Lic. Informatique, Univ Lille 1, 2010-11

- **Introduction aux bases de données relationnelles**
  - 7ème cours: **vues, autorisations et triggers en SQL**
- **Enseignante**: C. Kuttler
- **Biblio**: chapitres 4 et 5 de *Database Systems Concepts* de Silberschatz et al, McGraw-Hill (6ème edition, 2010)
- Ces transparents sont une adaptation de ceux disponibles sur: **www.db-book.com**

Intermediate and Advanced SQL

Database System Concepts, 6th Ed.

Chapter 4: Intermediate SQL

- **Views**
- Authorization

- Triggers (from Chap 5)

Views: virtual relations

- 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.
- Views can also be used to make it simpler to write queries. Each time a view is used in a query, it is computed.

Defining a View

- Syntax to create a new view called v

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

where <query expression> is any legal SQL expression.

- 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
  - The view definition is substituted into queries using the view.

Examples: creating and using views

- Create a view of instructors without their salary

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

- Query: find all instructors in the Biology department

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

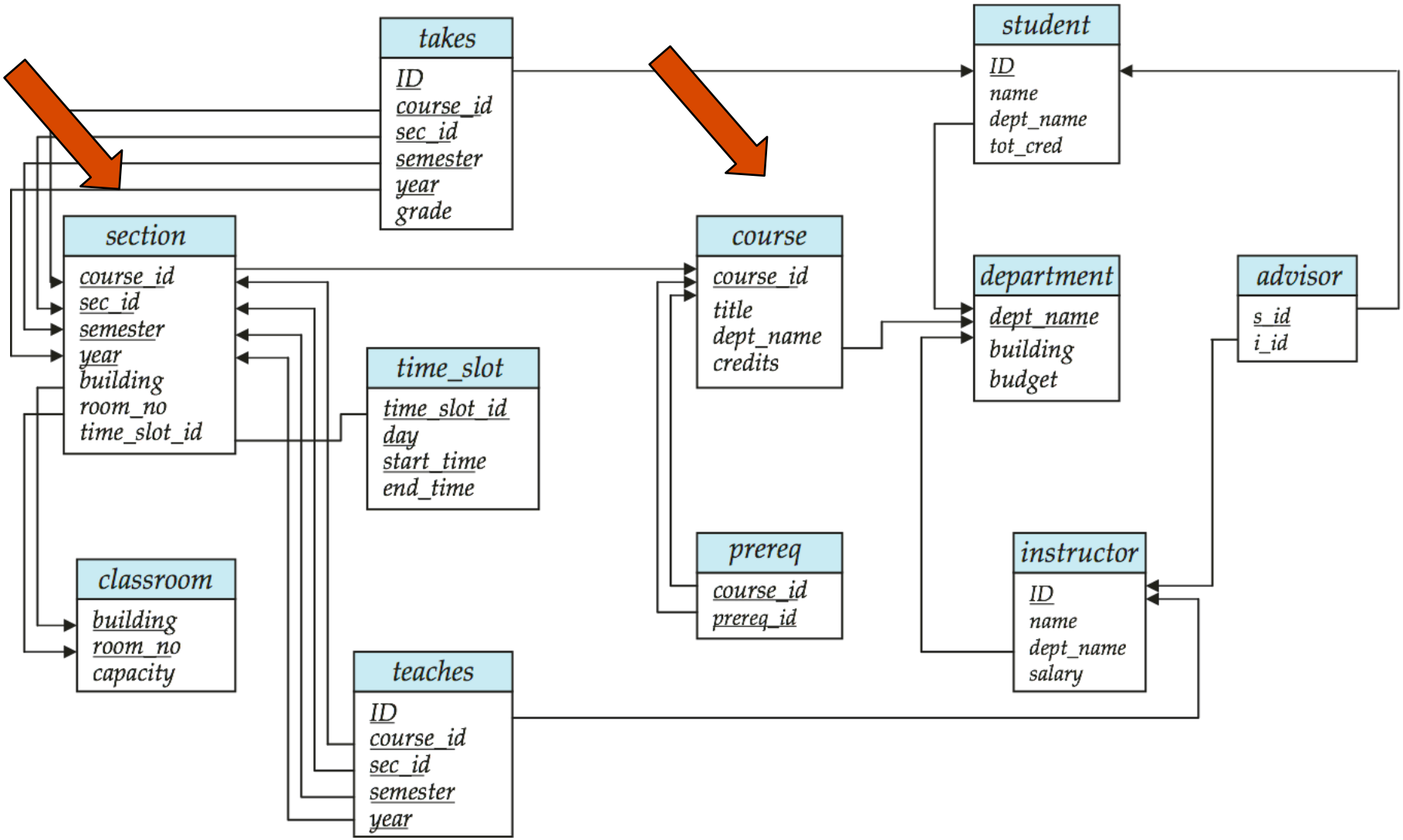
- Create a view of total salary per department. Specify attribute names in view definition.

