Views

T. M. Murali

September 21, 2009
What are 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.
- Just like a table, a view can be used in other queries.
- Unlike a table, a view cannot be updated unless it satisfies certain conditions.
Defining a View

- CREATE VIEW ViewName AS Query;
Defining a View

- CREATE VIEW ViewName AS Query;
- Suppose we want to perform a set of queries on those students who have taken courses in the computer science departments.
- Let us create a view to store the PIDs of these students and the numbers of the CS courses they are taking.
Defining a View

- CREATE VIEW ViewName AS Query;
- Suppose we want to perform a set of queries on those students who have taken courses in the computer science departments.
- Let us create a view to store the PIDs of these students and the numbers of the CS courses they are taking.

CREATE VIEW CSStudents AS

    SELECT StudentPID, Number
    FROM Take
    WHERE (DeptName = 'CS');
Defining a View

- CREATE VIEW ViewName AS Query;
- Suppose we want to perform a set of queries on those students who have taken courses in the computer science departments.
- Let us create a view to store the PIDs of these students and the numbers of the CS courses they are taking.

```sql
CREATE VIEW CSStudents(PID, CSCourseNumber) AS
    SELECT StudentPID, Number
    FROM Take
    WHERE (DeptName = 'CS');
```
- Can name attributes in the view’s schema.
We can query views just as we can query tables.

How many students took a CS course, i.e., what was the total enrollment in CS courses?

```
SELECT COUNT(PID)
FROM CSStudents;
```

Interpreting this query: replace view by its definition.

```
SELECT COUNT(StudentPID)
FROM (SELECT StudentPID, Number
      FROM Take
      WHERE (DeptName = 'CS'))
```
Querying Views

- We can query views just as we can query tables.
- How many students took a CS course, i.e., what was the total enrollment in CS courses?
Querying Views

▶ We can query views just as we can query tables.
▶ How many students took a CS course, i.e., what was the total enrollment in CS courses?

```sql
SELECT COUNT (PID)
FROM CSStudents;
```

Interpreting this query: replace view by its definition.

```sql
SELECT COUNT (StudentPID)
FROM (SELECT StudentPID, Number
      FROM Take
      WHERE (DeptName = 'CS'))
```

T. M. Murali September 21, 2009 CS 4604: Views
Querying Views

We can query views just as we can query tables.

How many students took a CS course, i.e., what was the total enrollment in CS courses?

\[
\text{SELECT COUNT (PID) FROM CSStudents;}
\]

Interpreting this query: replace view by its definition.

\[
\text{SELECT COUNT (StudentPID) FROM}
\]
\[
(\text{SELECT StudentPID, Number FROM Take WHERE (DeptName = 'CS');})
\]
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?
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 CSStudents;
- Only view definition deleted.
- Underlying tables are not affected.
Deleting Tuples from CSStudents

- Delete tuples for students taking ‘CS 4604’.

  ```sql
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604);
  ```

  Deletion is executed as if were executing
  ```sql
  DELETE FROM Take
  WHERE (Number = 4604);
  ```

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

  Correct deletion adds condition in the
  ```sql
  WHERE
  ```
  clause of the view
  ```sql
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604) AND (DeptName = 'CS');
  ```
Deleting Tuples from CSStudents

- Delete tuples for students taking 'CS 4604'.

  ```sql
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604);
  ```

  Deletion is executed as if were executing
  ```sql
  DELETE FROM Take
  WHERE (Number = 4604);
  ```

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

  Correct deletion adds condition in the
  ```sql
  WHERE
  ```
  clause of the view
  definition to the
  ```sql
  WHERE
  ```
  clause of the
  ```sql
  DELETE FROM
  ```
  statement.

  ```sql
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604) AND (DeptName = 'CS');
  ```
Deleting Tuples from CSStudents

- Delete tuples for students taking 'CS 4604'.

  ```
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604);
  ```

- Deletion is executed as if we were executing

  ```
  DELETE FROM Take
  WHERE (Number = 4604);
  ```

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

- Correct deletion adds condition in the WHERE clause of the view definition to the WHERE clause of the DELETE FROM statement.

  ```
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604) AND (DeptName = 'CS');
  ```
Deleting Tuples from CSStudents

- Delete tuples for students taking 'CS 4604'.
  
