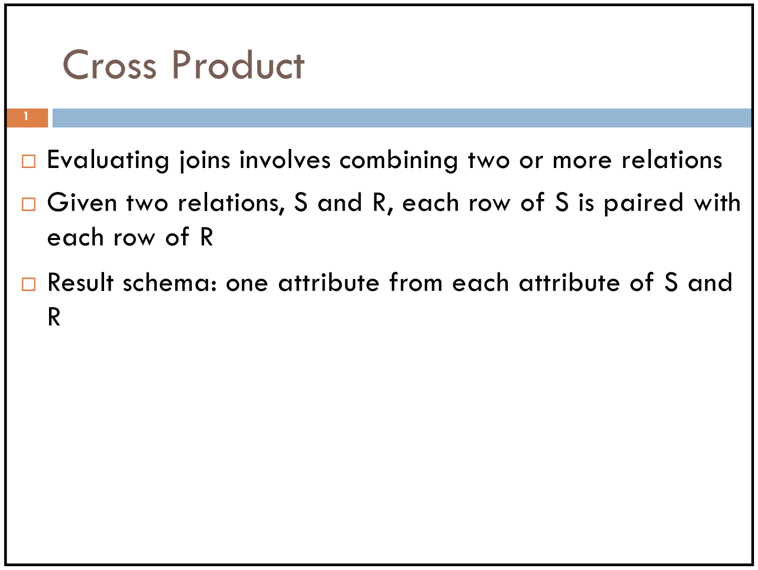
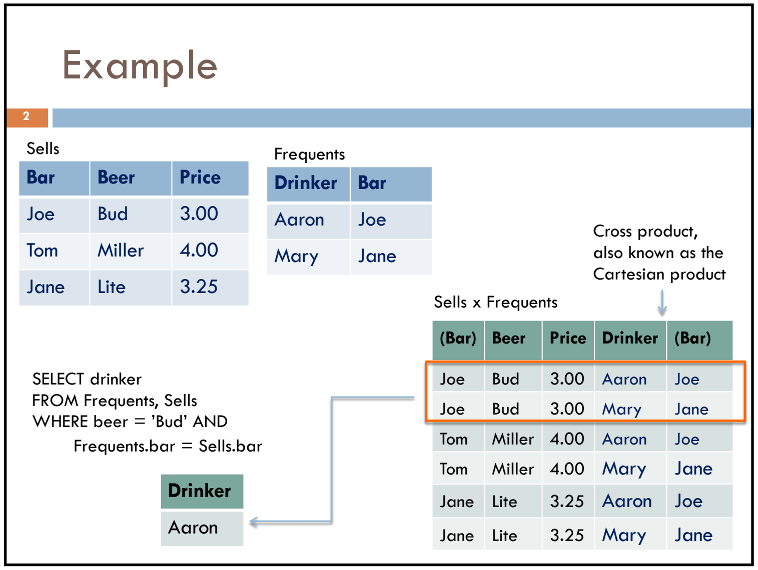


Cross Product

1

- □ Evaluating joins involves combining two or more relations
- □ Given two relations, S and R, each row of S is paired with each row of R
- □ Result schema: one attribute from each attribute of S and R

1



Example

2

Sells

Bar	Beer	Price
Joe	Bud	3.00
Tom	Miller	4.00
Jane	Lite	3.25

Frequents

Drinker	Bar
Aaron	Joe
Mary	Jane

Cross product, also known as the Cartesian product

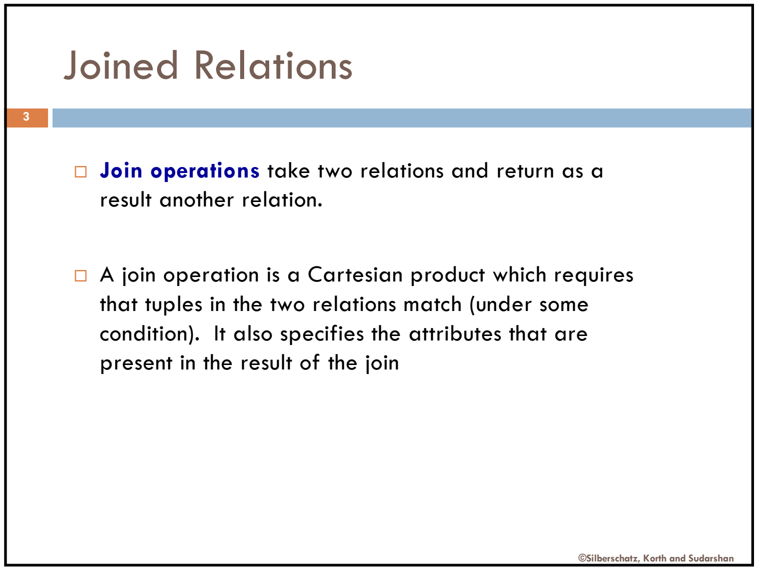
Sells x Frequents

(Bar)	Beer	Price	Drinker	(Bar)
Joe	Bud	3.00	Aaron	Joe
Joe	Bud	3.00	Mary	Jane
Tom	Miller	4.00	Aaron	Joe
Tom	Miller	4.00	Mary	Jane
Jane	Lite	3.25	Aaron	Joe
Jane	Lite	3.25	Mary	Jane

```
SELECT drinker
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
```