  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

- 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 to 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.

Physics courses in Fall 2009



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';

View Expansion

- A views can't be not computed and stored, because it may become out of date if the relations used to defined it are modified.
- When a view appears in a query, it is replaced by the stored query expression.
- 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.

View Expansion

- Expand use of a view in a query/another view
- Our previous example:

```
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';
```

Exo:

- Quelles sont des conditions de jointure que vous avez fréquemment du réécrire en TP sur la base INFOTOUR?
- Proposez des vues qui pourraient être utilisées pour simplifier des requêtes.

Update of a View

- Keeping views up-to-date is a delicate issue!
- 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 following tuple into the *instructor* relation (with unknown salary!)

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

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 should we add, if multiple departments in Taylor?
  - what if no department is in Taylor?
  - …

Updatable views

- Most SQL implementations allow updates, inserts and deletes 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.
- Beyond this: advanced topic!

Some Updates can not be translated at all

- **create view** *history_instructors* **as**
  select *
  from *instructor*
  where *dept_name*= ’History’;
- Insert (’25566’, ’Brown’, ’Biology’, 100000) into *history_instructors*

Exo

- Quelles des vues proposées sur INFOTOUR sont *simples,* et permettent donc des mises à jour sans problèmes?
- Quelles vues devraient être restreintes à l'utilisation dans des requêtes, sans permettre des mises à jour?

Chapter 4: Intermediate SQL

- Join Expressions
- Integrity Constraints
- Views
- **Authorization**

Authorization

Forms of authorization on parts of the database:

- **Read** - allows reading, but not modification of data.
- **Insert** - allows insertion of new data, but not modification of existing data.
- **Update** - allows modification, but not deletion of data.
- **Delete** - allows deletion of data.

Forms of authorization to modify the database schema

- **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.

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 (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- 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 *instructor* **to** U_1, U_2, U_3
- **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.

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.
- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Roles

- **create role** *instructor*;
- Privileges can be granted to roles:
  - **grant select on** *takes* **to** *instructor*;
  - This allows instructors to read all students' information on which courses they took, and grades they obtained.
- Roles can be granted to users, as well as to other roles
  - **create** role *student*
  - **grant** *instructor* **to** Amit;
  - **create role** *dean*;
  - **grant** *instructor* **to** *dean*;
  - **grant** *dean* **to** Satoshi;

Authorization on Views

- **create view** *geo_instructor* **as**
  (**select** *
  from *instructor*
  where *dept_name* = 'Geology');
- **grant select on** *geo_instructor* **to** *staff*
- Suppose that a staff member issues
  - **select** *
    from *geo_instructor*;
- What if
  - staff does not have permissions on *instructor*?
  - creator of view did not have some permissions on *instructor*?

