.	80000

Null Values

- It is possible for tuples to have a null value (denoted by *null*) for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Evaluating Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*,

    then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
    (*false* **and** *unknown*) = *false,*
    (*unknown* **and** *unknown*) = *unknown*
  - NOT: (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

Division Operation [1/5]

- Notation: $r \div s$
- Suited to queries that include the phrase “for all”.
- Let *r* and *s* be relations on schemas *R* and *S* respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of r ÷ s is a relation on schema $R - S = (A_1, \ldots, A_m)$

  $$r \div s = \{ t \mid t \in \prod_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$$ // tuple relational calculus

  Where *tu* means the concatenation of tuples *t* and *u* to produce a single tuple

Division Operation – Example [2/5]

- Relations *r, s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
$\in$	6
$\in$	1
β	2

r

B
1
2

s

- *r* ÷ *s*:

A
α
β

Another Division Example [3/5]

- Relations *r, s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	a	*1*
α	a	γ	b	*1*
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	a	*1*
γ	a	γ	b	*1*
γ	a	β	b	*1*

r

D	*E*
a	*1*
b	*1*

s

- *r* ÷ *s*:

A	*B*	*C*
α	a	γ
γ	a	γ

Division Operation [4/5]

- Property
  - Let $q = r \div s$
  - Then q is the largest relation satisfying $q \times s \subseteq r$
- Definition (in terms of the basic algebra operation)

  Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

  $$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

  To see why
  - $\Pi_{R-S,S}(r)$ simply reorders attributes of r
  - $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

Division (÷) Operator [5/5]

- Given relations r(R) and s(S), such that $S \subset R$, $r \div s$ is the largest relation t(R-S) such that

  $t \times s \subseteq r$

- E.g. let $r(ID, course\_id) = \prod_{ID,\ course\_id}(takes)$ and

  $s(course\_id) = \prod_{course\_id}(\sigma_{dept\_name=\text{“Biology”}}(course))$

  then $r \div s$ gives us students who have taken all courses in the Biology department

- Can write $r \div s$ as

  $temp1 \leftarrow \prod_{R-S}(r)$

  $temp2 \leftarrow \prod_{R-S}((temp1 \times s) - \prod_{R-S,S}(r))$

  $result = temp1 - temp2$

  - The result to the right of the ← is assigned to the relation variable on the left of the ←.
  - May use variable in subsequent expressions.

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- E is any relational-algebra expression
- Each of $F_1, F_2, \ldots, F_n$ are arithmetic expressions involving constants and attributes in the schema of E.
- Given relation *instructor(ID, name, dept_name,* salary) where salary is annual salary, get the same information but with monthly salary

$$\Pi_{ID,\ name,\ dept\_name,\ salary/12}\ (instructor)$$

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1,G_2,\ldots,G_n}\mathcal{G}_{F_1(A_1),F_2(A_2,\ldots,F_n(A_n)}(E)$$

  E is any relational-algebra expression

  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

- Note: Some books/articles use γ instead of $\mathcal{G}$ (Calligraphic G)

Aggregate Operation – Example

- Relation *r*:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathcal{G}_{\text{sum(c)}}(r)$

sum(*c*)
27

- Find the average salary in each department: (with renaming attribute)

$$dept\_name\ \mathcal{G}_{\textbf{avg}(salary)\ \textbf{as}\ avg\_sal}(instructor)$$