Drinker
Aaron

2

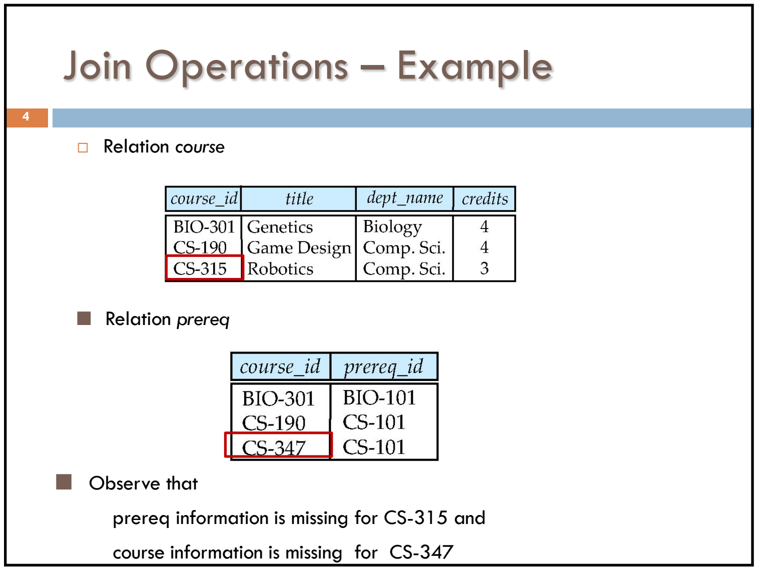


Joined Relations

3

- □ **Join operations** take two relations and return as a result another relation.
- □ A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join

3



Join Operations – Example

4

- □ Relation *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

- Relation *prereq*

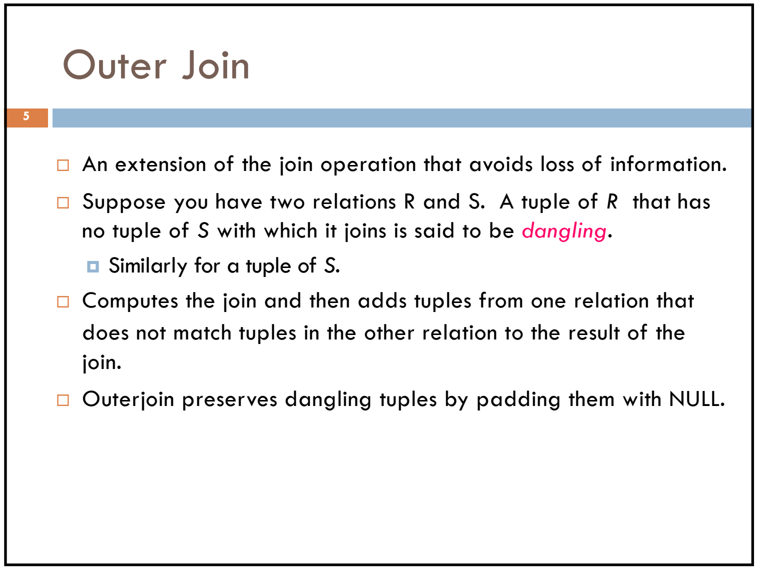
course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  prereq information is missing for CS-315 and

