VirginiaTech

CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #5: SQL and Relational Algebra---Part 3

NOT in BOOK!

EXTENDED OPERATORS

Bags

- A *bag* (or *multi-set*) is like a set, but an element may appear more than once.
- Example: {1,2,1,3} is a bag.
- Example: {1,2,3} is also a bag that happens to be a set.

Why Bags?

- Real RDBMSs treat relations as bags of tuples.
  - SQL, is actually a bag language.
  - With exceptions, all operators are multi-set by default
- Performance is one of the main reasons; duplicate elimination is expensive since it requires sorting.
  - Some operations, like projection, are much more efficient on bags than sets.
- But RA is a set language
  - Default: all operators are set-based
- If we use bag semantics, we may have to redefine the meaning of each relation algebra operator.

Operations on Bags

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, but as a bag operator, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Bag Semantics: Projection and Selection

- Project: process each tuple independently; a tuple might occur multiple times
- Selection: process each tuple independently…

R		
A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

$\pi_{A,B}(\mathrm{R})$	
A	B
1	2
1	2
2	3
2	3

$\sigma_{C \geq 3}(R)$		
A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

Bag Union

- An element appears in the union of two bags the sum of the number of times it appears in each bag.
- R **U** S: if tuple t appears k times in R and l times in S, t appears in R **U** S k + l times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cup S$

A	B
1	2
1	2
1	2
1	2
1	2
2	3
2	3
2	3
2	4

Bag Intersection

- An element appears in the intersection of two bags the minimum of the number of times it appears in either.
- R ∩ S: if tuple t appears k times in R and l times in S, t appears min {k, l} times in R ∩ S

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cap S$

A	B
1	2
1	2
2	3

Bag Difference

- An element appears in the difference R - S of bags as many times as it appears in *R*, minus the number of times it appears in *S*.
  – But never less than 0 times.
- R –S: if tuple t appears k times in R and l times in S, t appears in R – S max{0, k – l} times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R - S$

A	B
2	3

Bag Semantics: Products and Joins

- Product (X): If a tuple r appears k times in a relation R and tuple s appears l times in a relation S, then the tuple <r, s> appears kl times in $R \times S$.

- Theta-join and Natural join ($\bowtie$): Since both can be expressed as applying a selection followed by a projection to a product, use the semantics of selection, projection, and the product.

NOT in BOOK

Extended Operators

- Powerful operators based on basic relational operators and bag semantics.
- Duplicate elimination: turn a bag into a set by eliminating duplicate tuples.
- Grouping: partition the tuples of a relation into groups, based on their values among specified attributes.
- Aggregation: used by the grouping operator and to manipulate/combine attributes.
- Extended projections: projection on steroids.
- Outerjoin: extension of joins that make sure every tuple is in the output.

Duplicate Elimination

- RA: $\delta(R)$
  - Relation with one copy for each tuple
  - Again, needed ONLY with bag-semantics!

- SQL Equivalent?
  - SELECT DISTINCT ……
- **IMPORTANT ANOMALY:** SQL UNION, INTERSECT, EXCEPT eliminate duplicates by default!
  - To make them bag-semantics add keyword ALL like UNION ALL

Example: Duplicate Elimination

R = (

A	B
1	2
3	4
1	2

)

δ(R) =

A	B
1	2
3	4

Extended Projection

- Using the same $\boldsymbol{\pi}_L$ operator, we allow the list L to contain arbitrary expressions involving attributes, for example:
    - Arithmetic on attributes, e.g., $A+B$.
    - Duplicate occurrences of the same attribute.

Example: Extended Projection

R = (

A	B)
1	2
3	4

$\pi_{A+B,A,A}(R)$ =

A+B	A1	A2
3	1	1
7	3	3

Aggregation Operators

- Operators are the same in relational algebra and SQL.
  - All operators treat a relation as a bag of tuples.
- SUM: computes the sum of a column with numerical values.
- AVG: computes the average of a column with numerical values.
- MIN and MAX:
  - for a column with numerical values, computes the smallest or largest value, respectively.
  - for a column with string or character values, computes the lexicographically smallest or largest values, respectively.
- COUNT: computes the number of tuples in a column.

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

- RA: $\gamma_L(R)$
  - *L* = grouping attribute, aggregated attribute → new attr. name

- Example: Count the number of courses each dept. teaches (COURSES(deptName, number, enrollment) relation)
  - SQL?

