Views Defined Using Other Views

- create view *physics_fall_2009* as
  select *course.course_id*, course.title, *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*, title, *room_number*
  from *physics_fall_2009*
  where *building*= ’Watson’;

View Expansion

- Expand use of a view in a query/another view.
- For the previous example:

```
create view physics_fall_2009_watson as
(select course_id, room_number
from (select course.course_id, course.title,
             section, 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
  - Or it may depend by a path indirectly
- A view relation v is said to be *recursive* if it depends on itself.

Update of a View

- Add a new tuple to *faculty* view which we defined earlier on instructor (without salary)

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

This leads to actual insertion in instructor table with salary as null :

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

Some Updates cannot be Translated Uniquely

- create view *instructor_info* as
  select *ID*, *name*, *building*
  from *instructor A*, *department B*
  where *A.dept_name*= *B.dept_name*;

insert into *instructor_info* values ('69987', 'Smith', 'Watson');

- which department, if multiple departments in Watson?
- what if no department is in Watson?

- 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

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- If relations used in the query are updated, the materialized view result becomes out of date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.

Integrity Constraints

- Integrity constraints guard against incorrect, inconsistent data
- Defined as per business needs
  - A checking account must have a balance greater than 10,000.00
  - A salary of a bank employee must be at least 25000
  - A customer must have a (non-null) phone number

Integrity Constraints on a Single Relation

- not null
- primary key, foreign key
- unique
- check (P), where P is a predicate

Unique Constraints

- unique (A_1, A_2, …, A_m)
    - it states that the attributes *A*1, *A*2, … *A*m form a candidate key.
    - Candidate keys are permitted to be null (in contrast to primary keys).

The check clause

check (P)

where P is a predicate

Example: semester is one of fall, winter, spring or summer:

```
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’ ))
);
```

Referential Integrity

- Ensures that a value of an attribute that appears in one relation also appears as value for an attribute in another relation
- What if we update the value in the target table, or delete it ?
  - Department name (CSE) referred in course is updated (CS)
  - The CSE department is closed !
  - We may want to propagate these to referencing table too

Cascading Actions in Referential Integrity

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

- create table *course* (
  …
  dept_name varchar(20),
  foreign key (*dept_name*) references *department*
          on delete cascade
          on update cascade,
   . . .)
- alternative actions to cascade: set null, set default

Integrity Constraint Violation During Transactions

- E.g.

```
create table person (
    ID  char(10),
    name char(40),
    mother char(10),
    father  char(10),
    primary key ID,
    foreign key father references person,
    foreign key mother references  person)
```

- How to insert a tuple without causing constraint violation ?
  - insert father and mother of a person before inserting person
  - OR, set father and mother to null initially, update after inserting all persons (not possible if father and mother attributes declared to be not null)
  - OR defer constraint checking