  course information is missing for CS-347

4

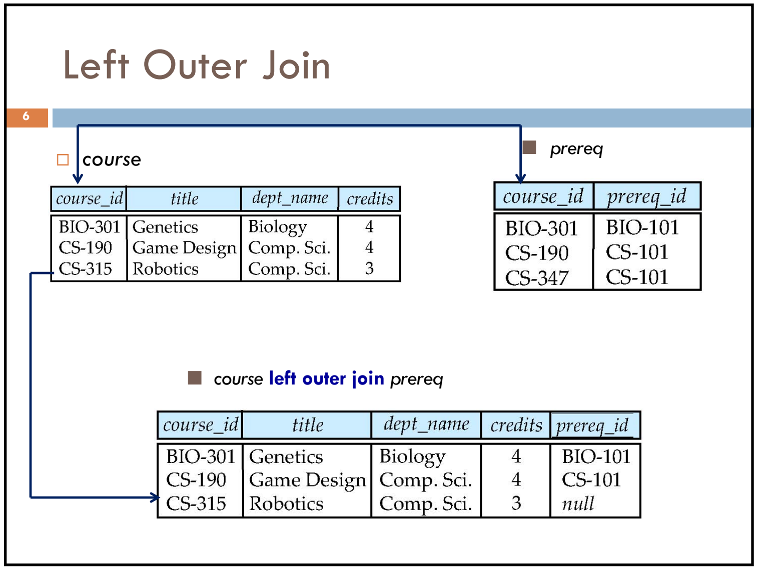


Outer Join

5

- An extension of the join operation that avoids loss of information.
- Suppose you have two relations R and S. A tuple of *R* that has no tuple of *S* with which it joins is said to be *dangling*.
  - Similarly for a tuple of *S*.
- 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.
- Outerjoin preserves dangling tuples by padding them with NULL.

5



Left Outer Join

6

- 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

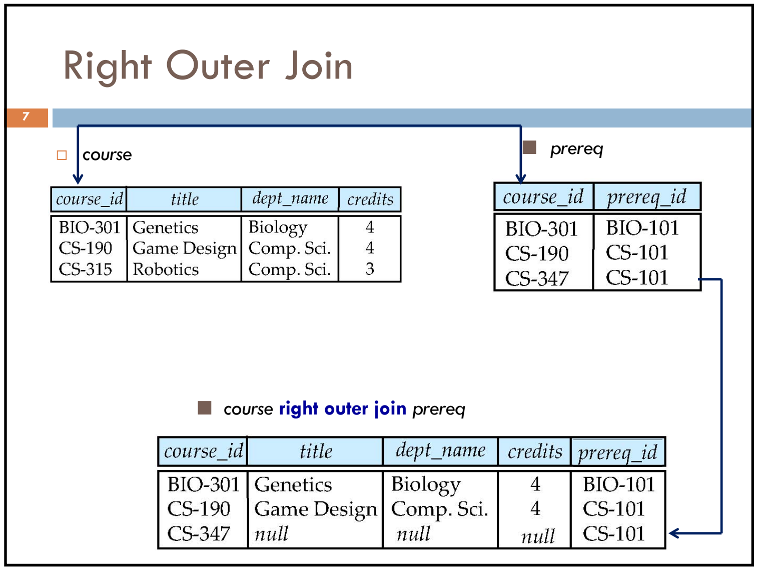
- *prereq*

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

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

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

6



Right Outer Join

7

- 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

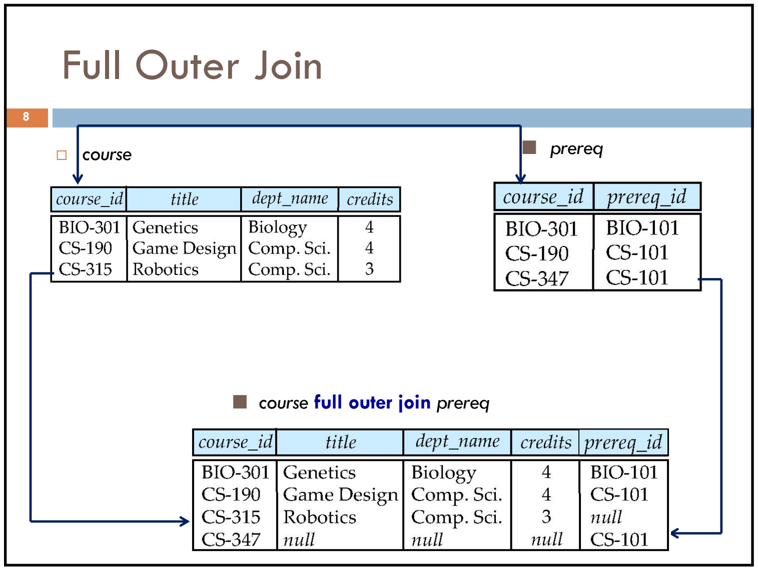
- *prereq*

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

- *course* **right outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

7



Full Outer Join

8

- 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* **full outer join** *prereq*

course_id	*title*	*dept_name*	*credits*	*prereq_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

8

Inner Join

9

- *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* **inner join** *prereq* **on**
  course.course_id = prereq.course_id

course_id	*title*	*dept_name*	*credits*	*prereq_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101

9

Outerjoins

10

- R OUTER JOIN S is the core of an outerjoin expression. It is modified by:
  1. Optional NATURAL in front of OUTER.
     - Check equality on all common attributes
     - No two attributes with the same name in the output
  2. Optional ON <condition> after JOIN.
  3. Optional LEFT, RIGHT, or FULL before OUTER.
     - LEFT = pad dangling tuples of R only.
     - RIGHT = pad dangling tuples of S only.
     - FULL = pad both; this choice is the default.

