Chapter 6: Relational Algebra and Relational Calculus

Last Lecture:

- More Relational Algebra
- Division Operator
- Null Values and Bags in Relational Algebra
- Introduction to Relational Calculus

Today:

- Tuple Relational Calculus
- Domain 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., \langle , \leq , =, \neq , \rangle , \geq)
- 3. Set of connectives: and (\land) , or (\lor) , not (\neg)
- 4. Implication (\Rightarrow): $x \Rightarrow y$, if x is true, then y is true $x \Rightarrow y \equiv \neg x \lor y$
- 5. Set of quantifiers:
 - ∃ t ∈ r (Q (t)) ≡ "there exists" a tuple in t in relation r such that predicate Q (t) is true

Don't use the notation in blue

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) \ (s \in \text{instructor} \land t [ID] = s [ID] \land s [salary] > 80000)\}
```

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

the query

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

```
\{t \mid \exists s \in instructor\ (t [name] = s [name] \land \exists u \in department\ (u [dept_name] = s[dept_name] \land u [building] = "Watson"))\}
```

Find the set of all courses taught in the Fall 2015 semester, or in

the Spring 2016 semester, or both

```
\{t \mid \exists s \in section \ (t [course\_id] = s [course\_id] \land s [semester] = "Fall" \land s [year] = 2015

\lor \exists u \in section \ (t [course\_id] = u [course\_id] \land u [semester] = "Spring" \land u [year] = 2016)\}
```

■ Find the set of all courses taught in the Fall 2015 semester, and in

```
the Spring 2016 semester \{t \mid (\exists s) \ (s \in section \land t \mid course\_id \ ] = s \mid course\_id \ ] \land s \mid semester \mid = "Fall" \land s \mid sear \mid = 2015 \land (\exists u) \ (u \in section \land t \mid course\_id \ ] = u \mid course\_id \ ] \land u \mid semester \mid = "Spring" \land u \mid sear \mid = "
```

2016)}
■ Find the set of all courses taught in the Fall 2015 semester, but not in

```
the Spring 2016 semester \{t \mid (\exists s) \ (s \in section \land t \ [course\_id \ ] = s \ [course\_id \ ] \land s \ [semester] = "Fall" \land s \ [year] = 2015
 \land \neg (\exists u) \ (u \in section \land t \ [course\_id \ ] = u \ [course\_id \ ] \land u \ [semester] = "Spring" \land u \ [year] = 2016)\}
```

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
- 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 | t [A] = 5 ∨ **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) \ (r \in student \land t \mid ID] = r \mid ID] \land (\forall u) \ (u \in course \land u \mid dept\_name] = "Biology" \Rightarrow (\exists s)(s \in takes \land t \mid ID] = s \mid ID \mid \land s \mid course\_id] = u \mid (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.

Manage (*pid*: integer, *mid*: integer)

Movie (<u>mid: integ</u>er, <u>budget</u>: real, <u>managerid</u>: a) Find the நக்கு of producers who manage at least one movie with budget larger than 1 million

b) Find the names of producers who manage all movies with budget larger than 1 million

c) Find the names of producers who manage only movies with budget larger than 1 million

Manage (pid: integer, mid: integer)

Movie (<u>mid: integ</u>er, budget: real, managerid:
a) Find theteames of producers who manage at least one movie with budget larger than 1 million

$$\{t \mid (\exists s) \ (s \in Producer \land t [name] = s [name] \land \land (\exists u) \ (u \in Movie \land s[id] = u [managerid] \land u [budget] > 1000000)$$

Manage (pid: integer, mid: integer)

Movie (*mid*: integer, *budget*: real, *managerid*:

- a) Find theteames of producers who manage at least one movie with budget larger than 1 million
- b) Find the names of producers who manage all movies with budget larger than 1 million

$$\{t \mid (\exists s) \ (s \in Producer \land t [name] = s \ [name] \ \land (\forall u) \ (u \in Movie \land u[budget] > 1000000 \ => s[id] = u[managerid])$$

Manage (*pid*: integer, *mid*: integer)

Movie (*mid*: integer, *budget*: real, *managerid*:

- a) Find thateames of producers who manage at least one movie with budget larger than 1 million
- b) Find the names of producers who manage all movies with budget larger than 1 million
- c) Find the names of producers who manage only movies with budget larger than 1 million

$$\{t \mid (\exists s) (s \in Producer \land t [name] = s [name]\}$$

$$\land$$
 (\forall u) ($u \in Movie \land u[managerid] = s[id] = > u[budget] > 1000000)$

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:

$$\{ \langle X_1, X_2, ..., X_n \rangle \mid P(X_1, X_2, ..., X_n) \}$$

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

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

```
\{ < i, n, d, s > | < i, n, d, s > \in instructor \land s > 80000 \}
```

- As in the previous query, but output only the *ID* attribute value $\{ \langle i \rangle \mid \langle i, n, d, s \rangle \in instructor \land s > 80000 \}$
- Find the names of all instructors whose department is in the Olin building

```
\{ \langle n \rangle \mid (\exists i, d, s) (\langle i, n, d, s \rangle \in instructor \land (\exists b, a) (\langle d, b, a \rangle \in department \land b = "Olin") \}
```

■ Find the set of all courses taught in the Fall 2015 semester, or in the Spring 2016 semester, or both

```
\{ \langle c \rangle \mid \exists \ a, \ s, \ y, \ b, \ r, \ t \ (\langle c, \ a, \ s, \ y, \ b, \ t \rangle \in section \land s = \text{``Fall''} \land y = 2015 \)
v \(\exists \ a, \ s, \ y, \ b, \ r, \ t \(\langle c, \ a, \ s, \ y, \ b, \ t \rangle \in section \] \(\lambda \ s = \text{``Spring''} \lambda \ y = 2016 \)\}
```

This case can also be written as