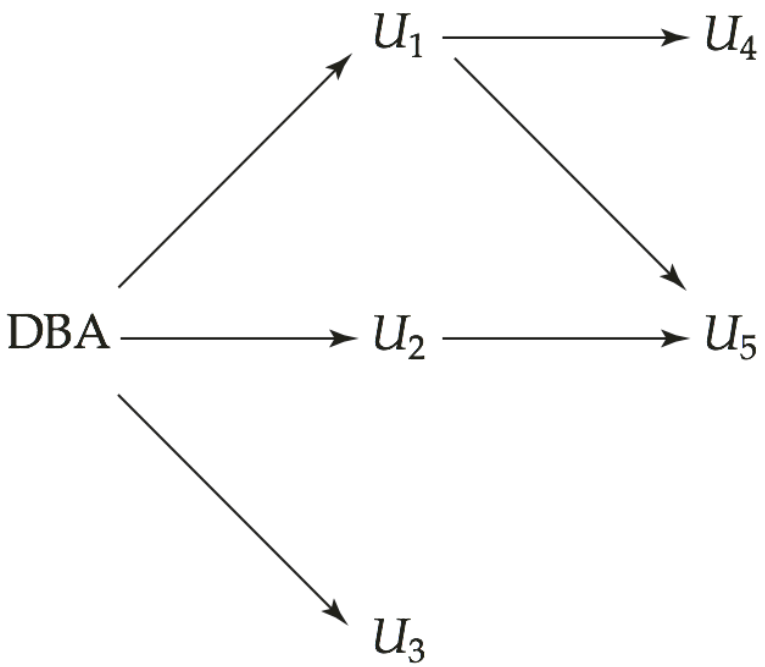
Grant reference

- **references** privilege to create foreign key
  - **grant reference** (*dept_name*) **on** *department* **to** Mariano;

- Why is this required?
  - User Michel creates a foreign key in a relation r, that references to department.dept_name
  - User Michel inserts a tuple into r, for instance, about the Geology deparment (that didn't previously exist)
  - If this tuple is later deleted, the "mother" relation is modified. This restricts the future activity of other users!

Transfer of privileges

- transfer of privileges
  - **grant select on** *department* **to** Amit **with grant option**;
  - **revoke select on** *department* **from** Amit, Satoshi **cascade**;
  - **revoke select on** *department* **from** Amit, Satoshi **restrict**;
- Etc.



End of Chapter 4

Chapter 5: Advanced SQL - triggers

Database System Concepts, 6th Ed.