Credit: Renee J. Miller

10

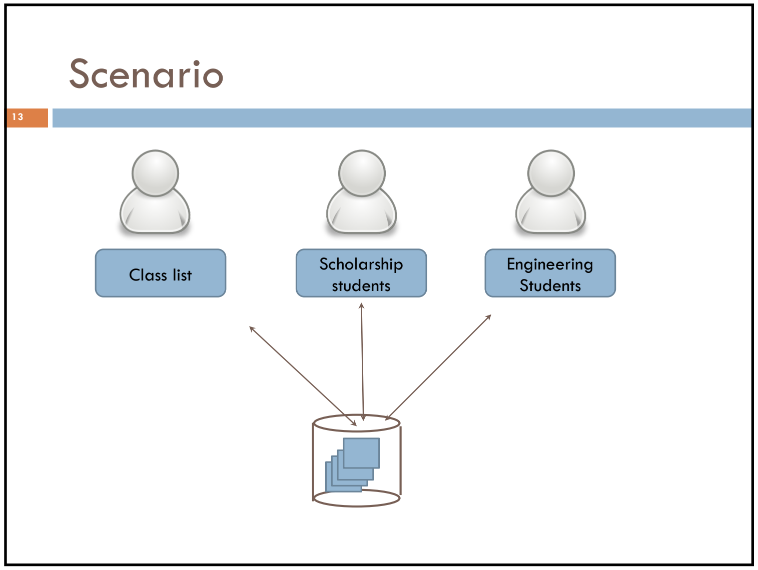
Class Example

11

11

VIEWS

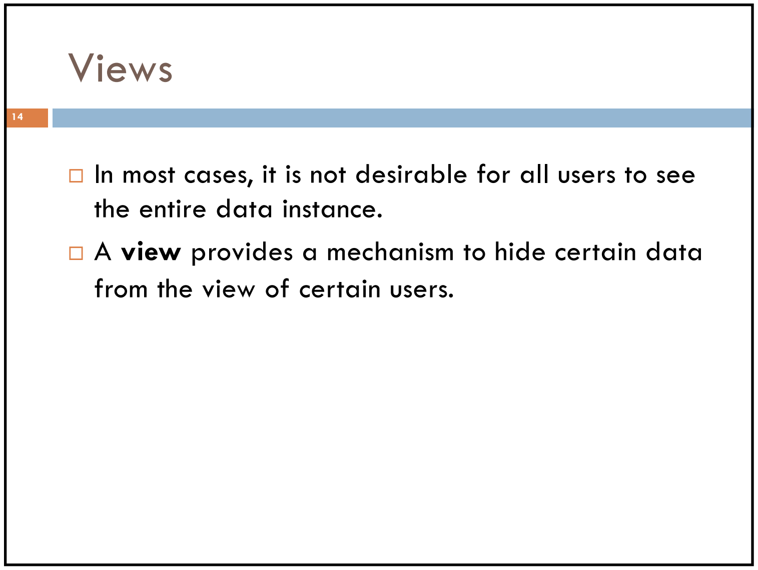
12



Scenario

13

13

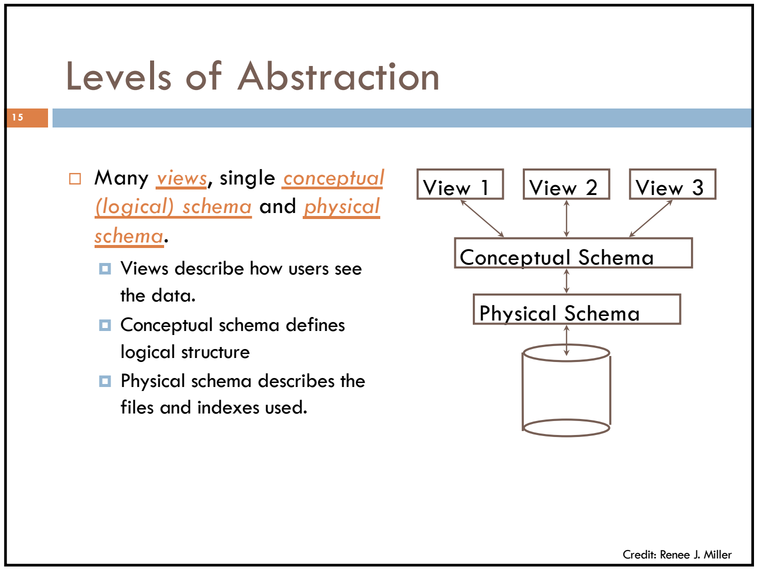


Views

14

- □ In most cases, it is not desirable for all users to see the entire data instance.
- □ A **view** provides a mechanism to hide certain data from the view of certain users.

