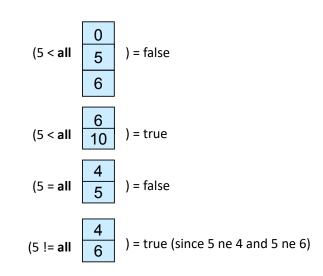
## Outline

- Overview of modeling
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Basic Data Definition (3.2)
  - Basic Queries (3.3-3.5)
  - Joins
  - Null values (3.6)
  - Aggregates (3.7)
  - Other

## Definition of all clause

• F <comp> all  $r \Leftrightarrow \forall t \in r$  (F <comp> t)



## Example Query

 Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

#### 118

## Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- exists  $r \Leftrightarrow r \neq \emptyset$
- not exists  $r \Leftrightarrow r = \emptyset$

## **Correlated Subqueries**

• Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

select course\_id
from section F
where semester = 'Fall' and year= 2009 and
 exists (select \*
 from section S
 where semester = 'Spring' and year= 2010
 and F.course\_id = S.course\_id);

Correlation name or correlation variable

#### 120

## Not Exists

• Find all students who have taken all courses offered in the Biology department.

Note that  $X - Y = \emptyset$  means  $X \subseteq Y$ 

## Test for Absence of Duplicate Tuples

Find all courses that were offered exactly once in 2009:

WRONG: unique is used to define constraints at table creation.

#### **RIGHT:**

122

## **Derived Relations**

- Subqueries can even be used in the from clause
- Find the average instructors' salaries of those departments where the average salary is greater than \$42,000."
   select dept\_name, avg\_salary

from (select dept\_name, avg (salary) as avg\_salary
 from instructor
 group by dept\_name)

where avg\_salary > 42000;

Note that the following is equivalent:

select dept\_name, avg (salary) as avg\_salary
from instructor
group by dept\_name
having avg(salary) > 42000;

### Views

- Might not want all users to see the entire logical model (that is, all the actual relations stored in the database.)
- A person who needs to know an instructor's name and department might not need to know 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

<ul> <li>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</li> </ul>
and course.dept_name = 'Physics' and section.semester = 'Fall' and section.year = '2009';
<ul> <li>create view physics_fall_2009_watson as select course_id, room_number from physics_fall_2009 where building= 'Watson';</li> </ul>

## Views Defined Using Other Views

create view physics\_fall\_2009\_watson as
 select course\_id, room\_number
 from physics\_fall\_2009
 where building= 'Watson';

Effect is the following:

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';

128

## **View Expansion**

A view may be used to define 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<sub>1</sub> is said to *depend on* view relation v<sub>2</sub> if either v<sub>1</sub> depends directly to v<sub>2</sub> or there is a path of dependencies from v<sub>1</sub> to v<sub>2</sub>
- A view relation *v* is said to be *recursive* if it depends on itself.

## **View Expansion**

- A way to interpret queries w/ views...
  - Let view v, be defined by an expression e, that may itself contain uses of view relations.
  - View expansion of an expression *e* repeats the following replacement step:

#### repeat

Find any view relation  $v_i$  in e

Replace the view relation  $v_i$  by expression  $e_i$ 

until no more view relations are present in e

• As long as the view definitions are not recursive, this loop will terminate.

#### 130

# Update of (through) 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) - Constant Statement State

into the *instructor* relation.

## Some Updates Do Not Translate 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';
- Insert ('25566', 'Brown', 'Biology', 100000) into history\_instructors

## Summary

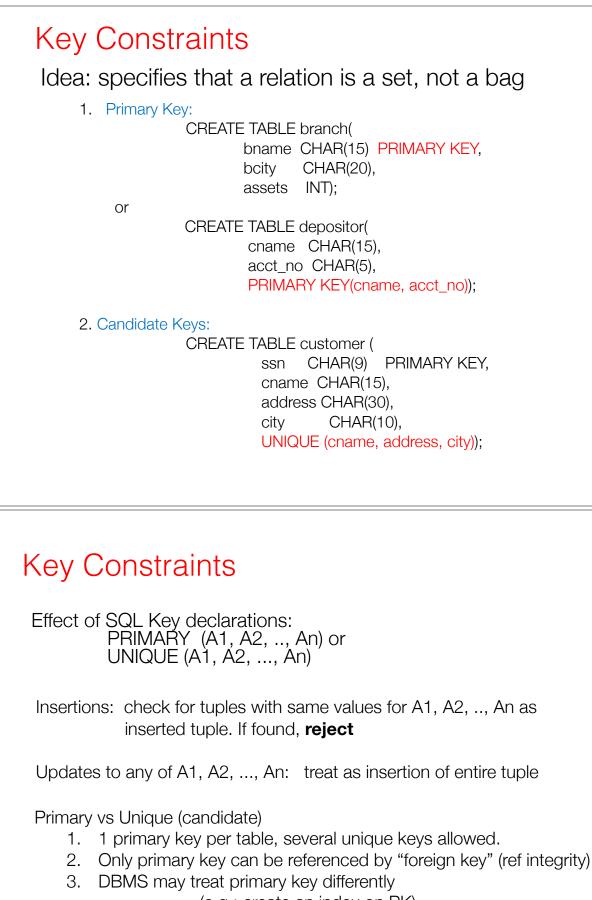
- Relational Model (Chapter 2)
  - Basics
  - Keys
  - Relational operations
  - Relational algebra basics
- SQL (Chapter 3)
  - Setting up the PostgreSQL database
  - Data Definition (3.2)
  - Basics (3.3-3.5)
  - Null values (3.6)
  - Aggregates (3.7)
  - Advanced operators

# Integrity Constraints

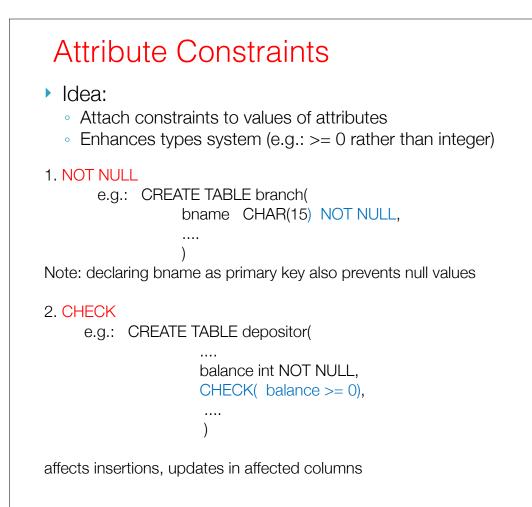
- Predicates on the database
- Must always be true (checked whenever db gets updated)

#### There are 4 types of IC's:

- Key constraints (1 table)
  - e.g., 2 accts can't share the same acct\_no
- Attribute constraints (1 table)
  - e.g., accts must have nonnegative balance
- Referential Integrity constraints (2 tables)
   E.g. *bnames* associated w/ *loans* must exist
- Global Constraints (n tables)
  - E.g., all *loans* must be carried by at least 1 *customer* with a savings acct



(e.g.: create an index on PK)



138

#### Attribute Constraints

#### Domains:

Attrib	oute Constraints	
Assoc	iating constraints with domains:	
	avoid repeating specification of same constraint multiple columns	
	name constraints CREATE DOMAIN bank-balance INT ( CONSTRAINT not-overdrawn CHECK (value >= 0), CONSTRAINT not-null-value CHECK (value NOT NUL	LL));
AL	ges: n add or remove: .TER DOMAIN bank-balance ADD CONSTRAINT capped CHECK( value <= ort better errors (know which constraint violated)	10000)