    SELECT DeptName, COUNT(Number) AS NumCourses

    FROM COURSES

    GROUP BY deptName;
  - Extended RA?

$$\gamma_{DeptName, COUNT(Number) \rightarrow NumCourses}(COURSES)$$

Another Example: Grouping/ Aggregation

R = (

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

)

$\gamma_{A,B,\mathrm{AVG}(C)}$ (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	AVG(C)
1	2	4
4	5	6

Joins

so far: 'INNER' joins, eg:

```
select ssn, c-name
from takes, class
where takes.c-id = class.c-id
```

Joins

Equivalently:

Non-standard form

select ssn, c-name

from takes **join** class **on** takes.c-id = class.c-id

Outerjoin

- Suppose we have: $R \bowtie_C S$
- 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 with a special NULL symbol in the result.

Example: Outerjoin

R = (

A	B
1	2
4	5

)

S = (

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-Joins

select [column list]
from *table_name*
  {**left** | **right** | **full**} **outer join**
  table_name
  on *qualification_list*
where…

RA: [redacted], [redacted], [redacted]

VIEWS

Views

- A view is a relation that does not exist physically.
- A view is defined by a query over other relations (tables and/or views).
- Just like a table, a view can be
  - queried: the query processor replaces the view by its definition.
  - used in other queries.
- Unlike a table, a view cannot be updated unless it satisfies certain conditions.

Example: View Definition

- CREATE VIEW ViewName AS Query;

- Suppose we want to perform a set of queries on those students who have taken courses both in the computer science and the mathematics departments.
- Let us create a view to store the PIDs of these students and the CS-Math course pairs they took.

Example: View Definition

- Suppose we want to perform a set of queries on those students who have taken courses both in the computer science and the mathematics departments.
- Let us create a view to store the PIDs of these students and the CS-Math course pairs they took.

```
CREATE VIEW CSMathStudents AS
 SELECT T1.StudentPID, T1.Number AS CSNum, T2.Number AS
MathNum
 FROM Take AS T1, Take AS T2
 WHERE (T1.StudentPID = T2.StudentPID)
     AND (T1.DeptName = 'CS')
     AND (T2.DeptName = 'Math');
```

VirginiaTech

Querying Views

- Query a view as if it were a base table.
- How many students took both CS and Math courses?

```
SELECT COUNT(StudentPID)
FROM CSMathStudents
```

Querying Views

- Just replace view by its definition

```
SELECT COUNT(StudentPID)
FROM CSMathStudents

SELECT COUNT(StudentPID)
FROM
  (SELECT T1.StudentPID, T1.Number AS CSNum,
 T2.Number AS MathNum
  FROM Take AS T1, Take AS T2
  WHERE (T1.StudentPID = T2.StudentPID)
       AND (T1.DeptName = ’CS’)
       AND (T2.DeptName = ’Math’));
```

Modifying Views

- What does it mean to modify a view?
- How is tuple deletion from a view executed?
- Can we insert a tuple into a view? Where will it be inserted, since a view does not physically exist?
- Can we insert tuples into any view? SQL includes rules that specify which views are updatable.

Deleting Views

- DROP VIEW CSMathStudents;
- Like a Symbolic Link: only the view definition is deleted

Deleting Tuples from Views

- Delete tuples for students taking 'CS 4604'.

  DELETE FROM CSMathStudents

  WHERE (CSNum = 4604);
- Deletion is executed as if were executing

  DELETE FROM Take

  WHERE (Number = 4604);

  ?
- Incorrect: non-CS tuples where (Number = 4604) will be deleted.

Deleting Tuples from Views

- Tuples only seen in the view should be deleted!
- Add conditions to the WHERE clause

```
DELETE FROM CSMathStudents
WHERE (CSNum = 4604) AND (DeptName = 'CS');
```

Inserting tuples into Views

- Again, passed through to the underlying relation

INSERT INTO CSMathStudents

VALUES ('123-45-6789', 4604, 8811);

- But Take schema is (PID, Number, Dept)
  – what should dept values be?
  – NULL?

  Then it is not part of CSMathStudents!

Inserting tuples into Views

- CREATE VIEW CSStudents AS
  SELECT StudentPID, Number
  FROM Take
  WHERE (DeptName = 'CS');

Works?

