Intro to DB

CHAPTER 4
INTERMEDIATE SQL

Chapter 4: Intermediate SQL

- Join Expressions
- Views
- Transactions
- Integrity Constraints
- SQL Data Types and Schemas
- Authorization

Joined Relations

- **Join operation**
  - takes two relations and return as a result another relation.
  - a Cartesian product which requires that tuples in the two relations match (under some condition).
  - 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 – On

- Join matches tuples with the same values specified in the on condition.

select *
from *instructor* **join** *teaches* **on** *(instructor.ID=teaches.ID)*; ⇒ from instructor, teaches where instructor.ID = teaches.ID

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

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
76766	BIO-301	1	Summer	2010

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

Join – On (cont.)

- Let $r_1(A_1, \ldots, A_k)$, $r_2(B_1, \ldots, B_n)$

  select *
  from r_1 **join** r_2 **on** *condition*;

  is equivalent to

  select *
  from r_1, r_2
  where *condition*

- You can use expressions (that evaluate to relations) in **from** clauses

  from $E_1, \ldots, E_n$

  kind of subquery

  select distinct *name, title*
  from (*instructor* **join** *teaches* **on** (*instructor.ID=teaches.ID*)), *course*
  where *teaches.course_id=course.course_id*

Join Operations

- take two relations and return as a result another relation.
- 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 $(A_1, A_1, \ldots, A_n)$

Inner join

- Inner join = join (default)

  select * **from** *instructor* **join** *teaches* **on** *(instructor.ID=teaches.ID)*;

  = **select** * **from** *instructor* **inner join** *teaches* **on** *(instructor.ID=teaches.ID)*;

  ↳ left only correspond result

- ON clause
  - an explicit join clause (acts like a **where** clause)
- USING clause
  - specifies which columns to test for *equality*
  - columns listed must be present in both of the two tables being joined

- Above example is equivalent to

  select * **from** *instructor* **inner join** *teaches* **using** *(ID)*;

  A B column-list

Natural Join

- (Section 3.3.3)
- *Natural join* matches tuples with the same values for all common attributes, and retains only one copy of each common column

  select *
  from *instructor* **natural join** *teaches*;

- Let $r_1(A_1, \ldots, A_k, C_1, \ldots, C_m)$, $r_2(B_1, \ldots, B_n, C_1, \ldots, C_m)$

  select *
  from r_1 **natural join** r_2

  is equivalent to

  select $A_1, \ldots, A_k, r_1.C_1, \ldots, r_1.C_m, B_1, \ldots, B_n$
  from r_1, r_2
  where $r_1.C_1=r_2.C_1$ **and** … **and** $r_1.C_m=r_2.C_m$

Natural Join (cont.)

- Find the IDs of all students who were taught by an instructor named Einstein in building 301 (remove duplicates in the result).

```
select   distinct takes.ID
from     takes, section, teaches, instructor
where    takes.course_id=section.course_id  and  takes.sec_id=section.sec_id and
         takes.semester=section.semester  and  takes.year=section.year and
         section.course_id=teaches.course_id and section.sec_id=teaches.sec_id and
         section.year=teaches.year and section.year=teaches.year and
         teaches.ID=instructor.ID  and
         instructor.name='Einstein'  and  building='301'
```

```
select   distinct takes.ID
from     (takes natural join section)
         join teaches using (crse_id, sec_id, smster, year)
         join instructor on (teaches.ID=instructor.ID)
where    instructor.name='Einstein' and building='301'
```

→ due to takes.ID (ambigious!)

Natural Join (cont.)

- **Danger in natural join**: beware of unrelated attributes with same name which get equated incorrectly
- List the names of instructors along with the titles of courses that they teach
  - Incorrect version (makes course.dept_name = instructor.dept_name)
    - **select distinct** *name*, *title*
      from *instructor* **natural join** *teaches* **natural join** *course*;

    → if physic's dept instructor teach electronic dept.

  - Correct version
    - **select distinct** *name*, *title*
      from *instructor* **natural join** *teaches*, *course*
      where *teaches.course_id* = *course.course_id*;

    (extra condition)

  - Another correct version
    - **select distinct** *name*, *title*
      from (*instructor* **natural join** *teaches*)
      join *course* **using** (*course_id*);

Join operation – Example

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

prereq

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

course **natural 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

Note: prereq information missing for CS-315 and course information missing for CS-437.