14



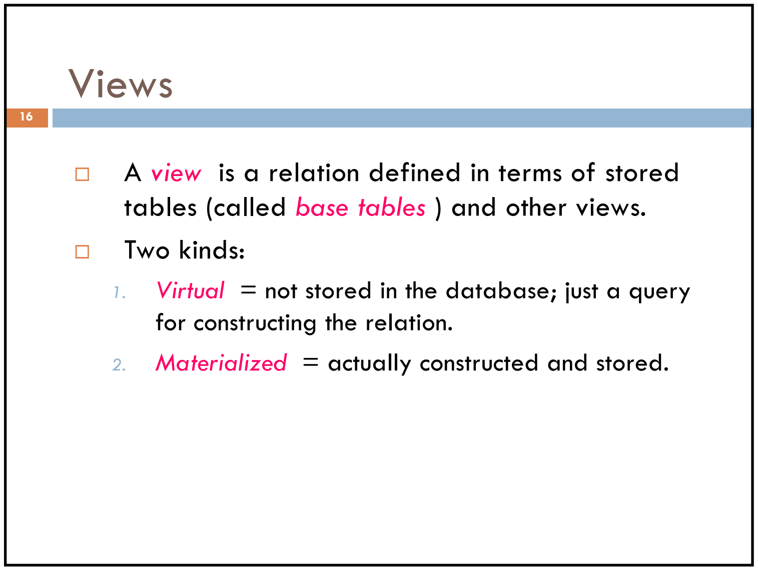
Levels of Abstraction

15

- □ Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.

Credit: Renee J. Miller

15

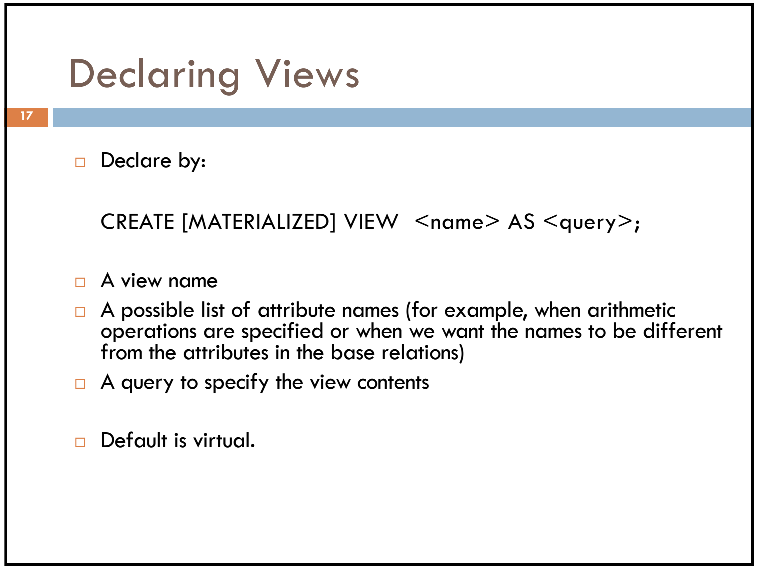


Views

16

- □ A *view* is a relation defined in terms of stored tables (called *base tables*) and other views.
- □ Two kinds:
  1. *Virtual* = not stored in the database; just a query for constructing the relation.
  2. *Materialized* = actually constructed and stored.

16



Declaring Views

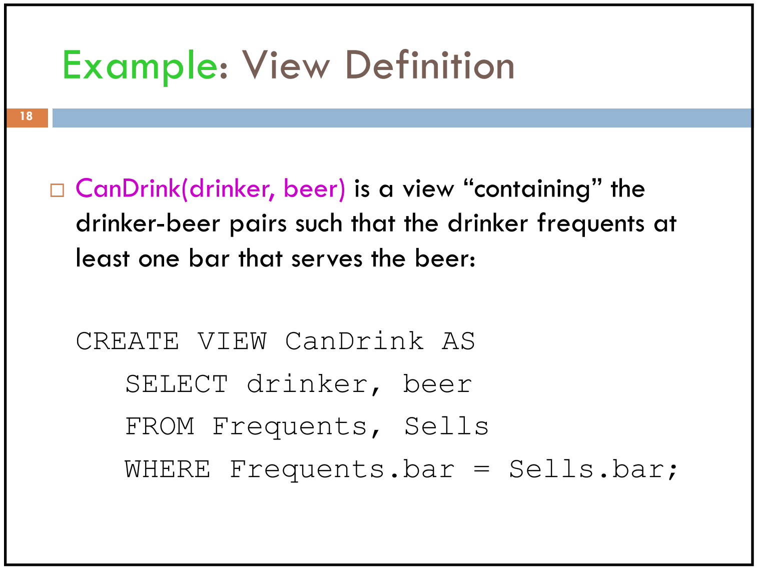
- Declare by:

```
CREATE [MATERIALIZED] VIEW  <name> AS <query>;
```