  ```sql
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604);
  ```

- Deletion is executed as if were executing
  
  ```sql
  DELETE FROM Take
  WHERE (Number = 4604);
  ```

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

  ```sql
  DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604) AND (DeptName = 'CS');
  ```
Deleting Tuples from CSStudents

- Delete tuples for students taking 'CS 4604'.
  
  `DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604);`

- Deletion is executed as if were executing
  
  `DELETE FROM Take
  WHERE (Number = 4604);`

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

- Correct deletion adds condition in the `WHERE` clause of the view definition to the `WHERE` clause of the `DELETE FROM` statement.
  
  `DELETE FROM CSStudents
  WHERE (CSCourseNumber = 4604) AND (DeptName = 'CS');`
Inserting a Tuple into CSStudents

\begin{verbatim}
INSERT INTO CSStudents
VALUES ('123-45-6789', 4604);
\end{verbatim}
Inserting a Tuple into CSStudents

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

- Insertion is executed as if we were executing
  
  ```
  INSERT INTO Take(StudentPID, Number)
  VALUES ('123-45-6789', 4604);
  ```

  Two problems:
  1. Key for `Take` is `(PID, DeptName, Number)`. What value does the new tuple get for `DeptName`?
  2. Even if we had not declared key for `Take`, new tuple gets `NULL` value for `DeptName` ⇒ it is not a part of the view after insertion!
Inserting a Tuple into CSStudents

\[
\text{INSERT INTO CSStudents} \\
\text{VALUES ('123-45-6789', 4604);} \\
\]

- Insertion is executed as if we were executing
  \[
  \text{INSERT INTO Take(StudentPID, Number)} \\
  \text{VALUES ('123-45-6789', 4604);} \\
  \]

- Two problems:
  1. Key for Take is (PID, DeptName, Number). What value does the new tuple get for DeptName?
  2. Even if we had not declared key for Take, new tuple gets NULL value for DeptName ⇒ it is not a part of the view after insertion!
Correcting the Definition of CSStudents

- Include DeptName in the view’s schema.

```sql
CREATE VIEW CSStudents AS
SELECT StudentPID, DeptName, Number
FROM Take
WHERE (DeptName = 'CS');
```
Correcting the Definition of **CSStudents**

- Include DeptName in the view’s schema.
  
  ```sql
  CREATE VIEW CSStudents AS
  SELECT StudentPID, DeptName, Number
  FROM Take
  WHERE (DeptName = 'CS');
  ```

- Write INSERT INTO as follows:
  
  ```sql
  INSERT INTO CSStudents
  VALUES ('123-45-6789', 'CS', 4604);
  ```
Updatable Views

- SQL has complex rules for when a view can be modified.
- Defined by selecting some attributes from one relation $R$.
- $R$ may itself be an updatable view.
- Use SELECT and not SELECT DISTINCT.
- WHERE clause must not involve $R$ in a sub-query.
- FROM clause can contain only one occurrence of $R$ and must not contain any other relation.
- SELECT clause must contain enough attributes so that for every tuple inserted into the view, other attributes can get NULL values or default values.
  - An attribute that is declared NOT NULL and has no default must be mentioned in the SELECT clause.
Materializing Views

- Some views may be frequently used in queries.
- It may be efficient to *materialise* such a view, i.e., maintain its value at all times as a physical table.
- Cost of materialising views:
Materializing Views

- Some views may be frequently used in queries.
- It may be efficient to *materialise* such a view, i.e., maintain its value at all times as a physical table.
- Cost of materialising views:
  - Recomputing it when the underlying tables change.
  - Materialised view may be much larger than original relations, e.g., in the case of joins.
Maintaining Materialised Views

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

▶ When does CSStudents need to be maintained?
Creating a Materialized View

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

- When does `CSStudents` need to be maintained? Insertion/deletion/update of `Take`.
- How expensive is the modification of `CSStudents`?
- Key idea is that many materialized views can be updated incrementally.
- Incremental maintenance of a view that involves a join: read Chapter 8.5.1 of the textbook.
Maintaining Materialised Views

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

▶ When does CSStudents need to be maintained?
   Insertion/deletion/update of Take.
▶ How expensive is the modification of CSStudents?
   ▶ Insertion of tuple: Insert tuple into CSStudents only if new tuple has DeptName = ‘CS’.
Maintaining Materialised Views

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

- When does CSStudents need to be maintained?
  Insertion/deletion/update of Take.

