



Chapter 4: Intermediate SQL

Join Expressions, Views

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Chapter 4: Intermediate SQL

- Join Expressions
- Views



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



Join operations – Example

■ Relation *course*

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

■ Relation *prereq*

<i>course_id</i>	<i>prereq_id</i>
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-437



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

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	<i>null</i>



Right Outer Join

- course natural right outer join *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prereq_id</i>
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	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.

<i>Join types</i>
inner join
left outer join
right outer join
full outer join

<i>Join Conditions</i>
natural
on <predicate>
using (A_1, A_1, \dots, 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



Joined Relations – Examples

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

course_id	title	dept_name	credits	prereq_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, and a natural join?
- **course left outer join prereq on**
course.course_id = prereq.course_id

course_id	title	dept_name	credits	prereq_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	prereq_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	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



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

- **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';



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.



Chapter 5: Advanced SQL

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan

See www.db-book.com for conditions on re-use



Recursive Queries



Recursion in SQL

- SQL:1999 permits recursive view definition
- Example: find which courses are a prerequisite, whether directly or indirectly, for a specific course

```
with recursive rec_prereq(course_id, prereq_id) as (
    select course_id, prereq_id
    from prereq
    union
    select rec_prereq.course_id, prereq.prereq_id,
    from rec_prereq, prereq
    where rec_prereq.prereq_id = prereq.course_id
)
select *
from rec_prereq;
```

This example view, *rec_prereq*, is called the *transitive closure* of the *prereq* relation



The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration.
 - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *prereq* with itself
 - ▶ This can give only a fixed number of levels of managers
 - ▶ Given a fixed non-recursive query, we can construct a database with a greater number of levels of prerequisites on which the query will not work
 - ▶ Alternative: write a procedure to iterate as many times as required
 - See procedure *findAllPrereqs* in book



The Power of Recursion

- Computing transitive closure using iteration, adding successive tuples to *rec_prereq*
 - The next slide shows a *prereq* relation
 - Each step of the iterative process constructs an extended version of *rec_prereq* from its recursive definition.
 - The final result is called the *fixed point* of the recursive view definition.
- Recursive views are required to be **monotonic**. That is, if we add tuples to *prereq* the view *rec_prereq* contains all of the tuples it contained before, plus possibly more