- A view name
- A possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
- A query to specify the view contents

- Default is virtual.

17

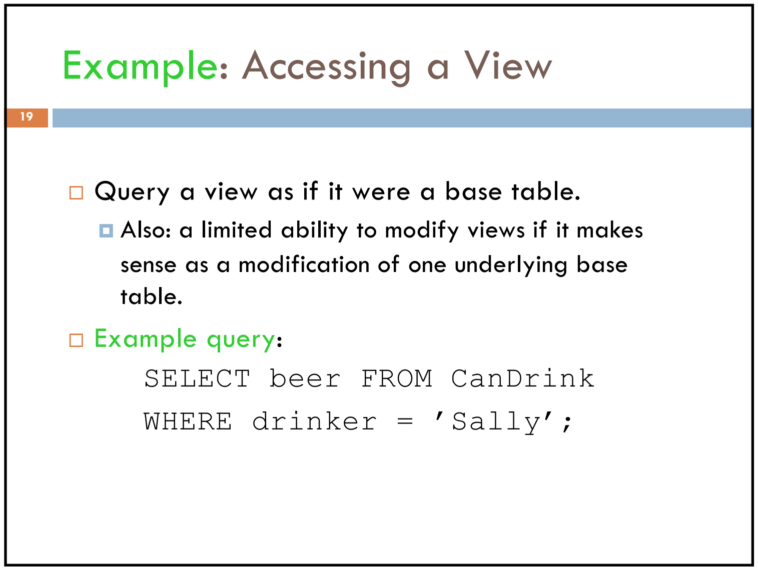


Example: View Definition

- CanDrink(drinker, beer) is a view “containing” the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
   SELECT drinker, beer
   FROM Frequents, Sells
   WHERE Frequents.bar = Sells.bar;
```

18

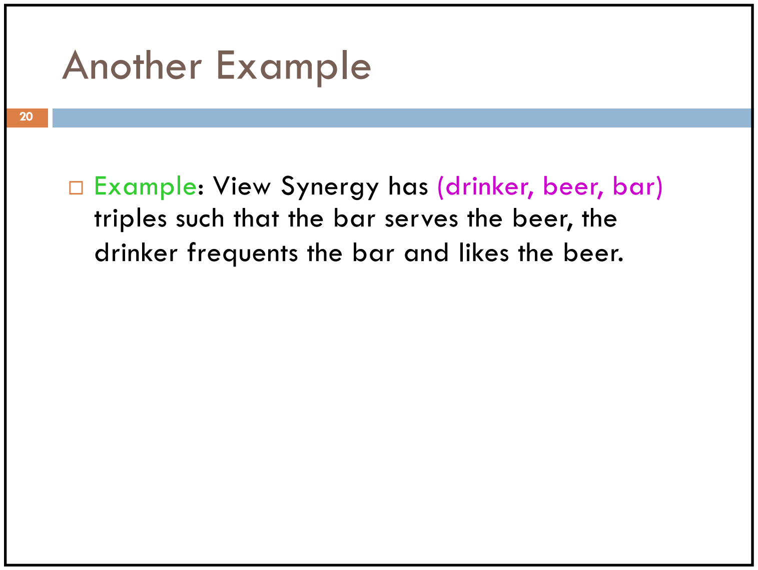


Example: Accessing a View

- Query a view as if it were a base table.
  - Also: a limited ability to modify views if it makes sense as a modification of one underlying base table.
- Example query:

```
SELECT beer FROM CanDrink
WHERE drinker = ’Sally’;
```

19



Another Example

- Example: View Synergy has (drinker, beer, bar) triples such that the bar serves the beer, the drinker frequents the bar and likes the beer.

20

Example: The View