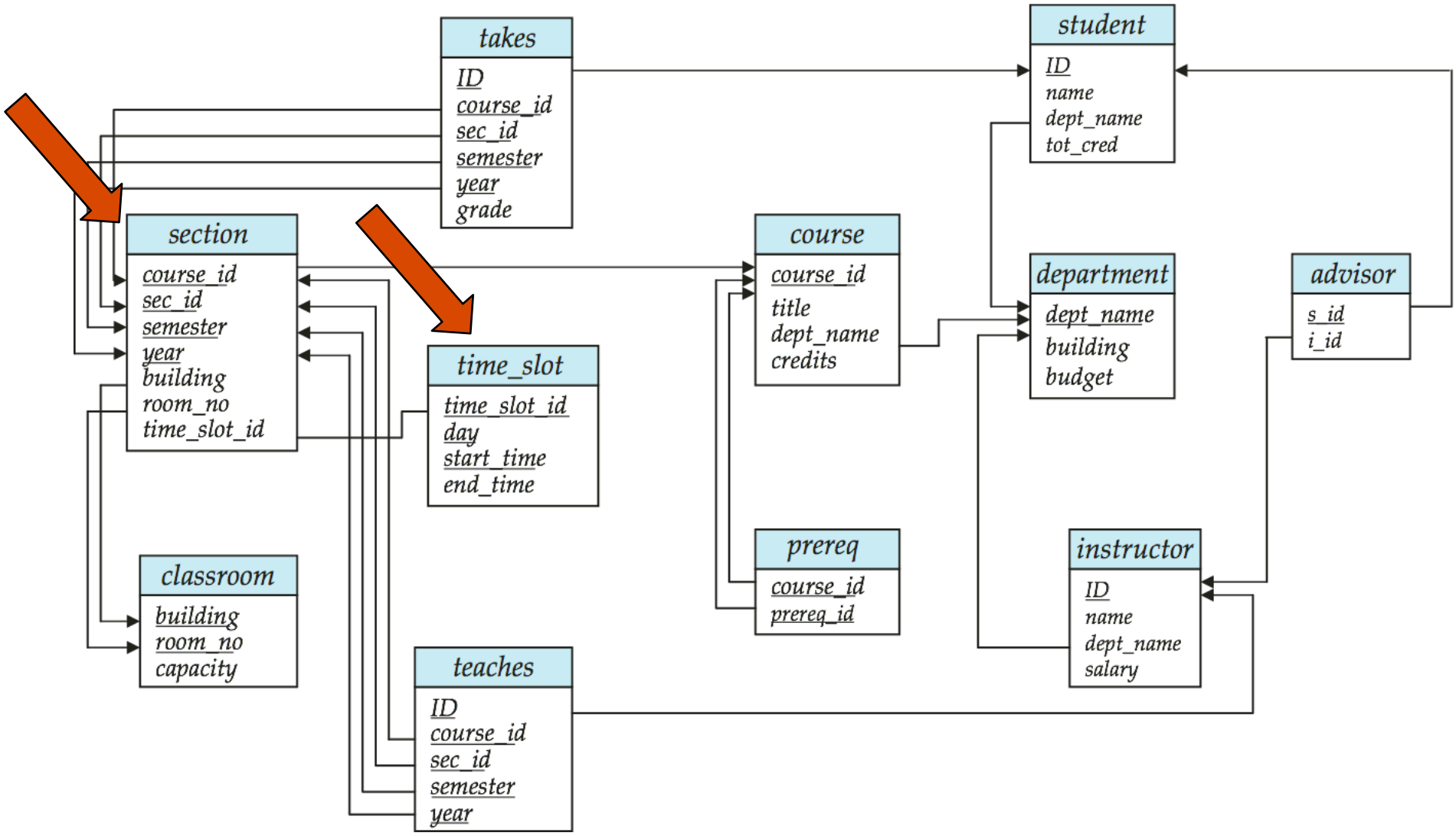
Triggers: event-condition-action

- **Triggers allow to maintain database in a coherent state, beyond what key constraints allow.**

- **Event**: database modification, such as insert, delete, update
- **Condition**: any true/false expression
- **Action**: Sequence of SQL statements that will automatically be executed.

- Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.
  - Here, we use this idealized syntax to explain the main principles.
  - Check the lab manual for Oracle Syntax!