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

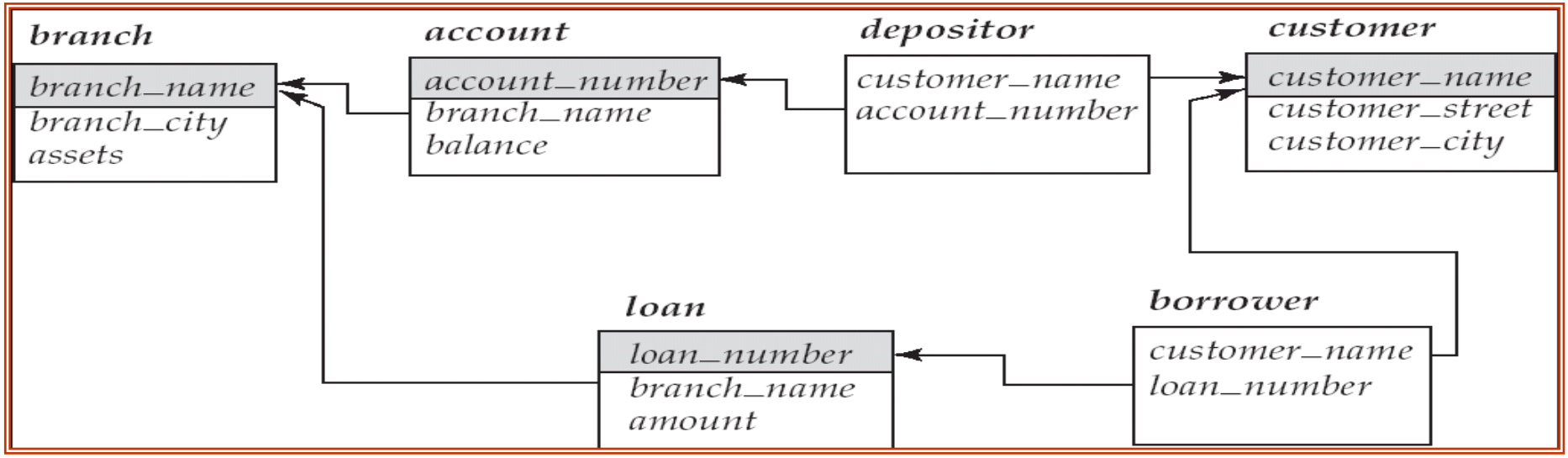
dept_name	*avg_sal*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Relational Algebra Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations can be expressed using the assignment operator

- SQL의 deletion, insertion, update를 relational algebra차원에서 이해하기 위해서…….

Bank Database [1]



The Account Relation

account_number	branch_name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

The *branch* relation

branch_name	branch_city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

Bank Database [2]

The *customer* Relation

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

The *depositor* Relation

customer_name	*account_number*
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

The *loan* relation

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

The *borrower* relation

customer_name	*loan_number*
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Deletion in Relational Algebra

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.

- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

Deletion Examples in Relational Algebra

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\_name = \text{“Perryridge”}}(account)$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$$

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch\_city = \text{“Needham”}}(account \bowtie branch)$$
$$r_2 \leftarrow \Pi_{account\_number,\ branch\_name,\ balance}(r_1)$$
$$r_3 \leftarrow \Pi_{customer\_name,\ account\_number}(r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

Insertion in Relational Algebra

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- In relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

Insertion Examples in Relational Algebra

- Insert information in the database specifying that HJKIM is a instructor for Comp. Sci. and his ID is 12345 and his salary is $40000.

$$account \leftarrow account \cup \{(\text{“12345”}, \text{“HJKIM”}, \text{“Comp. Sci”}, 40000)\}$$

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch\_name = \text{“Perryridge”}}(borrower \bowtie loan))$$

$$account \leftarrow account \cup \prod_{loan\_number,\ branch\_name,\ 200}(r_1)$$

$$depositor \leftarrow depositor \cup \prod_{customer\_name,\ loan\_number}(r_1)$$

Updating in Relational Algebra

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l} (r)$$

- Each F_i is either
  - the I^{th} attribute of r, if the I^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

- Update salary by increasing all balances by 5 percent.

  $Instructor \leftarrow \prod_{\text{salary} * 1.05} (instructor)$

Multi-Set Concepts

- Pure Relational Algebra ➔ No Multi-set
- SQL ➔ Multi-set
  - Multiple copies of a tuple 이 input될수도 있고
  - Projection 형태의 query 결과는 Multi-set형태의 tuple을 가질수 있고….

- Therefore, we need multiset relational algebra!

- Multiset relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature

Multiset Relational Algebra

- Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are *m* copies of *t1* in *r*, and *n* copies of *t2* in *s*, there are *m* x *n* copies of *t1.t2* in *r* x *s*
  - Other operators similarly defined
    - E.g. union: *m + n copies,*

      intersection: min(*m, n*) copies

      difference: min(0, *m* – *n*) copies

SQL and Multiset Relational Algebra

- **select** *A1, A2, .. An*
  from *r1, r2, …, rm*
  where P

  is equivalent to the following expression in multiset relational algebra

  $$\Pi_{A1, .., An} (\sigma_{P} (r1 \times r2 \times .. \times rm))$$

- More generally, the non-aggregated attributes in the **select** clause may be a subset of the **group by** attributes, in which case the equivalence is as follows:

  select *A1,* **sum***(A3)*
  from *r1, r2, …, rm*
  where P
  group by *A1, A2*

  is equivalent to the following expression in multiset relational algebra

  $$\Pi_{A1,sumA3} ({}_{A1,A2}\mathcal{G}_{\textbf{sum}(A3)\ \textbf{as}\ sumA3} (\sigma_{P} (r1 \times r2 \times .. \times rm)))$$

