Database Management Systems

Module 09: Introduction to SQL/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced relational query language
- Familiarized with data definition and basic query structure

Module Objectives

- To complete the understanding of basic query structure

Module Outline

- Additional Basic Operations
  - Cartesian Product
  - Rename AS Operation
  - String Values
  - Order By
  - Select Top / Fetch
  - Where Clause Predicate
  - Duplicates

Additional Basic Operations

Additional Basic Operations

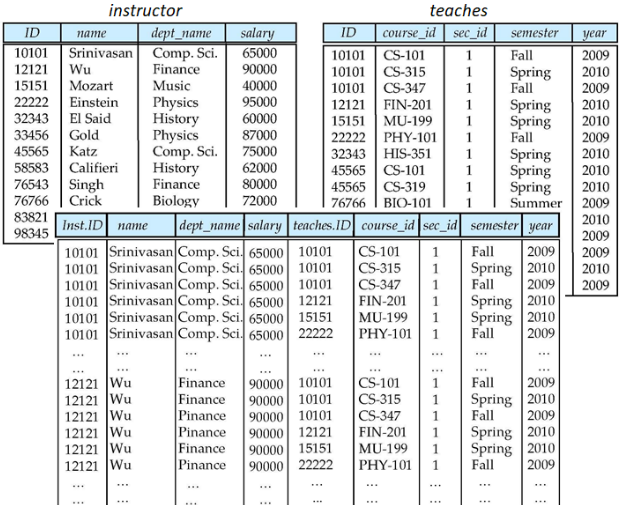
Cartesian Product

- Find the Cartesian product *instructor X teaches*

  select *
  from *instructor*, *teaches*

  - generates every possible instructor-teaches pair, with all attributes from both relations
  - For common attributes (for example, *ID*), the attributes in the resulting table are renamed using the relation name (for example, *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Cartesian Product



instructor

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			
98345			

teaches

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
				2010
				2009
				2009
				2010
				2009

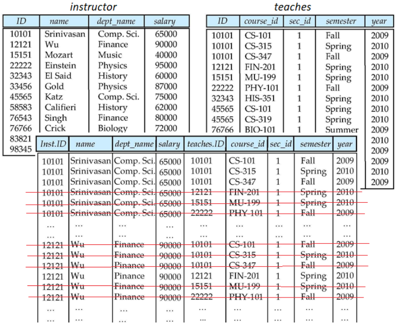
Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	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
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Examples

- Find the names of all instructors who have taught some course and the course_id

  select *name, course_id*
  from *instructor*, *teaches*
  where *instructor.ID* = *teaches.ID*

  ○ Equi-Join, Natural Join



instructor

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			
98345			

teaches

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
				2010
				2009
				2009
				2010
				2009

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~~
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Examples

- Find the names of all instructors in the Art department who have taught some course and the course_id

  select *name, course_id*
  from *instructor*, *teaches*
  where $instructor.ID = teaches.ID$ **and** $instructor.dept\_name =$ 'Art'

Rename AS Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old_name **as** *new_name*

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

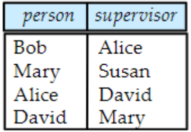
  select distinct $T.name$
  from *instructor* **as** T, *instructor* **as** S,
  where $T.salary > S.salary$ **and** $S.dept\_name =$ *'Comp. Sci'*

- Keyword **as** is optional and may be omitted

  instructor **as** $T \equiv$ *instructor* T

Cartesian Product Example

- Relation *emp_super*



person	*supervisor*
Bob	Alice
Mary	Susan
Alice	David
David	Mary

- Find the supervisor of “Bob”
- Find the supervisor of the supervisor of “Bob”
- Find ALL the supervisors (direct and indirect) of “Bob”

String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator **like** uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring
  - underscore (_). The _ character matches any character
- Find the names of all instructors whose name includes the substring "dar"

  select *name*
  from *instructor*
  where *name* **like** '%*dar*%'
- Match the string "100%"

  like '100\%' **escape** '\'
- in that above we use backslash (\) as the escape character

String Operations (2)

- Patterns are case sensitive
- Pattern matching examples:
  - 'Intro%' matches any string beginning with “Intro”
  - '%Comp%' matches any string containing “Comp” as a substring
  - '_ _ _' matches any string of exactly three characters
  - '_ _ _ %' matches any string of at least three characters
- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors

  select distinct *name*
  from *instructor*
  order by *name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**
- Can sort on multiple attributes
  - Example: **order by** *dept_name*, *name*

Selecting Number of Tuples in Output

- The **Select Top** clause is used to specify the number of records to return
- The **Select Top** clause is useful on large tables with thousands of records. Returning a large number of records can impact performance

  select top 10 **distinct** *name*
  from *instructor*

- Not all database systems support the SELECT TOP clause.
  - SQL Server & MS Access support **select top**
  - MySQL supports the **limit** clause
  - Oracle uses **fetch first** n **rows only** and **rownum**

  select distinct *name*
  from *instructor*
  order by *name*
  fetch first 10 **rows only**

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $\$90,000$ and $\$100,000$ (that is, $\geq \$90,000$ and $\leq \$100,000$)

```
select name
from instructor
where salary between 90000 and 100000
```

- Tuple comparison

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

In Operator

- The **in** operator allows you to specify multiple values in a **where** clause
- The **in** operator is a shorthand for multiple **or** conditions

  select *name*
  from *instructor*
  where *dept_name* **in** ('Comp. Sci.', 'Biology')

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result
- **Multiset** versions of some of the relational algebra operators – given multiset relations r_1 and r_2:
  a) $\sigma_\theta(r_1)$: If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$
  b) $\Pi_A(r)$: For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1
  c) r_1 x r_2: If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are c_1 x c_2 copies of the tuple $t_1.t_2$ in r_1 x r_2

Duplicates (2)

- Example: Suppose multiset relations $r_1(A, B)$ and $r_2(C)$ are as follows:
  $r_1 = \{(1, a)(2, a)\} \qquad r2 = \{(2), (3), (3)\}$
- Then $\Pi_B(r_1)$ would be $\{(a), (a)\}$, while $\Pi_B(r_1)$ x r_2 would be
  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL duplicate semantics:
  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P
  is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

Module Summary

- Completed the understanding of basic query structure

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".