(only left exist value)

Outer Join

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

Left Outer Join : preserve left table's dangling data

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*

↳ unmatched, but survive

= *course* **left outer join** *prereq* **on** *course.course_id = prereq.course_id*
= *course* **left outer join** *prereq* **using** (*course_id*)

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

prereq

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

Right Outer Join : reverse of left

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

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

prereq

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

Full Outer Join : left + right

course **natural full 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*
CS-347	*null*	*null*	*null*	CS-101

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

prereq

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

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*

Summary – Joined Operations

- **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 $(A_1, A_1, \ldots, A_n)$

Integrity Constraints

- Integrity constraints guard against accidental damage to the database
  - by ensuring that authorized changes to the database do not result in a loss of data consistency
- Examples
  - A checking account must have a balance greater than $10,000.00
  - A salary of a bank employee must be at least $4.00 an hour
  - A customer must have a (non-null) phone number
- Constraints on a single relation
  - **not null**
  - **unique**
  - **primary key**
  - **check** (*P*), where *P* is a predicate
- Referential integrity

Data integrity : meet the rule of 'model'

↳ primary key - foreign key

Not Null Constraint

- Declare *name* for *budget* to be **not null**

  name **varchar**(20) **not null**
  budget **numeric**(12,2) **not null**

- Example

  create table *instructor* (
  ID **char**(5) **not null**,
  name **varchar**(20) **not null,**
  dept_name **varchar**(20),
  salary **numeric**(8,2))

The Unique & Primary Key Constraints

- **unique** ($A_1, A_2, \ldots, A_m$)
  - $A_1, A_2, \ldots A_m$ form a candidate key
  - can be null unless declared to be non null explicitly

- **primary key** ($A_1, A_2, \ldots, A_m$)
  - $A_1, A_2, \ldots A_m$ form the primary key for the table
  - Non null & unique

```
create table instructor (
        ID              char(5),
        name            varchar(20) not null,
        dept_name       varchar(20),
        salary          numeric(8,2)
        primary key (ID) );
```

unique(ID) → primary key(ID, name)

The check clause

- **check** (*P*), where *P* is a predicate
- Example:
  - ensure that semester is one of fall, winter, spring or summer

integrity constraints ↑
⇒ overhead also increase

```
create table section (
        course_id varchar (8),
        sec_id varchar (8),
        semester varchar (6),
        year numeric (4,0),
        building varchar (15),
        room_number varchar (7),
        time slot id varchar (4),
        primary key (course_id, sec_id, semester, year),
        check (semester in ('Fall', 'Winter', 'Spring', 'Summer') )
)
```

must be one of this lists

Referential Integrity

- "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 *course* relation,
    - then there exists a tuple in the *department* relation for "Biology".
- specified as part of **create table** statement
- **Foreign Key** ⟶ == check (A in B.primary_key)
  - the attributes that comprise the foreign key, and
  - the name of the relation referenced by the foreign key
  - By default, a foreign key references the primary key of the referenced table.

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Cascading Actions in Referential Integrity