Same Problem

- INSERT INTO CSStudents
  VALUES ('123-45-6789', 4604);

Inserting tuples into Views

- Include DeptName in the view's schema
- CREATE VIEW CSStudents AS

  SELECT StudentPID, DeptName, Number

  FROM Take

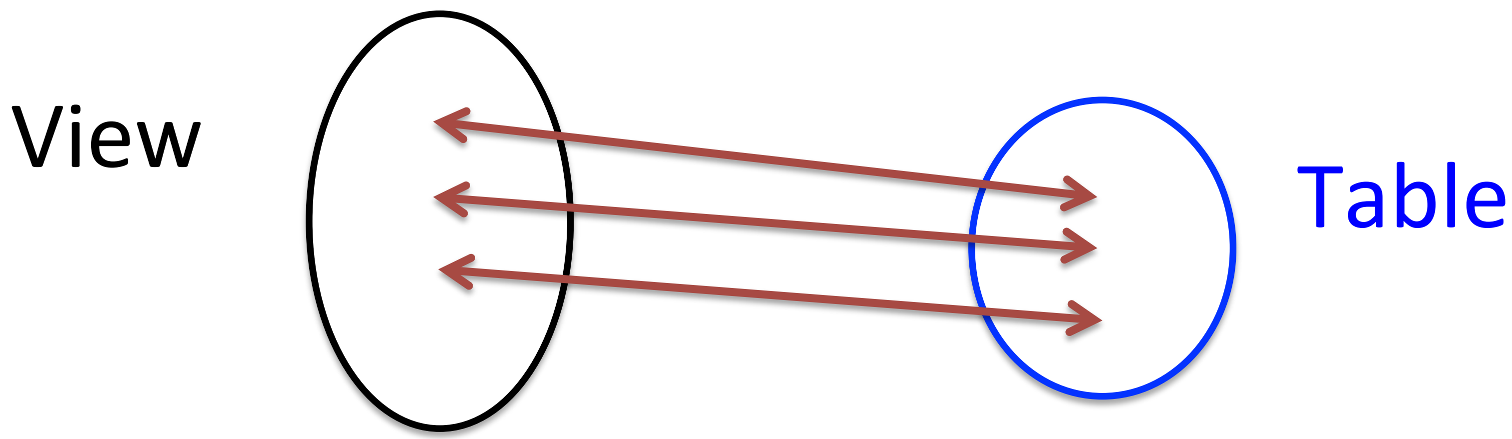
  WHERE (DeptName = 'CS');

- INSERT INTO CSStudents

  VALUES ('123-45-6789', 'CS', 4604)

Updatable Views

- The idea is that there must be a one-one relationship between rows in the view and the rows in the underlying table



Updatable Views

SQL:92 standard:

- Defined by selecting/projecting some attributes from one relation R
- R may itself be an updatable view.
- Use SELECT and not SELECT DISTINCT.
- FROM clause can contain only one occurrence of R and must not contain any other relation.
- NO aggregation operations

Materialized Views

- Two kinds:
  1. ***Virtual*** = not stored in the database; just a query for constructing the relation.
  2. *Materialized* = actually constructed and stored.

WHY?

– Some views may be frequently used in queries.

– It may be efficient to materialize such a view, i.e., maintain its value at all times as a physical table

Declaring Views

- Declare by:

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

- Default is virtual.

Maintaining Materializing Views

- Cost?
  - Re-computing it when the underlying tables change
  - Materialized view may be much larger than original relations, e.g., in the case of joins

Maintaining Materialized Views

- CREATE MATERIALIZED VIEW CSStudents AS
  SELECT StudentPID, DeptName, Number
  FROM Take
  WHERE (DeptName = 'CS');
- When?
  - Insertion/deletion/update of Take
- Cost?
  - Insertion of tuple: Insert tuple into CSStudents only if new tuple has DeptName = 'CS’
  - Same for Deletion
  - Update? Delete followed by an Insert…

Maintaining Materialized Views

- Key idea is that many materialized views can be updated incrementally.