Chapter 6: Formal Relational Query Languages

- 6.1 Relational Algebra
- 6.2 Tuple Relational Calculus
- 6.3 Domain Relational Calculus

First-Order Logic

참고자료

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- has a rich structure of decidable fragments
- has a rich model and proof theory

- First-order logic is also called
  - Predicate logic
  - First-order Predicate Calculus

참고자료

Propositional Logic

- ***Propositional logic*** is a mathematical system for reasoning about propositions and how they relate to one another.
- A ***propositional variable*** is a variable that is either true or false.
- The ***propositional connectives*** are
  - Negation: $\neg p$
  - Conjunction: $p \wedge q$
  - Disjunction: $p \vee q$
  - Implication: $p \rightarrow q$
  - Biconditional: $p \leftrightarrow q$
  - True: $\top$
  - False: $\bot$

Expression in Propositional Logic

참고자료

- Some Sample Propositions
  - *a*: I will get up early this morning
  - *b*: There is a lunar eclipse this morning
  - *c*: There are no clouds in the sky this morning
  - *d*: I will see the lunar eclipse

"If I get up early this morning,
but it's cloudy outside,
I won't see the lunar eclipse.

$$a \wedge \neg c \rightarrow \neg d$$

참고자료

What is First Order Logic?

First-order logic is a logical system for reasoning about properties of objects.

- Augments the logical connectives from propositional logic with
  - *predicates* that describe properties of objects, and
  - *functions* that map objects to one another,
  - *quantifiers* that allow us to reason about multiple objects simultaneously.

참고자료

Expressions in First Order Logic

- Example: "Everyone loves someone else."

$$\forall p.\ (Person(p) \rightarrow \exists q.\ (Person(q) \wedge p \neq q \wedge Loves(p, q)))$$

For every person, — there is some person — who isn't them — That they love.

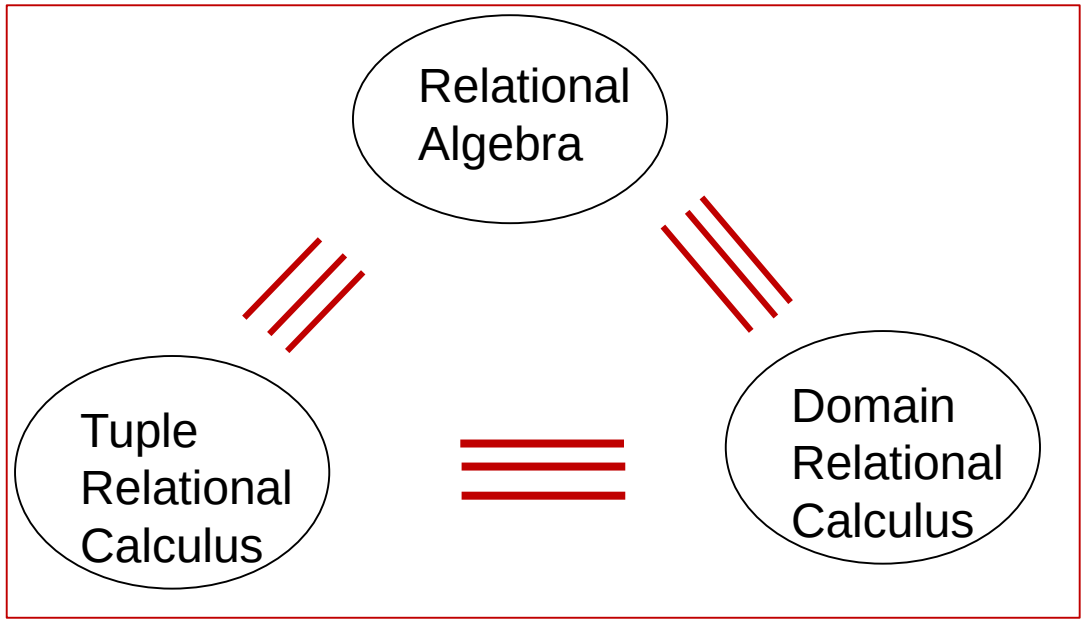
- Example: "There is someone everyone else loves."