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

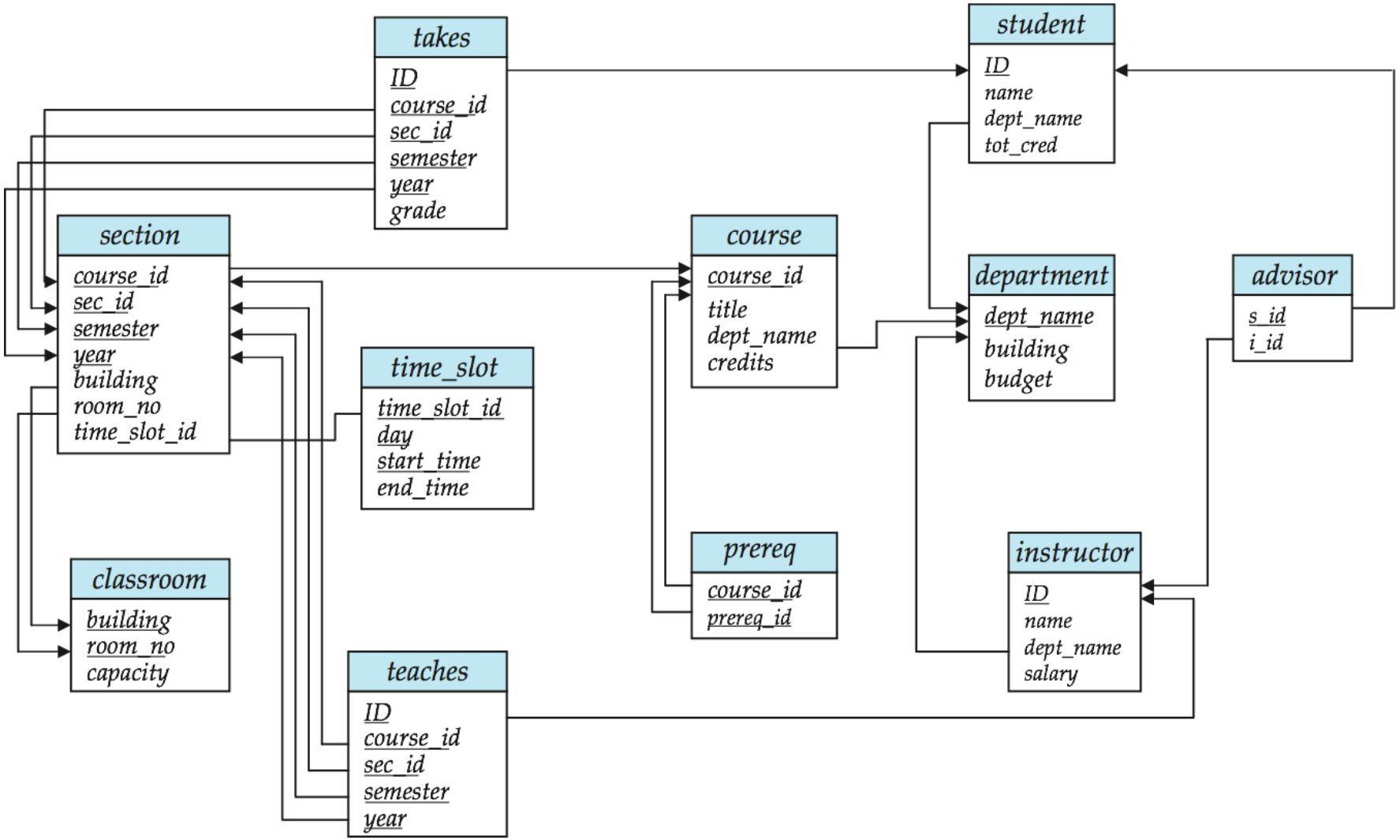
automatically detect primary and link those key

- **create table** *course* (
  …
  dept_name **varchar**(20),
  foreign key (*dept_name*) **references** *department*
  on delete cascade
  . . .
)

automatic delete when primary key delete

- alternative actions to cascade: **set null**, **set default**

Primary & Foreign Keys in Schema Diagram



Authorization Specification in SQL

- The **grant** statement is used to confer authorization

  grant <privilege list>

  on <relation name or view name>

  to <user list>

- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - a role (explained later)
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- **select:** allows read access to relation
  - (or the ability to query using the view)
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *branch* relation:

grant select on *branch* **to** U_1, U_2, U_3

table target users

- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges
- **index:** allows creation and deletion of indices.
- **resources:** allows creation of new relations.
- **alteration:** allows addition or deletion of attributes in a relation.
- **drop:** allows deletion of relations.