```
\{ \langle c \rangle \mid \exists \ a, \ s, \ y, \ b, \ r, \ t \ (\langle c, \ a, \ s, \ y, \ b, \ t \rangle \in section \land ((s = \text{``Fall''} \land y = 2015)) \lor (s = \text{``Spring''} \land y = 2016)) \}
```

Find the set of all courses taught in the Fall 2015 semester, and in the Spring 2016 semester

```
\{ <c> | ∃ a, s, y, b, r, t ( <c, a, s, y, b, t > ∈ section ∧ s = "Fall" ∧ y = 2015 ) 
 ∧∃ a, s, y, b, r, t ( <c, a, s, y, b, t > ∈ section ] ∧ s = "Spring" ∧ y = 2016) }
```

Safety of Expressions

The expression:

$$\{ \langle x_1, x_2, ..., x_n \rangle \mid P(x_1, x_2, ..., x_n) \}$$

is **safe** if all of the following hold:

- 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).
- 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.
- 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

```
\{ \langle i \rangle \mid \exists n, d, tc \ ( \langle i, n, d, tc \rangle \in student \land ( \forall ci, ti, dn, cr \ ( \langle ci, ti, dn, cr \rangle \in course \land dn = "Biology" 
 <math>\Rightarrow \exists si, se, y, g \ ( \langle i, ci, si, se, y, g \rangle \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

End of Chapter 6

Why Bags?

- SQL, the most important query language for relational databases, is actually a bag language.
- Some operations, like projection, are more efficient on bags than sets.

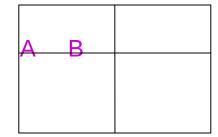
Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

R(A,		В)	
1	2			
5	6			
1	2			

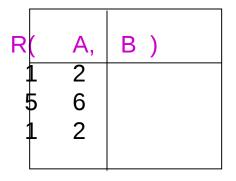
$$\mathbf{O}_{A+B<5}(R) = 1 2 1 2$$



Example: Bag Projection

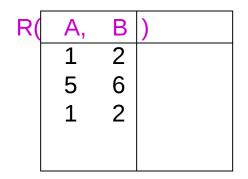
R	(A,	В)
1	2	
5	6	
1	2	

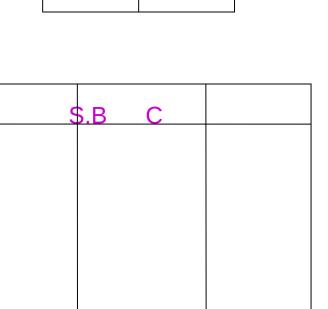
Example: Bag Product



r X s =	A R.E		S.B	С	
1 2		3	4		
1 2		7	8		
5 6		3	4		
5 6		7	8		
1 2		3	4		
1 2		7	8		

Example: Bag Theta-Join





Bag Union

- An element appears in the union of two bags the sum of the number of times it appears in each bag.
- Example: $\{1,2,1\}$ U $\{1,1,2,3,1\}$ = $\{1,1,1,1,1,2,2,3\}$

Bag Intersection

- An element appears in the intersection of two bags the minimum of the number of times it appears in either.
- Example: $\{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}.$

Bag Difference

- An element appears in the difference A B of bags as many times as it appears in A, **minus** the number of times it appears in B.
 - But never less than 0 times.
- Example: $\{1,2,1,1\} \{1,2,3\} = \{1,1\}.$

Beware: Bag Laws!= Set Laws

- Some, but *not all* algebraic laws that hold for sets also hold for bags.
- Example: the commutative law for union
 - $(R \ US = S \ UR)$ does hold for bags.
 - Since addition is commutative, adding the number of times
 x appears in R and S doesn't depend on the order of R and
 S.

Example: A Law That Fails

• Set union is *idempotent*, meaning that $S \cup S = S$.

• However, for bags, if x appears n times in S, then it appears 2n times in $S \cup S$.

- Thus $S \cup S != S$ in general.
 - e.g., $\{1\} \cup \{1\} = \{1,1\} != \{1\}.$

Bank Example

- Borrower (customer_name, loan_number, branch_name)
- Depositor(customer_name, account_number, branch_name)
- Loan (loan_number, amount, loan_type)
- Account (account_number, balance, account_type)
- Branch(branchname, branch_city, address, phone)

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

$$\prod_{customer\ name}$$
 (borrower) $\cap \prod_{customer\ name}$ (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 loan)

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

Query 1

$$\begin{split} &\Pi_{customer_name}\left(\sigma_{branch_name = \text{``Downtown''}}\left(depositor & account \right)\right) \cap \\ &\Pi_{customer_name}\left(\sigma_{branch_name = \text{``Uptown''}}\left(depositor & account \right)\right) \\ &\text{Query 2} \\ &\Pi_{customer_name, \ branch_name}\left(depositor & account \right) \\ &\div \rho_{temp(branch_name)}\left(\{(\text{``Downtown''}), (\text{``Uptown''})\}\right) \end{split}$$

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.

```
\prod_{customer\_name, \ branch\_name} (depositor \ account)
\div \prod_{branch\_name} (\sigma_{branch \ city = \text{``Brooklyn''}} (branch))
```

Relational Algebra on Bags

- A bag (or multiset) is like a set, but an element may appear more than once.
- Example: {1,2,1,3} is a bag.
- Example: {1,2,3} is also a bag that happens to be a set.