```
CREATE VIEW Synergy AS
  SELECT Likes.drinker, Likes.beer, Sells.bar
  FROM Likes, Sells, Frequents
  WHERE Likes.drinker = Frequents.drinker
      AND Likes.beer = Sells.beer
      AND Sells.bar = Frequents.bar;
```

Pick one copy of each attribute

Natural join of Likes, Sells, and Frequents

21

Updates on Views

- □ Generally, it is impossible to modify a virtual view, because it doesn't exist.
- □ Can't we "translate" updates on views into "equivalent" updates on base tables?
  - ▫ Not always (in fact, not often)
  - ▫ Most systems prohibit most view updates
- □ We cannot insert into Synergy --- it is a virtual view.

22

Interpreting a View Insertion

- □ But we could try to translate a (drinker, beer, bar) triple into three insertions of projected pairs, one for each of Likes, Sells, and Frequents.

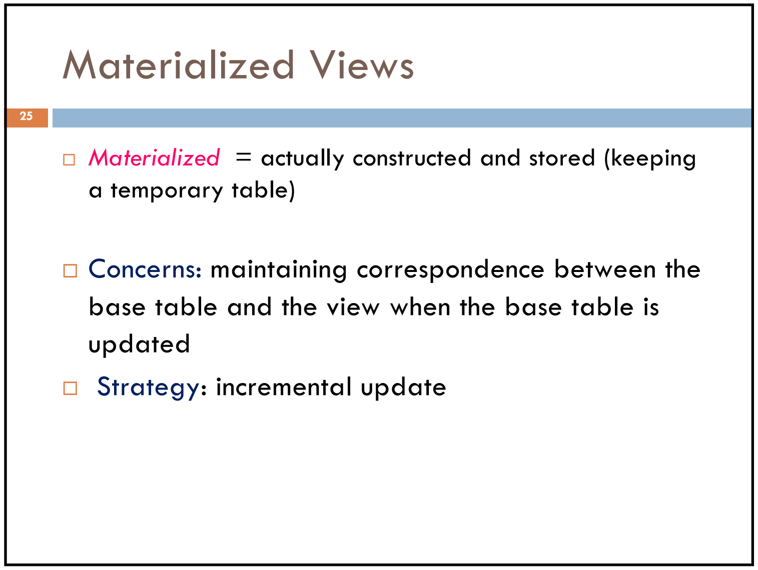
23

Insertion