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

  revoke <privilege list>

  on <relation name or view name>

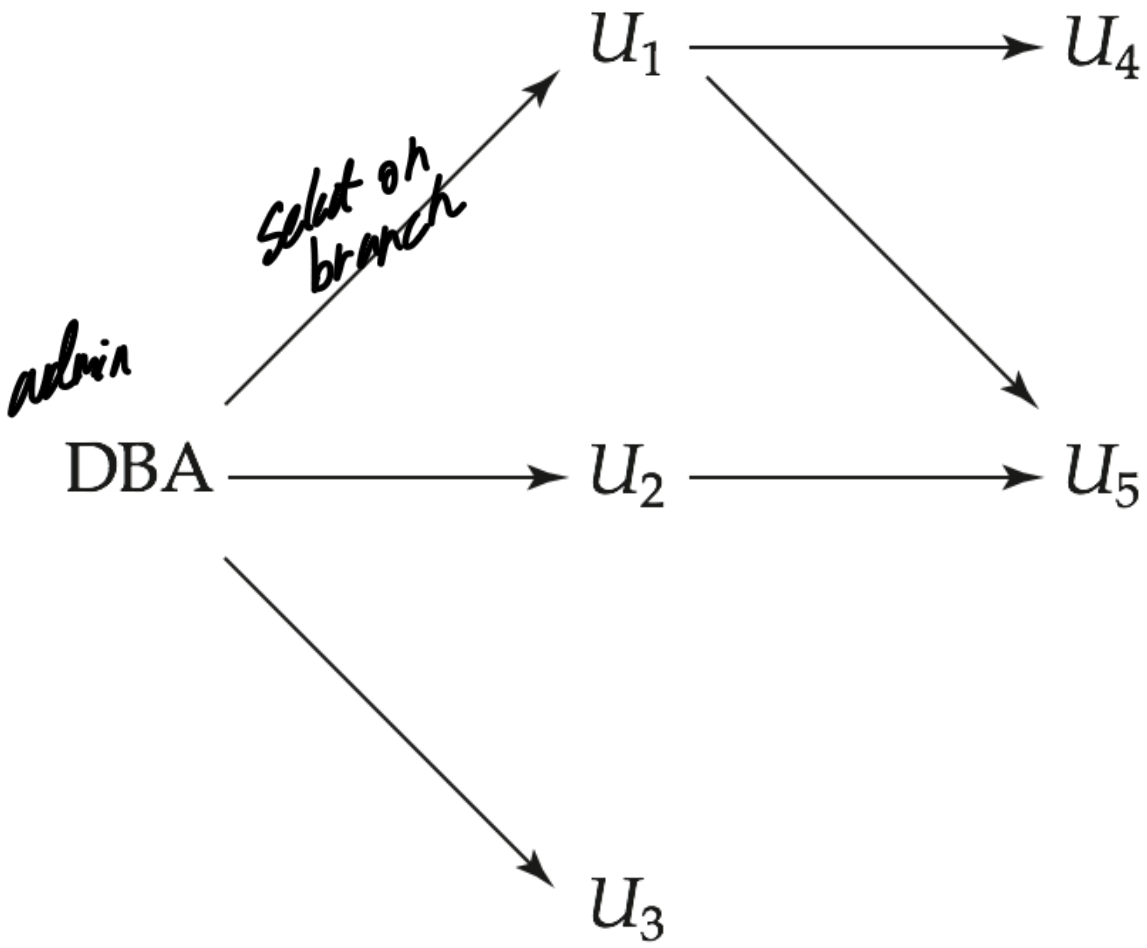
  from <user list>

  - Example:

    revoke select on *branch* **from** U_1, U_2, U_3

- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- All privileges that depend on the privilege being revoked are also revoked.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.