University Database: inserting a new section

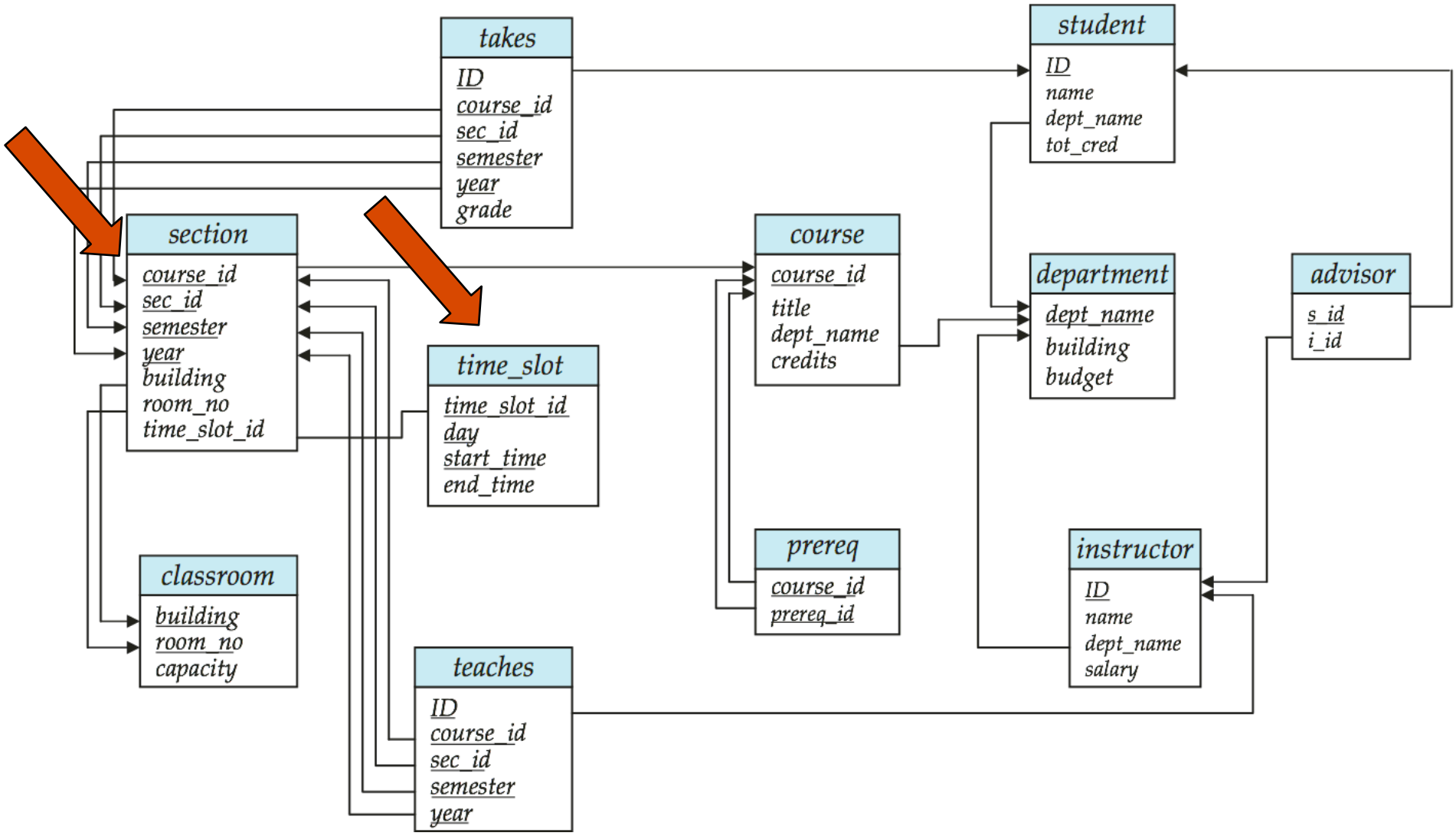


Trigger 1: ensure that time slot exists

- E.g. *time_slot_id* is not a primary key of *timeslot,* so we cannot create a foreign key constraint from *section* to *timeslot.*
- Alternative: use triggers on *section* and *timeslot* to enforce integrity constraints

```
create trigger timeslot_check1

after insert on section
referencing new row as nrow
for each row
when

        /* time_slot_id not present in time_slot */
        (nrow.time_slot_id  not in

                (select time_slot_id from time_slot) )

begin
    rollback
end;
```

University Database: deleting a time_slot



Trigger 2: don't delete a time_slot that is still used!

