Sorting

- R1 := τ_L (R2).
  - L is a list of some of the attributes of R2.
- R1 is the list of tuples of R2 sorted first on the value of the first attribute on L, then on the second attribute of L, and so on.
  - Break ties arbitrarily.
- τ is the only operator whose result is neither a set nor a bag.

Example: Sorting

R =

A	B
1	2
3	4
5	2

τ_B (R) = [(5,2), (1,2), (3,4)]

Aggregation Operators

- Aggregation operators are not operators of relational algebra.
- Rather, they apply to entire columns of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.

Example: Aggregation

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

Grouping Operator

9

- R1 := γ_L (R2). *L* is a list of elements that are either:
  1. Individual (*grouping*) attributes.
  2. AGG(*A*), where AGG is one of the aggregation operators and *A* is an attribute.
     - An arrow and a new attribute name renames the component.

Applying γ_L(R)

10

- Group *R* according to all the grouping attributes on list *L*.
  - That is: form one group for each distinct list of values for those attributes in *R*.
- Within each group, compute AGG(*A*) for each aggregation on list *L*.
- Result has one tuple for each group:
  1. The grouping attributes and
  2. Their group's aggregations.

Example: Grouping/Aggregation

11

R = (A B C)

A	B	C
1	2	3
4	5	6
1	2	5

$\gamma_{A,B,AVG(C)\text{->}X}$ (R) = ??

First, group *R* by *A* and *B* :

A	B	C
1	2	3
1	2	5
4	5	6

Then, average *C* within groups:

A	B	X
1	2	4
4	5	6

Recall: Outerjoin

12

- Suppose we join $R \bowtie_C S$.
- 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*.
- Outerjoin preserves dangling tuples by padding them NULL.

Example: Outerjoin

R = (A B)

A	B
1	2
4	5

S = (B C)

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples are dangling.

R OUTERJOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

Outer Join – Example

- instructor

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

teaches

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

instructor $\bowtie$ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

  instructor ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example

- Right Outer Join

  instructor ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join

  instructor ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Outer Join using Joins

- Outer join can be expressed using basic operations
  - e.g. r ⟕ s can be written as

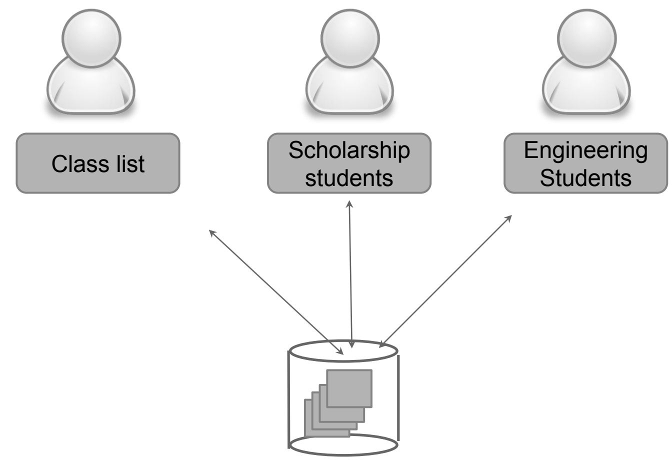
$$(r \bowtie s) \cup (r - \pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$$

VIEWS & INDEXES

MICHAEL LIUT (LIUTM@MCMASTER.CA)
DEPARTMENT OF COMPUTING AND SOFTWARE
MCMASTER UNIVERSITY

SE 3DB3 (Slides adapted from Dr. Fei Chiang) Fall 2016

Scenario

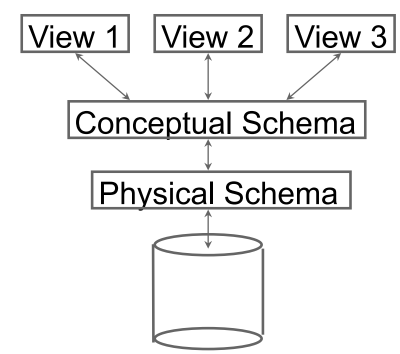


Views

- 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.

Levels of Abstraction

- 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

Views

- 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.

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.

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;
```

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';
```

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.

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

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.

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.

Insertion

15

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.

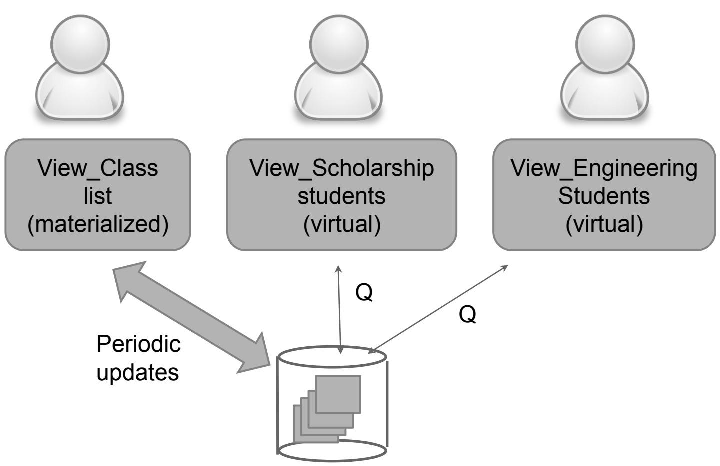
Materialized Views

16

- *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

Example

17



Example: Class Mailing List

18

- 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

Materialized View Updates

19

- 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

Example: A Data Warehouse

20

- 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.

Indexes

21

- *Index* = data structure used to speed access to tuples of a relation, given values of one or more attributes.
- Could be a hash table, but in a DBMS it is always a balanced search tree with giant nodes called a *B-tree*.