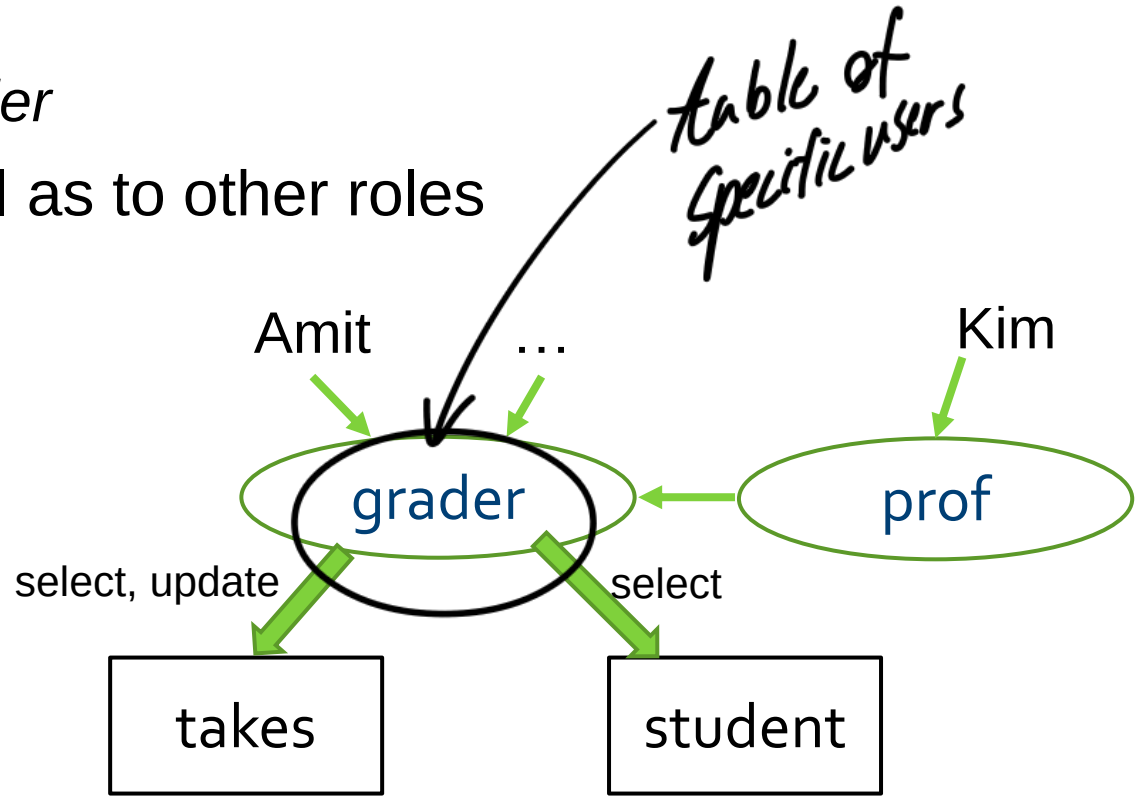
Authorization-Grant Graph



Roles

kind of authorized position

- *Roles* are used to represent a group of users and their privileges
  - **create role** *grader*;
- Privileges can be granted to roles:
  - **grant select on** *student* **to** *grader*
  - **grant select, update on** *takes* **to** *grader*
- Roles can be granted to users, as well as to other roles
  - **grant** *grader* **to** Amit;
  - **create role** *prof*;
  - **grant** *grader* **to** *prof*;
  - **grant** *prof* **to** Kim;

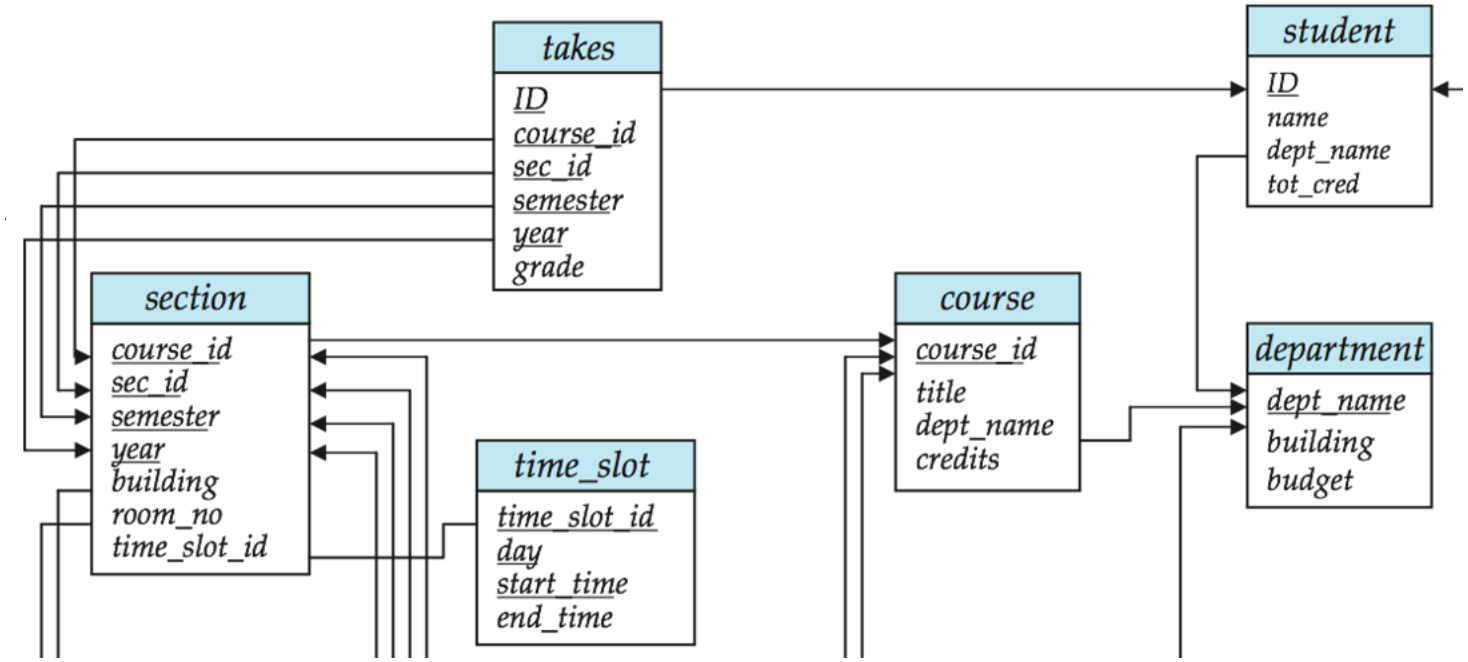


DISCUSSIONS – CHAPTER 4

Discussion 4-1

Write the following query in SQL using the **join** operator.