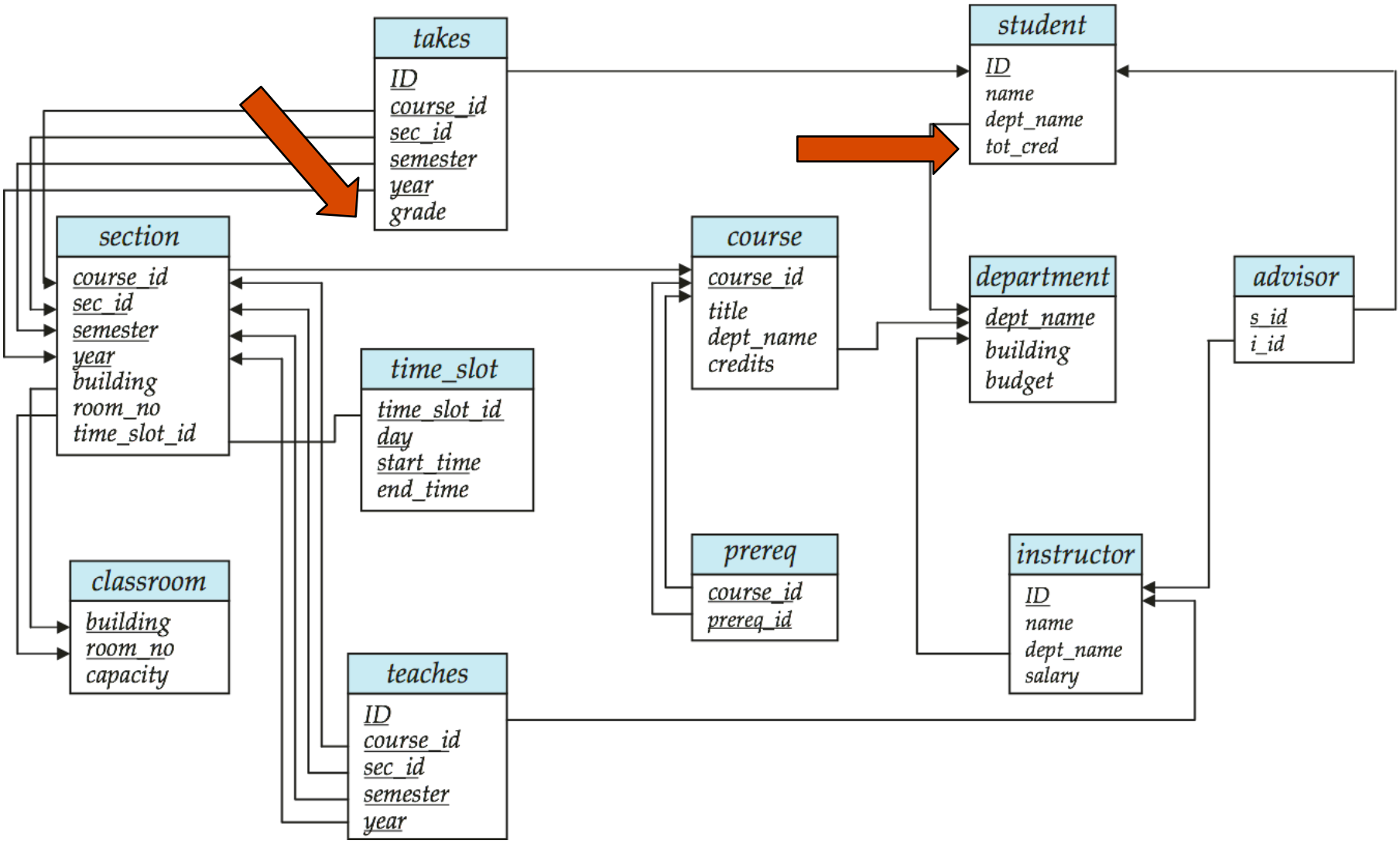
```
create trigger timeslot_check2 after delete on time_slot
    referencing old row as orow
    for each row
    when ( /*   time_slot_id has just been deleted from time slot */

            orow.time_slot_id  not in (

                    select time_slot_id from time_slot)


          /* and time_slot_id still referenced from section*/
           and orow.time_slot_id in (
              select time_slot_id     from section)

            )
    begin
       rollback
    end;
```

Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - **E.g., after update of** *takes* **on** *grade*
- Values of attributes before and after an update can be referenced
  - **referencing old row as** : for deletes and updates
  - **referencing new row as** : for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. E.g. convert blank grades to null.

```
create trigger setnull_trigger before update of takes
referencing new row as nrow
for each row
when (nrow.grade = ' ')
 begin atomic
        set nrow.grade = null;
 end;
```

At the end of the term, a student has passed an exam!



Trigger 3: increase the lucky student's tot_credit value

```
create trigger credits_earned after update of takes on (grade)
referencing new row as nrow
referencing old row as orow
for each row
when nrow.grade <> 'F' and nrow.grade is not null
   and (orow.grade = 'F' or orow.grade is null)
begin atomic
    update student
    set tot_cred= tot_cred +
        (select credits
         from course
         where course.course_id= nrow.course_id)
    where student.id = nrow.id;
end;
```

When Not To Use Triggers

- Triggers were used earlier for tasks such as
  - maintaining summary data (e.g., total salary of each department)
  - Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
  - Databases today provide built in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger
- Risk of unintended execution of triggers, for example, when
  - loading data from a backup copy
  - replicating updates at a remote site
  - Trigger execution can be disabled before such actions.
- Other risks with triggers:

Seen today

- Views
- Authorization
- Triggers