- Read Sections 25.9, and 25.10.1 from the textbook (~3 pages total)

Maintaining Materialized Views with Joins

- CREATE MATERIALIZED VIEW CSMathProfs(PID, Pname, CNum, CName) AS

  SELECT PID, P.Name, T.Number, T.Name

  FROM Teach AS T, Professors AS P

  WHERE (P.DeptName = 'CS') AND (T.DeptName = 'Math') AND (T.ProfessorPID = P.PID);

- Insert a tuple t into Teach:

- Delete a tuple t from Teach:

Maintaining Materialized Views with Joins

- CREATE MATERIALIZED VIEW CSMathProfs(PID, Pname, CNum, CName) AS

  SELECT PID, P.Name, T.Number, T.Name

  FROM Teach AS T, Professors AS P

  WHERE (P.DeptName = 'CS') AND (T.DeptName = 'Math') AND (T.ProfessorPID = P.PID);

- Insert a tuple t into Teach (assume t.DeptName = Math):

  Find the tuple p in Professors such that (t.ProfessorPID = p.PID) AND (p.DeptName = 'CS').

  Insert (p.PID, p.Name, t.Number, t.Name) into CSMathProfs

Maintaining Materialized Views with Joins

- CREATE MATERIALIZED VIEW CSMathProfs(PID, Pname, CNum, CName) AS

  SELECT PID, P.Name, T.Number, T.Name

  FROM Teach AS T, Professors AS P

  WHERE (P.DeptName = 'CS') AND (T.DeptName = 'Math') AND (T.ProfessorPID = P.PID);

- Delete a tuple t from Teach (assume t.DeptName = Math):

  DELETE FROM CSMathProfs WHERE CNum = t.Number;

Maintaining Materialized Views with Joins

- CREATE MATERIALIZED VIEW CSMathProfs(PID, Pname, CNum, CName) AS

  SELECT PID, P.Name, T.Number, T.Name

  FROM Teach AS T, Professors AS P

  WHERE (P.DeptName = 'CS') AND (T.DeptName = 'Math') AND (T.ProfessorPID = P.PID);

- Insert a tuple t into Professors:

- Delete a tuple t into Professors:

Maintaining Materialized Views with Joins

- CREATE MATERIALIZED VIEW CSMathProfs(PID, Pname, CNum, CName) AS

```
SELECT PID, P.Name, T.Number, T.Name
FROM Teach AS T, Professors AS P
WHERE (P.DeptName = 'CS') AND (T.DeptName = 'Math') AND
(T.ProfessorPID = P.PID);
```

- Insert a tuple t into Professors (assume p.DeptName = CS):

```
INSERT INTO CSMathProfs
    SELECT p.PID, p.Name, T.Number, T.Name
    WHERE (p.PID = T.ProfessorPID) AND (T.DeptName = 'Math');
```

Maintaining Materialized Views with Joins

- CREATE MATERIALIZED VIEW CSMathProfs(PID, Pname, CNum, CName) AS

  SELECT PID, P.Name, T.Number, T.Name

  FROM Teach AS T, Professors AS P

  WHERE (P.DeptName = 'CS') AND (T.DeptName = 'Math') AND (T.ProfessorPID = P.PID);

- Delete a tuple t from Professors (assume p.DeptName = CS):

  DELETE FROM CSMathProfs WHERE (PID = p.PID);

Periodic Maintenance

- DB for inventory of a department store.
- Aggregate buyer patterns for further analysis → can be a (materialized) view
- Analysis is only periodic, so update the materialized view at only regular intervals

Rewriting Queries Using Materialized Views

EXTRA: NOT IN EXAM

- In practice, views are materialized because they are helpful to answer common queries.

- Can we rewrite a query to use a materialized view rather than the original relations?

Rewriting Queries Using Materialized Views

EXTRA: NOT IN EXAM

- Find names and addresses of students taking CS courses

  SELECT Name, Address

  FROM Students, Take

  WHERE (Students.PID = Take.StudentPID) AND (DeptName = 'CS');

Rewrite it using CSStudents?

SELECT Name, Address

FROM Students, CSStudents

WHERE (Students.PID = CSStudents.StudentPID);

VirginiaTech

Rules for Rewriting Queries

EXTRA: NOT IN EXAM

- Complete sets of rules is very complex!
- A simple rule

View V:	Query Q:	(New) Query Q':
SELECT *LV*	SELECT *LQ*	SELECT *LQ*
FROM *RV*	FROM *RQ*	FROM V, *RQ* - *RV*
WHERE *CV*	WHERE *CQ*	WHERE *C*

- We can replace Q by the new query Q' if
  - RV $\subseteq$ RQ
  - CQ == CV AND C, for some condition C, which may be empty
  - If C is not empty, then attributes of relations in RV that C mentions are also in LV
  - Attributes in LQ that come from relations in RV are also in the list of attributes LV