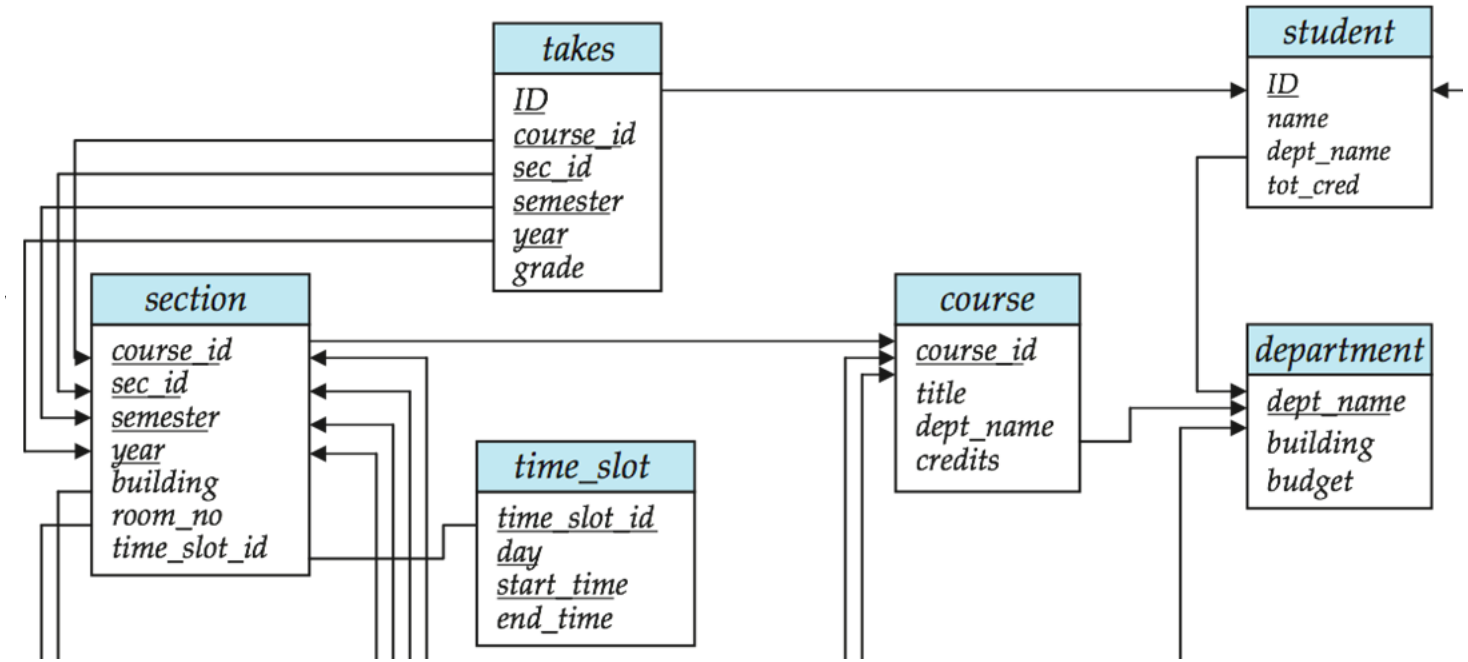
Find students (ID & name) who took a course in 2017.



Discussion 4-2

Write the following query in SQL using the **join** operator.

