Chapter 6: Formal Relational Query Languages

Database System Concepts, 6th Ed.

Chapter 6: Formal Relational Query Languages

- Relational Algebra
- Tuple Relational Calculus
- Domain Relational Calculus

Relational Algebra

- Procedural language
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: $\times$
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.
  - E.g. $\prod: r \rightarrow s$ $s = \prod(r)$

Select Operation – 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

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

$$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

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

formula	:=	term
		term <conjunction> term
		(term)
term	:=	expr
		expr <op> expr
		(expr)
expr	:=	attribute
		constant

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

- Example of selection:

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

Project Operation – Example

- Relation *r*:

A	*B*	*C*
α	10	1
α	20	1
β	30	1
β	40	2

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

A	*C*
α	1
α	1
β	1
β	2

=

A	*C*
α	1
β	1
β	2

Project Operation

- 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: *instructor(ID, name, salary, dept_name)*

To eliminate the *dept_name* attribute of *instructor* write:

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

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:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- 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** (e.g.: 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

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

Set difference of two relations

- Relations *r*, *s*:

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

- *r* – *s*:

A	B
α	1
β	1

Set Difference Operation

- Notation *r – s*
- Defined as:

  $$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

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

$$\Pi_{course\_id}(\sigma_{semester="Fall" \land year=2009}(section)) -$$
$$\Pi_{course\_id}(\sigma_{semester="Spring" \land year=2010}(section))$$

Cartesian-Product 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

- $r \times s$:

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

Cartesian-Product Operation

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

$$r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

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

Composition of Operations

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

- $r \times s$

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

- $\sigma_{A=C}(r \times s)$

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

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

- 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
      - $\Pi_{salary}(instructor) - \Pi_{instructor.salary}(\sigma_{instructor.salary < d.salary}(instructor \times \rho_d(instructor)))$

Example Queries

- Find the names of all instructors in the Physics department, along with the *course_id* of all courses they have taught
  - Query 1

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

  - Query 2

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

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
  - $\prod_S(E_1)$, 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
- Outer join
- Assignment

Set-Intersection Operation

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

Set-Intersection Operation – Example

- 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 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:

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

  $s(S)$, where $S = (E, B, D)$

  - Result schema of $r \bowtie s$ is (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
  - $\Pi_{name,\ title}(\sigma_{dept\_name=\text{'Comp. Sci.'}}(instructor \bowtie teaches \bowtie course))$
- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$ is equivalent to $instructor \bowtie (teaches \bowtie course)$
- Natural join is commutative
  - $instructor \bowtie teaches$ is equivalent to $teaches \bowtie instructor$
- The **theta join** operation $r \bowtie_\theta s$ is defined as
  - $r \bowtie_\theta s = \sigma_\theta(r \times s)$

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 – Example

- Relation *instructor1*

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

- Relation *teaches1*

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

Outer Join – Example

- Join

instructor $\bowtie$ *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

- 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

Outer Join using Joins

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

    $(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$

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)

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 there is a special operator "**is null**", so "*P* **is null**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

Extended Relational-Algebra-Operations

- Generalized Projection
- Aggregate Functions

Generalized Projection

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

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

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n are 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

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

  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

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

sum(*c*)
27

Aggregate Operation – Example

- Find the average salary in each department

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

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

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

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

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 ($\leftarrow$)

Deletion

- 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.
- Example:
  - Delete all account records in the Perryridge branch.

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

Insertion

- 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.
- Example:
  - Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

$$account \leftarrow account \cup \{(\text{"A-973"}, \text{"Perryridge"}, 1200)\}$$

$$\text{depositor} \leftarrow depositor \cup \{(\text{"Smith"}, \text{"A-973"})\}$$

Updating

- 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
- Example:
  - Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \prod_{account\_number,\ branch\_name,\ balance\ *\ 1.05} (account)$$

Multi-set Relational Algebra

- Pure relational algebra removes all duplicates
  - e.g. after projection
- Multi-set relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature
- Multi-set 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 $m \times n$ copies of *t1.t2* in $r \times s$
  - Other operators similarly defined
    - E.g. union: *m + n copies,* intersection: min(*m, n*) copies difference: max(0, *m – n*) copies

Relational Algebra and SQL

- Assume the following expressions in multi-set relational algebra:
- $\Pi_{A1,\ ..,\ An} (\sigma_P (r1 \times\ r2 \times \ldots \times rm))$

  is equivalent to the following expression in SQL

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

- $_{A1,\ A2}\mathcal{G}_{\mathbf{sum}(A3)} (\sigma_P (r1 \times r2 \times \ldots \times rm)))$

  is equivalent to the following expression in SQL

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

SQL and Relational Algebra

- 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

$$\Pi_{A1,sumA3}(_{A1,A2}\mathcal{G}_{\textbf{sum}(A3)\ \textbf{as}\ \text{sumA3}}(\sigma_P(r1 \times r2 \times \ldots \times rm)))$$

Tuple Relational Calculus

Tuple Relational Calculus

- A nonprocedural query language, where each 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

Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<, \leq, =, \neq, >, \geq$)
3. Set of connectives: and ($\wedge$), or ($\vee$), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - $\exists\, t \in r\, (Q\, (t\,)) \equiv$ "there exists" a tuple *t* in relation *r* such that predicate *Q* (*t*) is true
   - $\forall t \in r\, (Q\, (t\,)) \equiv Q$ is true "for all" tuples *t* in relation *r*

Example Queries

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

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

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

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

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

Example Queries

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

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

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

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

Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, $\{ t \mid \neg\, t \in r \}$ results in an infinite relation if the domain of any attribute of relation *r* is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression $\{t \mid P(t)\}$ in the tuple relational calculus is *safe* if every component of *t* appears in one of the relations, tuples, or constants that appear in *P*
  - NOTE: this is more than just a syntax condition.
    - E.g. $\{ t \mid t[A] = 5 \vee$ **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*.

Universal 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.

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

Example Queries

- 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

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

Example Queries

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

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

This case can also be written as

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

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

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

Safety of Expressions

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).

Universal 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.

* Above query fixes bug in page 246, last query

Tento obrázek nyní nelze zobrazit.

End of Chapter 6