$$\exists p.\ (Person(p) \wedge \forall q.\ (Person(q) \wedge p \neq q \rightarrow Loves(q, p)))$$

There is some person — who everyone — who isn't them — loves

Relational Calculus

- By E.F. Codd, 1970
- 2 Types
  - Tuple Relational Calculus (TRC)
  - Domain Relational Calculus (DRC)
- ~~ First Order Logic without Functions
  - *"function"을 고려 하지 않는 First Order Logic의 subset*



Tuple Relational Calculus

- Relation algebra was procedural language (specifying how to retrieve)
- TRC is a nonprocedural (declarative) query language (specifying what to retrieve)
- TRC is based on predicate calculus (= First Order Logic)
  - Without Functions

- Each TRC query is of the form

$$\{t \mid P(t)\}$$

- It is the set of all tuples t such that predicate P is true for t
- t is a *tuple variable*, $t[A]$ denotes the value of tuple t on attribute A
- $t \in r$ denotes that tuple t is in relation r
- P is a *formula* similar to that of the predicate calculus (FOL)

TRC ➔ relation에 variable을 assign

TextBook
243PP

$$\{t \mid P(t)\}$$

where P is a *formula*. Several tuple variables may appear in a formula. A tuple variable is said to be a *free variable* unless it is quantified by a $\exists$ or $\forall$. Thus, in:

$$t \in instructor \wedge \exists s \in department(t[dept\_name] = s[dept\_name])$$

t is a free variable. Tuple variable s is said to be a *bound* variable.

A tuple-relational-calculus formula is built up out of *atoms*. An atom has one of the following forms:

- $s \in r$, where s is a tuple variable and r is a relation (we do not allow use of the $\notin$ operator).
- $s[x] \Theta u[y]$, where s and u are tuple variables, x is an attribute on which s is defined, y is an attribute on which u is defined, and Θ is a comparison operator ($<, \leq, =, \neq, >, \geq$); we require that attributes x and y have domains whose members can be compared by Θ.
- $s[x] \Theta c$, where s is a tuple variable, x is an attribute on which s is defined, Θ is a comparison operator, and c is a constant in the domain of attribute x.

We build up formulae from atoms by using the following rules:

- An atom is a formula.
- If P_1 is a formula, then so are $\neg P_1$ and (P_1).
- If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$.
- If $P_1(s)$ is a formula containing a free tuple variable s, and r is a relation, then

$$\exists s \in r\ (P_1(s)) \text{ and } \forall s \in r\ (P_1(s))$$

are also formulae.

TRC Queries [1/3]

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000

  $\{t \mid t \in instructor \wedge t[salary] > 80000\}$

- As in the previous query, but output only the *ID* attribute value

  $\{t \mid \exists s \in \text{instructor } (t[ID] = s[ID] \wedge s[salary] > 80000)\}$

  Notice that a relation on schema (*ID*) is implicitly defined by the query

Figure 6.01: The Instructor relation

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

TRC Queries [2/3]

- Find the names of all instructors whose department is in the Watson building

$$\{t \mid \exists s \in instructor\ (t[name] = s[name] \wedge \exists u \in department\ (u[dept\_name] = s[dept\_name] \wedge u[building] = \text{“Watson”}))\}$$

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \wedge s[semester] = \text{“Fall”} \wedge s[year] = 2009 \vee \exists u \in section\ (t[course\_id] = u[course\_id] \wedge u[semester] = \text{“Spring”} \wedge u[year] = 2010)\}$$

Figure 2.05: The Department Relation

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.06: The Section relation

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E

TRC Queries [3/3]

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \wedge s[semester] = \text{“Fall”} \wedge s[year] = 2009 \wedge \exists u \in section\ (t[course\_id] = u[course\_id] \wedge u[semester] = \text{“Spring”} \wedge u[year] = 2010)\}$$

- Find the set of all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \wedge s[semester] = \text{“Fall”} \wedge s[year] = 2009 \wedge \neg \exists u \in section\ (t[course\_id] = u[course\_id] \wedge u[semester] = \text{“Spring”} \wedge u[year] = 2010)\}$$