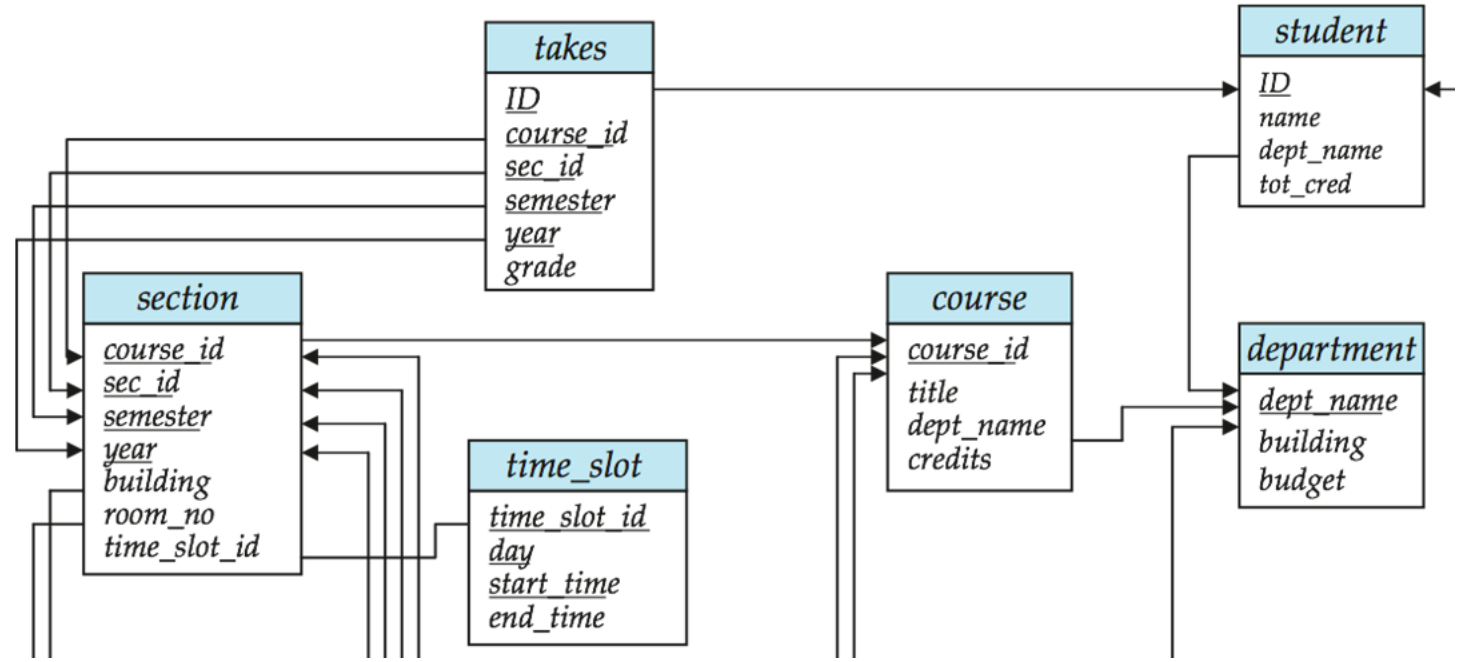
Find students (ID & name) who took a course in 2017 held in building '301'.



Discussion 4-3

Write the following query in SQL using the **join** operator.

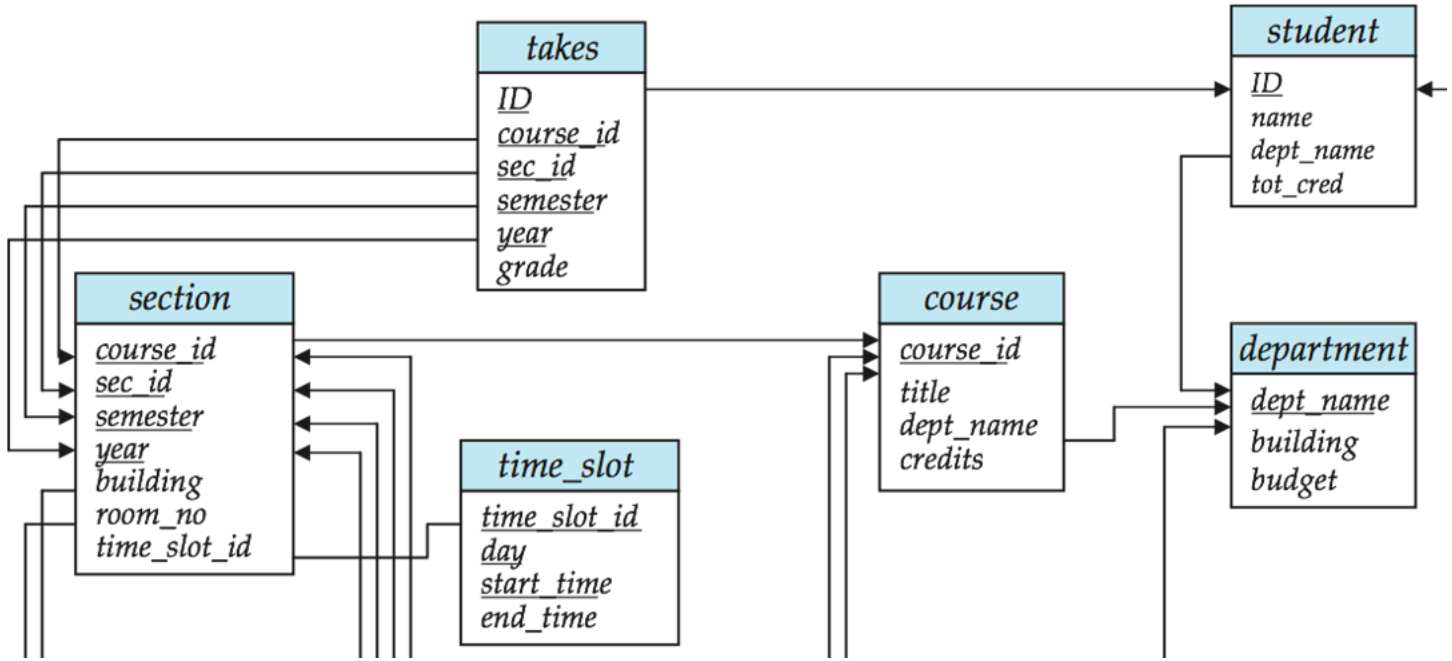
Find students (ID & name) who took a course in 2017 offered by the 'CS' department.