```
INSERT INTO LIKES VALUES(n.drinker, n.beer);
INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer);
INSERT INTO FREQUENTS VALUES(n.drinker, n.bar);
```

- ▫ Sells.price will have to be NULL.
- ▫ There isn't always a unique translation.

24

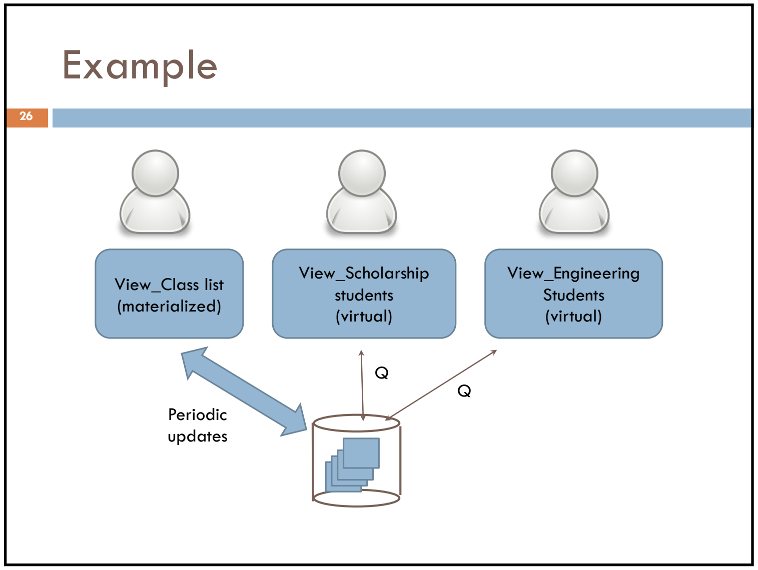


Materialized Views

25

- □ *Materialized* = actually constructed and stored (keeping a temporary table)
- □ Concerns: maintaining correspondence between the base table and the view when the base table is updated
- □ Strategy: incremental update

25

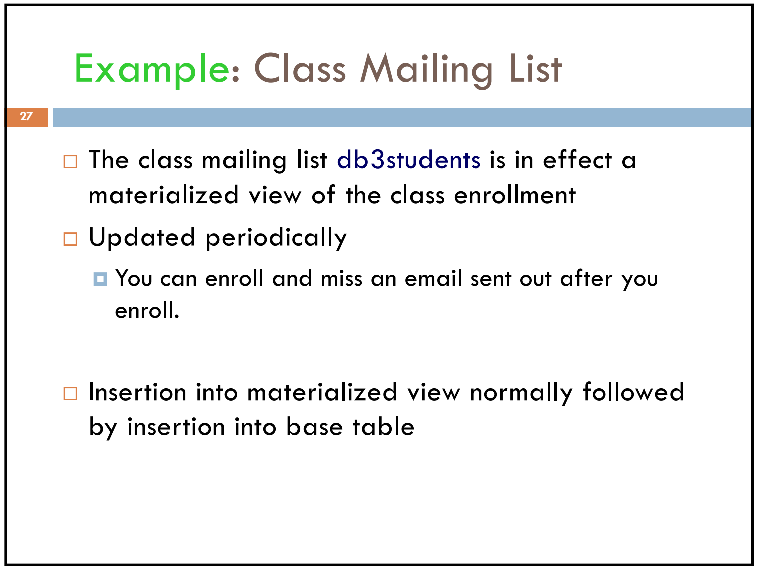


Example

26

View_Class list (materialized)
View_Scholarship students (virtual)
View_Engineering Students (virtual)
Periodic updates
Q
Q

26

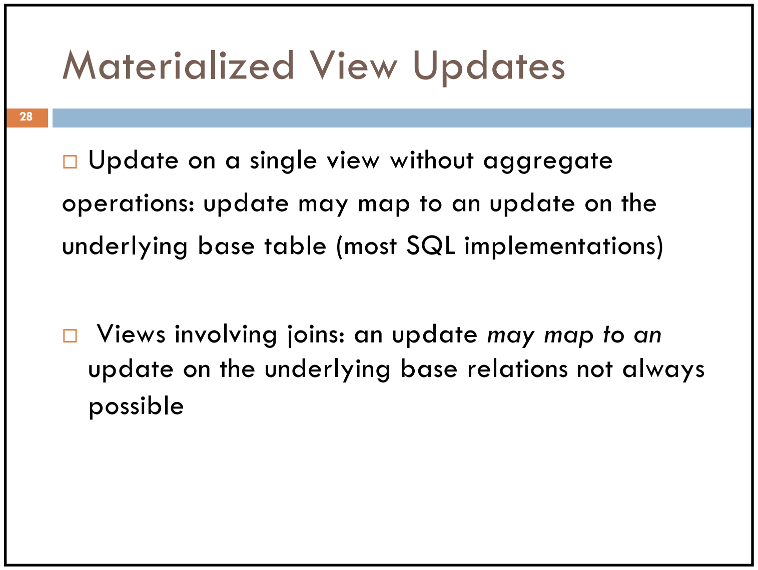


Example: Class Mailing List

27

- □ The class mailing list db3students is in effect a materialized view of the class enrollment
- □ Updated periodically
  - □ You can enroll and miss an email sent out after you enroll.
- □ Insertion into materialized view normally followed by insertion into base table

27

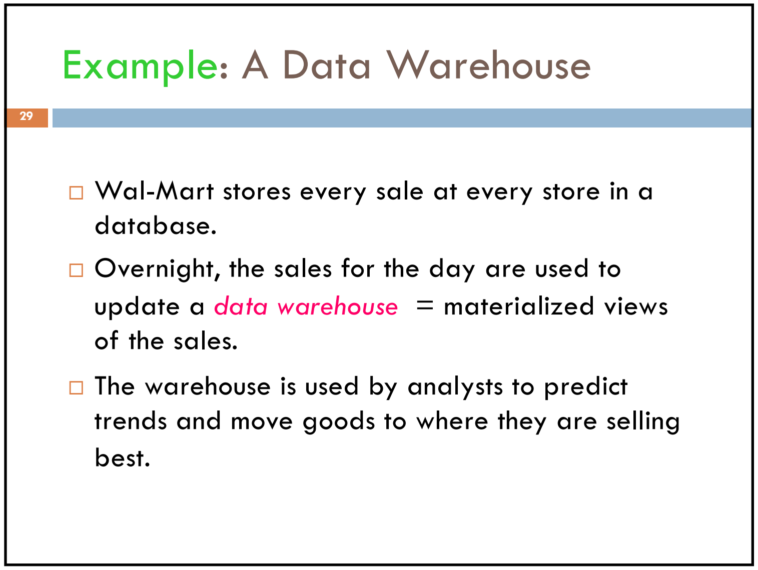


Materialized View Updates

28

- □ Update on a single view without aggregate operations: update may map to an update on the underlying base table (most SQL implementations)
- □ Views involving joins: an update *may map to an* update on the underlying base relations not always possible

28



Example: A Data Warehouse

29

- □ Wal-Mart stores every sale at every store in a database.
- □ Overnight, the sales for the day are used to update a *data warehouse* = materialized views of the sales.
- □ The warehouse is used by analysts to predict trends and move goods to where they are selling best.

29