- How expensive is the modification of CSStudents?
  - Insertion of tuple: Insert tuple into CSStudents only if new tuple has DeptName = 'CS'.
  - Deletion of tuple: same reasoning as insertion.
  - Update of tuple: View as delete followed by insert.

Key idea is that many materialised views can be updated incrementally. Incremental maintenance of a view that involves a join: read Chapter 8.5.1 of the textbook.
Maintaining Materialised Views

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

▶ When does CSStudents need to be maintained?
   Insertion/deletion/update of Take.

▶ How expensive is the modification of CSStudents?
  ▶ Insertion of tuple: Insert tuple into CSStudents only if new tuple has DeptName = 'CS'.
  ▶ Deletion of tuple: same reasoning as insertion.
  ▶ Update of tuple: View as delete followed by insert.

▶ Key idea is that many materialised views can be updated incrementally.

▶ Incremental maintenance of a view that involves a join: read Chapter 8.5.1 of the textbook.
Periodic Maintenance

- Consider a database that records the inventory of a department store.
- A view may aggregate buyer patterns for further analysis.
- A (materialised) view is a good candidate for storing these aggregated data.
Periodic Maintenance

- Consider a database that records the inventory of a department store.
- A view may aggregate buyer patterns for further analysis.
- A (materialised) view is a good candidate for storing these aggregated data.
- Since such analysis is performed periodically, materialised view need not be maintained after each customer transaction. It can be update at regular intervals, for instance once every day.
- Automatic creation of materialised views: Read Chapter 8.5.4 of the textbook.
Rewriting Queries Using Materialised Views

- In practice, views are materialised because they are helpful to answer common queries.
- Can we rewrite a query to *use* a materialised view rather than the original relations?
Rewriting Queries Using Materialised Views

- In practice, views are materialised because they are helpful to answer common queries.
- Can we rewrite a query to use a materialised view rather than the original relations?
- Find names and addresses of students taking CS courses?
Rewriting Queries Using Materialised Views

- In practice, views are materialised because they are helpful to answer common queries.
- Can we rewrite a query to use a materialised view rather than the original relations?
- Find names and addresses of students taking CS courses?
  
  ```sql
  SELECT Name, Address
  FROM Students, Take
  WHERE (Students.PID = Take.StudentPID) AND (DeptName = 'CS');
  ```
- Can we rewrite the query using the view CSStudents?
Rewriting Queries Using Materialised Views

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

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

- Find names and addresses of students taking CS courses?

  ```sql
  SELECT Name, Address
  FROM Students, Take
  WHERE (Students.PID = Take.StudentPID) AND (DeptName = 'CS');
  ```

- Can we rewrite the query using the view CSStudents?

  ```sql
  SELECT Name, Address
  FROM Students, CSStudents
  WHERE (Students.PID = CSStudents.StudentPID);
  ```
Rules for Rewriting Queries

Complete sets of rules is very complex. We discuss a simple rule to rewrite a query using a materialised view.
Rules for Rewriting Queries

- Complete sets of rules is very complex. We discuss a simple rule to rewrite a query using a materialised view.

View $V$:
SELECT $L_V$
FROM $R_V$
WHERE $C_V$.

Query $Q$:
SELECT $L_Q$
FROM $R_Q$
WHERE $C_Q$.

New query $Q'$:
SELECT $L_Q$
FROM $V$,$R_Q$ − $R_V$
WHERE $C$.

- We can replace $Q$ by the new query $Q'$ if
  - $R_V \subseteq R_Q$.
  - $C_Q \equiv C_V$ AND $C$, for some condition $C$, which may be empty.
  - If $C$ is not empty, then attributes of relations in $R_V$ that $C$ mentions are also in $L_V$.
  - Attributes in $L_Q$ that come from relations in $R_V$ are also in the list of attributes $L_V$. 

T. M. Murali September 21, 2009 CS 4604: Views