Safety of Expressions in TRC

- It is possible to write tuple calculus expressions that generate infinite relations
- dom(P): the set of all values referenced by the TRC expression P
  - relations, tuples, or constants that appear in *P*
- An expression P such that {*t* | *P* (*t*)} in the tuple relational calculus is *safe* if all values of the result of P appears in dom(P)
- For example, { t | $\neg$ *t* $\in$ *r* } is not safe
  - dom($\neg$ *t* $\in$ *r*) *is the set of all values in relation r*
  - *However, the* result of { t | $\neg$ *t* $\in$ *r* } contains a tuple t not in dom($\neg$ *t* $\in$ *r*)
  - Furthermore, *the* result of { t | $\neg$ *t* $\in$ *r* } generates infinite tuples
- Another example, { *t* | *t* [*A*] = 5 $\vee$ **true** } is not safe
  - it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in *P*
    - *t* [*A*] could be anything
- To guard this type of problem
  - Existential Quantification
  - Closed World Assumption

Existential Quantification

- Find all students who have taken all courses offered in the Biology department

  - $\{t \mid \exists r \in student\ (t[ID] = r[ID]) \wedge$
    $(\forall u \in course\ (u[dept\_name] = \text{“Biology”} \Rightarrow$
    $\exists s \in takes\ (t[ID] = s[ID] \wedge$
    $s[course\_id] = u[course\_id]))\}$

  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.

Chapter 6: Formal Relational Query Languages

- 6.1 Relational Algebra
- 6.2 Tuple Relational Calculus
- 6.3 Domain Relational Calculus

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus

- Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P\ (x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula similar to that of the predicate calculus

DRC ➔ 모든 field에 variable을 assign

DRC Queries [1/2]

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000
  - $\{< i, n, d, s> \mid < i, n, d, s> \in instructor \wedge s > 80000\}$

- As in the previous query, but output only the *ID* attribute value
  - $\{< i> \mid < i, n, d, s> \in instructor \wedge s > 80000\}$

- Find the names of all instructors whose department is in the Watson building

$$\{< n > \mid \exists\, i, d, s\, (< i, n, d, s > \in instructor \wedge \exists\, b, a\, (< d, b, a> \in department \wedge b = \text{“Watson”}))\}$$

DRC Queries [2/2]

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section \wedge s = \text{“Fall”} \wedge y = 2009) \vee \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section \wedge s = \text{“Spring”} \wedge y = 2010)\}$$

This case can also be written as

$$\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section \wedge ((s = \text{“Fall”} \wedge y = 2009) \vee (s = \text{“Spring”} \wedge y = 2010))\}$$

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section \wedge s = \text{“Fall”} \wedge y = 2009) \wedge \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section \wedge s = \text{“Spring”} \wedge y = 2010)\}$$

Safety of Expressions in DRC

The expression:

$$\{ < x_1, x_2, \ldots, x_n > \mid P\ (x_1, x_2, \ldots, x_n)\}$$

is safe if all of the following hold:

1. All values that appear in tuples of the expression are values from *dom* (P) (that is, the values appear either in P or in a tuple of a relation mentioned in P).
2. For every "there exists" subformula of the form $\exists\ x\ (P_1(x\ ))$, the subformula is true if and only if there is a value of x in *dom* (P_1) such that $P_1(x\ )$ is true.
3. For every "for all" subformula of the form $\forall_x\ (P_1\ (x\ ))$, the subformula is true if and only if $P_1(x\ )$ is true for all values x from *dom* (P_1).

Existential Quantification

- Find all students who have taken all courses offered in the Biology department
  - $\{< i > \mid \exists\, n, d, tc\ (< i, n, d, tc > \in student\ \wedge$
    $(\forall\, ci, ti, dn, cr\ (< ci, ti, dn, cr > \in course \wedge dn = \text{“Biology”}$
    $\Rightarrow\ \exists\, si, se, y, g\ (< i, ci, si, se, y, g > \in takes\ ))\}$
  - Note that without the existential quantification on student, the above query would be unsafe if the Biology department has not offered any courses.
  - Student relation에 $< i, n, d, tc >$ 가 binding이 안되어 있으면 $< i >$ could be anything

* Above query fixes bug in page 246, last query