Database Management Systems

Module 13: Intermediate SQL/2

Partha Pratim Das

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

ppd@cse.iitkgp.ac.in

Module Recap

- Nested subquery in SQL
- Processes for data modification

Module Objectives

- To learn SQL expressions for Join
- To learn SQL expressions for Views

Module Outline

- Join Expressions
- Views

Join Expressions

Join Expressions

Joined Relations

- **Join operations** take two relations and return as a result another relation
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition).
- It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the **from** clause

Types of Join between Relations

- Cross join
- Inner join
    - Equi-join
        - ▷ Natural join
- Outer join
    - Left outer join
    - Right outer join
    - Full outer join
- Self-join

Cross Join

- CROSS JOIN returns the Cartesian product of rows from tables in the join
  - Explicit

    select *

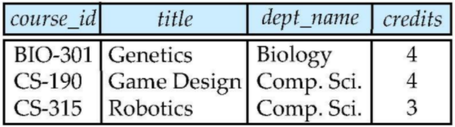
    from *employee* **cross join** *department*;
  - Implicit

    select *

    from *employee*, *department*;

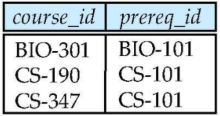
Join operations – Example

- Relation *course*



course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

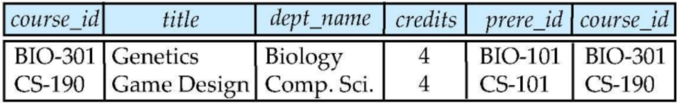


course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that
  prereq information is missing for CS-315 and
  course information is missing for CS-347

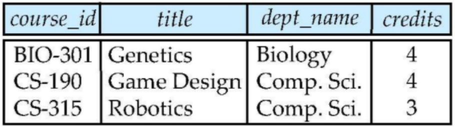
Inner Join

- *course* **inner join** *prereq*

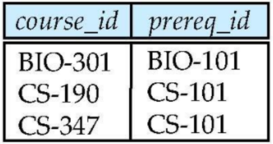


course_id	*title*	*dept_name*	*credits*	*prere_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

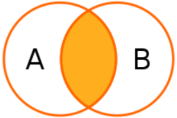
- If specified as **natural**, the 2^{nd} course_id field is skipped



course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3



course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

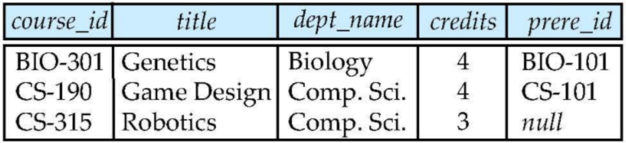


Outer Join

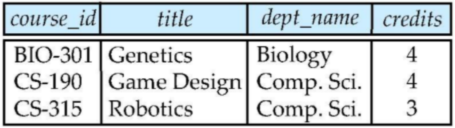
- An extension of the join operation that avoids loss of information
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join
- Uses *null* values

Left Outer Join

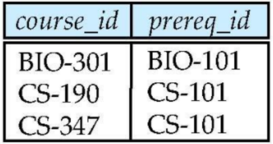
- *course* **natural left outer join** *prereq*



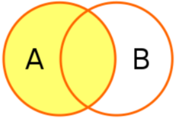
course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*



course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

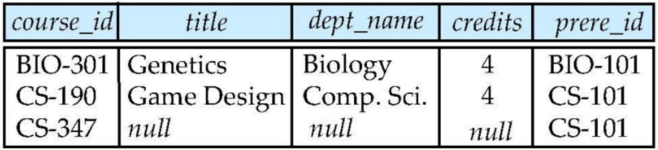


course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

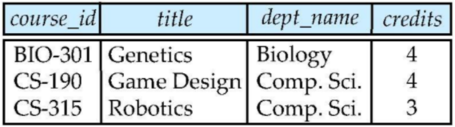


Right Outer Join

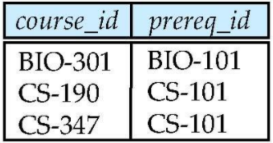
- *course* **natural right outer join** *prereq*



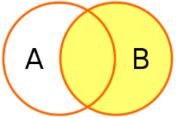
course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101



course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

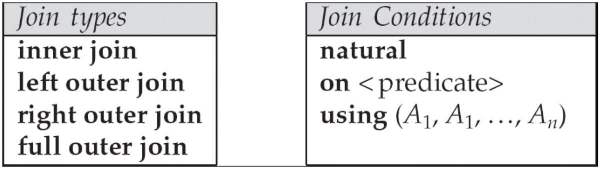


course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101



Joined Relations

- Join operations take two relations and return as a result another relation
- These additional operations are typically used as subquery expressions in the **from** clause
- Join condition – defines which tuples in the two relations match, and what attributes are present in the result of the join
- Join type – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated

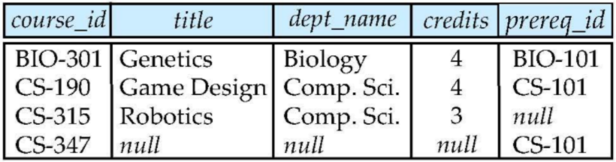


Join types
inner join
left outer join
right outer join
full outer join

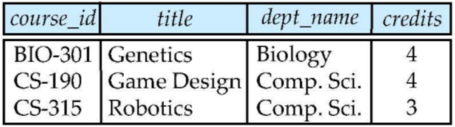
Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Full Outer Join