Discussion 4-4

Write the following query in SQL.

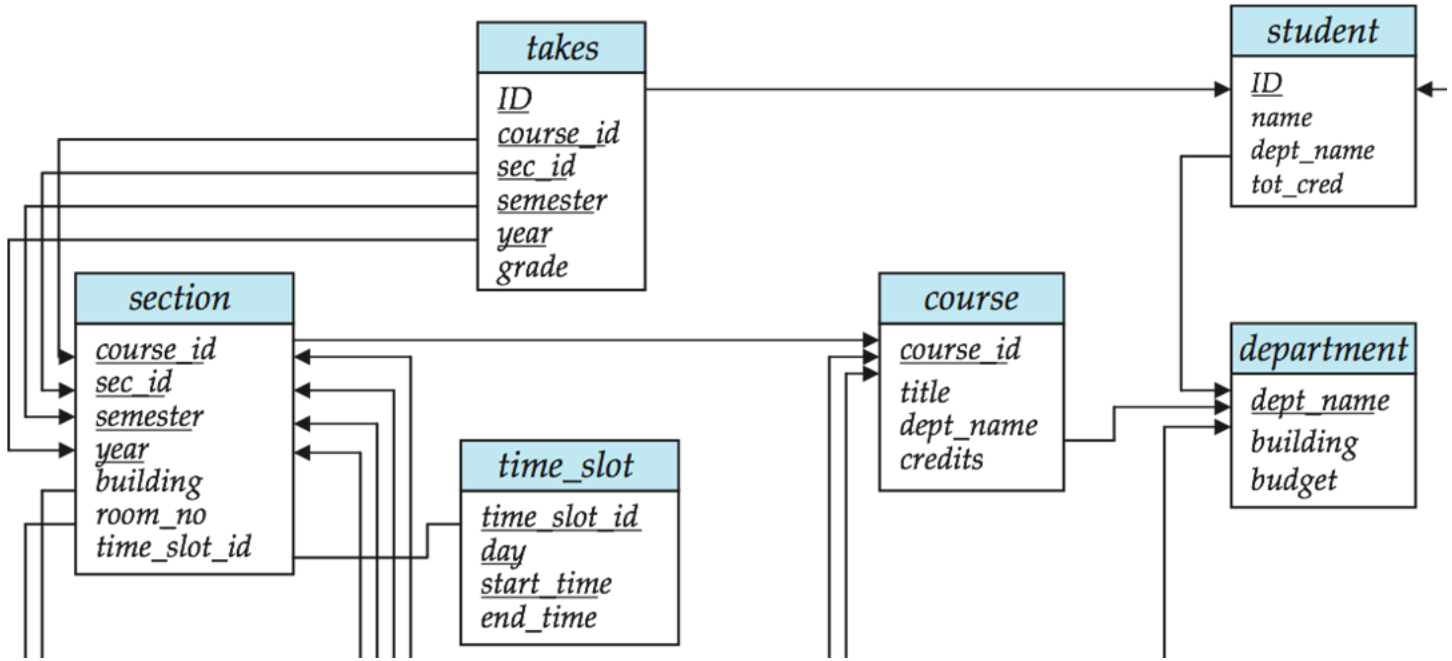
List course titles with years they were offered. Include courses that were never offered.



Discussion 4-5

Write the following query in SQL.

Update the tot_cred of each student to the sum of credits that she/he has taken with a grade other than 'F' and null.



END OF CHAPTER 4