



Chapter 4: Intermediate SQL



Chapter 4: Intermediate SQL

■ Join Expressions

- join types: inner join, left/right/full outer join
- join conditions: natural, on <predicate>, using

■ Views

■ Integrity Constraints

- primary key
- foreign key
- not null
- unique
- check <predicate>
- assertion



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

<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

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101



Left Outer Join

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

<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

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101



Left Outer Join

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

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prere_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>

<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

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101



Right Outer Join

■ *course* **natural right outer join** *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prere_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

<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

<i>course_id</i>	<i>prereq_id</i>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101



Full Outer Join

■ *course* **natural full outer join** *prereq*

<i>course_id</i>	<i>title</i>	<i>dept_name</i>	<i>credits</i>	<i>prere_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>
CS-347	<i>null</i>	<i>null</i>	<i>null</i>	CS-101

<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

<i>course_id</i>	<i>prereq_id</i>
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.
- **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 condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.

<i>Join types</i>	<i>Join Conditions</i>
inner join left outer join right outer join full outer join	natural on <predicate> using $(A_1, A_1, \dots, A_n)$



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

- **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

- **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

- 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.)
- A **view** provides a mechanism to hide certain data from the view of certain users.
- 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'



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.



Constraints on a Single Relation

- **not null**
- **primary key**
- **unique** ($A_1, A_2, \dots, A_m$)
 - The unique specification states that the attributes $A_1, A_2, \dots, A_m$ form a candidate key.
 - Candidate keys are permitted to be null (in contrast to primary keys).
- **check** (P), where P is a predicate

```
create table teaches (  
    course_id varchar (8),  
    instructor_id varchar (8),  
    semester varchar (6),  
    year numeric (4,0),  
    classroom varchar (6) not null,  
    primary key (course_id, instructor_id, semester, year),  
    // implies unique (course_id, instructor_id, semester, year),  
    check (semester in ('Fall', 'Winter', 'Spring', 'Summer'))  
);
```



Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
 - Example: If “Biology” is a department name appearing in one of the tuples in the *instructor* relation, then there exists a tuple in the *department* relation for “Biology”.

- **create table** *course* (
 course_id **char(5) primary key**,
 title **varchar(20)**,
 dept_name **varchar(20) references department**
)



Assertion

- **create assertion** <assertion-name> **check** <predicate>;
- Every loan has at least one borrower who maintains an account with a minimum balance of \$1000.00

```
create assertion balance_constraint check  
  (not exists (  
    select *  
  from loan  
  where not exists (  
    select *  
    from borrower, depositor, account  
    where loan.loan_number = borrower.loan_number  
      and borrower.customer_name = depositor.customer_name  
      and depositor.account_number = account.account_number  
      and account.balance >= 1000)))
```



Chapter 4: Intermediate SQL

■ Join Expressions

- join types: inner join, left/right/full outer join
- join conditions: natural, on <predicate>, using

■ Views

■ Integrity Constraints

- primary key
- foreign key
- not null
- unique
- check <predicate>
- assertion



End of Chapter 4