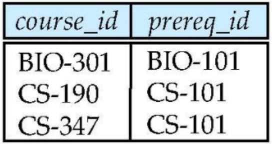
- *course* **natural full outer join** *prereq*



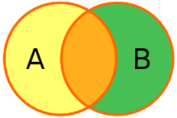
course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101



course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

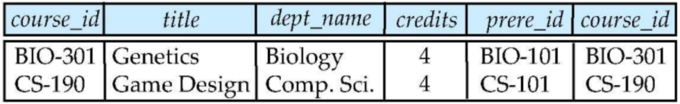


course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101



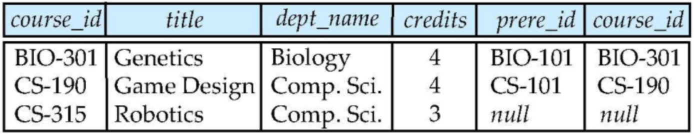
Joined Relations - Examples

- *course* **inner join** *prereq* **on** *course.course_id* = *prereq.course_id*



course_id	*title*	*dept_name*	*credits*	*prere_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

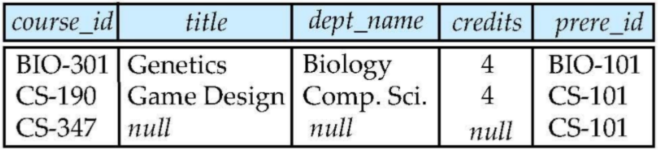
- What is the difference between the above (equi_join), and a natural join?
- *course* **left outer join** *prereq* **on** *course.course_id* = *prereq.course_id*



course_id	*title*	*dept_name*	*credits*	*prere_id*	*course_id*
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	*null*	*null*

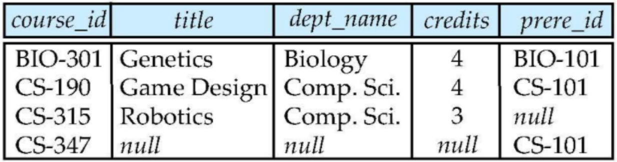
Joined Relations - Examples

- *course* **natural right outer join** *prereq*



course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

- *course* **full outer join** *prereq* **using** *(course_id)*



course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Views

Views

Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

  select *ID, name, dept_name*
  from *instructor*
- A **view** provides a mechanism to hide certain data from the view of certain users
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** < query expression >

  where < query expression > is any legal SQL expression
- The view name is represented by *v*
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view

Example Views

- A view of instructors without their salary

  create view *faculty* **as**
      select *ID*, *name*, *dept_name*
      from *instructor*

- Find all instructors in the Biology department

  select *name*
      from *faculty*
      where *dept_name* = 'Biology'

- Create a view of department salary totals

  create view *departments_total_salary(dept_name, total_salary)* **as**
      select *dept_name*, **sum** (*salary*)
      from *instructor*
      group by *dept_name*;

Views Defined Using Other Views

Module 13

Partha Pratim Das

Objectives & Outline

Join Expressions
Cross Join
Inner Join
Outer Join
Left Outer Join
Right Outer Join
Full Outer Join

Views
View Expansion
View Update
Materialized Views

Module Summary

- **create view** *physics_fall_2009* **as**
  select *course.course_id*, *sec_id*, *building*, *room_number*
  from *course*, *section*
  where *course.course_id* = *section.course_id*
  and *course.dept_name* = 'Physics'
  and *section.semester* = 'Fall'
  and *section.year* = '2009';
- **create view** *physics_fall_2009_watson* **as**
  select *course_id*, *room_number*
  from *physics_fall_2009*
  where *building*= 'Watson';

View Expansion

- Expand use of a view in a query/another view

```
create view physics_fall_2009_watson as
          (select course_id, room_number
          from (select course.course_id, building, room_number
                    from course, section
                    where course.course_id = section.course_id
                              and course.dept_name = 'Physics'
                              and section.semester = 'Fall'
                              and section.year = '2009')
          where building= 'Watson');
```

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to *depend directly* on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to *depend on* view relation v_2 if either v_1 depends directly on v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be *recursive* if it depends on itself

View Expansion*

- A way to define the meaning of views defined in terms of other views
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations
- View expansion of an expression repeats the following replacement step:

```
repeat
    Find any view relation $v_i$ in $e_1$
    Replace the view relation $v_i$ by the expression defining $v_i$
until no more view relations are present in $e_1$
```

- As long as the view definitions are not recursive, this loop will terminate

Update of a View

- Add a new tuple to *faculty* view which we defined earlier

  insert into *faculty* **values** ('30765', 'Green', 'Music');

- This insertion must be represented by the insertion of the tuple

  ('30765', 'Green', 'Music', null)

  into the *instructor* relation

Some Updates cannot be Translated Uniquely

- **create view** *instructor_info* **as**
  select *ID, name, building*
  from *instructor, department*
  where *instructor.dept_name*= *department.dept_name*;
- **insert into** *instructor_info* **values** ('69987', 'White', 'Taylor');
  - which department, if multiple departments in Taylor?
  - what if no department is in Taylor?
- Most SQL implementations allow updates only on simple views
  - The **from** clause has only one database relation
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group by** or **having** clause

And Some Not at All

- **create view** *history_instructors* **as**
  select *
  from *instructor*
  where *dept_name*= 'History';
- What happens if we insert ('25566', 'Brown', 'Biology', 100000) into history_instructors?

Materialized Views

- **Materializing a view**: create a physical table containing all the tuples in the result of the query defining the view
- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated

Module Summary

- Learnt SQL expressions for Join and Views

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".