CS425 – Fall 2016
Boris Glavic
Chapter 3: Formal Relational Query Languages

Modified from:

Database System Concepts, 6th Ed.

Chapter 3: Formal Relational Query Languages

- **Relational Algebra**
- Tuple Relational Calculus
- Domain Relational Calculus

Textbook: Chapter 6

Relational Query Languages

- Procedural vs non-procedural (**declarative**)
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Expressive power of a query language
  - What queries can be expressed in this language?
- Relational algebra:
  - Algebra of relations -> set of operators that take relations as input and produce relations as output
  - -> **composable**: the output of evaluating an expression in relational algebra can be used as input to another relational algebra expression
- Now: First introduction to operators of the relational algebra

Relational Algebra

- Procedural language
- 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.
  - **composable**

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:

$$\sigma_p(r) = \{t \mid t \in r \wedge p(t)\}$$

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:

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

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:

$$_{A_1, A_2, \ldots, A_k}(r)$$

  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
- Let A be a subset of the attributes of relation r then:

$$\pi_A(r) = \{t.A \mid t \in r\}$$

- Example: To eliminate the *dept_name* attribute of *instructor*

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

Union Operation – Example

- 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
α	2
β	1
β	3

Union Operation

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

$$r \cup s = \{t \mid t \in r \vee t \in s\}$$

- 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** (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*:

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:

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

- 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

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

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* 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

Cartesian-Product Operation

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

$$r \times s = \{t, t' \mid t \in r \wedge t' \in s\}$$

- 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:

$$\rho_X(r)$$

returns the expression E under the name X

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

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

returns the result of expression E under the name X, and with the attributes renamed to A_1, A_2, …., A_n.

$$\rho_X(r) = \{t(X) \mid t \in r\}$$

$$\rho_{X(A)}(r) = \{t(X).A \mid t \in r\}$$

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

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

  - Query 2

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

Formal Definition (Syntax)

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation: e.g., {(1),(2)}
- 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

Formal Definition (Semantics)

- Let E be an relational algebra expression. We use [E](I) to denote the evaluation of E over a database instance I
  - For simplicity we will often drop I and []
- The result of evaluating a simple relational algebra expression E over a database is defined as
  - Simple relation: [R](I) = R(I)
  - Constant relation: [C](I) = C

Formal Definition (Semantics)

- Let E_1 and E_2 be relational-algebra expressions.

$$[E_1 \cup E_2] = \{t \mid t \in [E_1] \lor t \in [E_2]\}$$

$$[E_1 - E_2] = \{t \mid t \in [E_1] \land t \notin [E_2]\}$$

$$[E_1 \times E_2] = \{t, t' \mid t \in [E_1] \land t' \in [E_2]\}$$

$$[\sigma_p(E_1)] = \{t \mid t \in [E_1] \land p(t)\}$$

$$[\pi_A(E_1)] = \{t.A \mid t \in [E_1]\}$$

$$[\rho_X(E_1)] = \{t(X) \mid t \in [E_1]\}$$

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)
- For logical expressions
  - True AND null = null, False AND null = False
  - True OR null = True, False OR null = null
  - Not (null) = null

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*

Additional Operations

We define additional operations that do not add any expressive 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:

$$r \cap s = \{t \mid t \in r \wedge t \in s\}$$

- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Note: $r \cap s = r - (r - s)$
  - That is adding intersection to the language does not make it more expressive

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 = (*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 Operation (cont.)

- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, r ⋈ s is defined as:

$$X = R \cap S$$

$$S' = S - R$$

$$r \bowtie s = \pi_{R,S'}(\sigma_{r.X=s.X}(r \times s))$$

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* ⋈ *teaches* ⋈ *course*))
- Natural join is associative
  - (*instructor* ⋈ *teaches*) ⋈ *course* is equivalent to *instructor* ⋈ (*teaches* ⋈ *course*)
- Natural join is commutative (we ignore attribute order)
  - *instruct* ⋈ *teaches* is equivalent to *teaches* ⋈ *instructor*
- The **theta join** operation $r \bowtie_\theta s$ is defined as

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

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.

$$E_1 \leftarrow \sigma_{salary > 40000}(instructor)$$

$$E_2 \leftarrow \sigma_{salary < 10000}(instructor)$$

$$E_3 \leftarrow E_1 \cup E_2$$

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

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

$$r ⟖ s = (r \bowtie s) \cup (\{(null, \ldots, null)\} \times (s - \Pi_S(r \bowtie s)))$$

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

Division Operator

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

  $t \times s \subseteq r$

  - Alternatively, all tuples from r.(R-S) such that all their extensions on $R \cap S$ with tuples from s exist in R
- E.g. let $r(ID, course\_id) = \Pi_{ID,\ course\_id}(takes)$ and

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

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

$$E_1 \leftarrow \Pi_{R-S}(r)$$

$$E_2 \leftarrow \Pi_{R-S}((E_1 \times s) - \Pi_{R-S,S}(r \bowtie s))$$

$$r \div s = E_1 - E_2$$

Division Operator Example

- Return the name of all persons that read all newspapers

reads

name	*newspaper*
Peter	Times
Bob	Wall Street
Alice	Times
Alice	Wall Street

newspaper

newspaper
Times
Wall Street

$$E_1 \leftarrow \Pi_{name}(reads)$$

$$E_2 \leftarrow ((E_1 \times newspaper) - \Pi_{name,newspaper}(reads \bowtie newspaper))$$

$$reads \div newspaper = E_1 - E_2$$

$$[reads \div newspaper] = \{(Alice)\}$$

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.

$$\pi_{F_1,\ldots,F_n}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n are arithmetic expressions and function calls 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

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

- *Adding functions increases expressive power!*
  - *In standard relational algebra there is no way to change attribute values*

Aggregate Functions and Operations

- **Aggregation function** takes a set 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 …, 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

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

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

Aggregate Functions (Cont.)

- What are the names for attributes in aggregation results?
  - Need some convention!
    - E.g., use the expression as a name **avg**(salary)
  - 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
- Example: Delete instructors with salary over $1,000,000

$$R \leftarrow R - (\sigma_{salary > 1000000}(R))$$

Restrictions for Modification

- Consider a modification where R=(A,B) and S=(C)

$$R \leftarrow \sigma_{C>5}(S)$$

- This would change the schema of R!
  - Should not be allowed

- Requirements for modifications
  - The name **R** on the left-hand side of the assignment operator refers to an existing relation in the database schema
  - The expression on the right-hand side of the assignment operator should be union-compatible with **R**

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 logical connectives: and ($\wedge$), or (v), 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 in 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, because

1) t is not bound to any relation by the predicate

2) we implicitly state that t has an ID attribute (*t[ID] = s[ID]*)

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

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

Example Queries

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

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

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

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

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. $\{ t \mid t[A] = 5 \vee \textbf{true} \}$ 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
    - Variables that range of attribute values
  - P represents a formula similar to that of the predicate calculus
  - Tuples can be formed using <>
    - E.g., <'Einstein','Physics'>

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

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

This case can also be written as

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

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

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

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 as constants 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

Relationship between Relational Algebra and Tuple (Domain) Calculus

- **Codd's theorem**
  - Relational algebra and tuple calculus are equivalent in terms of expressiveness

- That means that every query expressible in relational algebra can also be expressed in tuple calculus and vice versa
- Since domain calculus is as expressive as tuple calculus the same holds for the domain calculus
- Note: Here relational algebra refers to the standard version (no aggregation and projection with functions)

End of Chapter 3

Modified from:

Database System Concepts, 6^{th} Ed.

Recap

- **Query language**
  - **Declarative**
  - Retrieve, combine, and analyze data from a database instance
- **Relational algebra**
  - Standard relational algebra:
    - Selection, projection, renaming, cross product, union, set difference
  - Null values
  - Semantic sugar operators:
    - Intersection, joins, division,
  - Extensions:
    - Aggregation, extended projection
- **Tuple Calculus**
  - safety
- **Domain Calculus**

Outline

- Introduction
- Relational Data Model
- Formal Relational Languages (relational algebra)
- **SQL - Introduction**
- Database Design
- Transaction Processing, Recovery, and Concurrency Control
- Storage and File Structures
- Indexing and Hashing
- Query Processing and Optimization

Figure 6.01

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

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

Figure 6.03

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 6.04

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

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

Figure 6.06

course_id
CS-347
PHY-101

Figure 6.07

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

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

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

Figure 6.11

salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

Figure 6.12

salary
95000

Figure 6.13

course_id
CS-101

Figure 6.14

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

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

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 6.17

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

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

Figure 6.19

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

Figure 6.20

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

Figure 6.21

name
Einstein
Crick
Gold

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.

Deletion Examples

- 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

- 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

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

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

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

- 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.

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

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

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

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

Update Examples

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

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

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.06}\ (\sigma_{\,BAL\ >\ 10000}\ (account\ ))$$
$$\cup\ \Pi_{\,account\_number,\ branch\_name,\ balance\ *\ 1.05}\ (\sigma_{BAL\ \leq\ 10000}$$
$$(account))$$

Example Queries

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\_name}\ (borrower) \cap \Pi_{customer\_name}\ (depositor)$$

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer\_name,\ loan\_number,\ amount}\ (borrower \bowtie loan)$$

Example Queries

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.
  - Query 1

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Downtown”}}(depositor \bowtie account)) \cap$$

$$\Pi_{customer\_name}(\sigma_{branch\_name = \text{“Uptown”}}(depositor \bowtie account))$$

  - Query 2

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \rho_{temp(branch\_name)}(\{(\text{“Downtown”}), (\text{“Uptown”})\})$$

Note that Query 2 uses a constant relation.

Bank Example Queries

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_city = \text{“Brooklyn”}}(branch))$$