CS143 Notes: Views & Authorization

Book Chapters

(4th) Chapter 4.7, 6.5-6
(5th) Chapter 4.2, 8.6
(6th) Chapter 4.4, 5.3

Views

• What is a view?
  – A “virtual” table created on top of other “real” tables
  – Almost the same as a “real” table except that
    * the tuples are computed on the fly using “real” tables.
    * a view does not really “exist”.

• Syntax and example:
  – CREATE VIEW ViewName(A₁, A₂, ...) AS Query
    * Attribute lists are optional
    * Example: SidNameAddr view with (sid, name, address) from Student

  – Views can be used in a query like:
    SELECT *
    FROM SidNameAddr S, Enroll E
    WHERE S.sid = E.sid
    * The system automatically rewrites the query using the SidNameAddr
      view definition

  – Views can be created on top of other views
    CREATE VIEW NameAddr AS
    SELECT name, addr
    FROM SidNameAddr
– Q: MultiClass: View of students (sid, name) who take more than one class?

– Q: Why use views?

– Three-level vision of database:
  * Virtual database: Views \( V := \text{ViewQuery}(R_1, R_2, \ldots, R_n) \)
    built on top of . . .
  * Conceptual database: Tables (Relations)
    built on top of . . .
  * Physical database: Pages on disk

Modifying Views

• Updates on views are allowed (under certain conditions)

• Q: How can we “update” a view when it does not exist?
  – Q: UPDATE SidNameAddr SET Name = ‘James’ WHERE sid = 301?

  * Modification to a view is “translated” into a modification to the underlying table
  – Q: INSERT INTO SidNameAddr VALUES (305, ‘Peter’, ‘1234 Westwood’)?

  * Missing columns are filled with the DEFAULT value (or NULL)
  – Q: INSERT INTO SameAddr VALUES (‘Tony’, ‘Joshua’)?

CREATE VIEW SameAddr AS
  SELECT S1.name, S2.name
  FROM Student S1, Student S2
  WHERE S1.addr = S2.addr AND S1.sid > S2.sid

– Q: UPDATE AvgGPA SET a = 3.0?

CREATE VIEW AvgGPA(a) AS SELECT avg(GPA) FROM Student;
– For some views, update may not make any sense
  * Precise conditions for updatable views are very complicated
    · may involve keys, equality conditions, etc.
– SQL2 uses very conservative conditions. View must be defined as:
  * SELECT on a single table T, without DISTINCT
  * Subqueries in WHERE must not refer to T
  * Attributes of T not projected in view allowed to be be NULL or default
  * No aggregation

• Q: INSERT INTO Student17 VALUES (403, ‘Peter’, ‘123 Olympic’, 3.0, 20)

CREATE VIEW Student17 AS
SELECT *
FROM Student
WHERE age = 17

– Q: Is the new tuple in Student17?

– WITH CHECK OPTION
  * syntax: CREATE VIEW ... AS ... WITH CHECK OPTION
  * check INSERT/UPDATE to ensure the new tuple is still in the view
  * reject the statement if not

• Q: What will happen if we drop HonorStudent view?

YoungHonorStudent
  ↑
HonorStudent
  ↑
Student

– DROP ... [ CASCADE | RESTRICT ]
  * CASCADE : drop anything that references the view
  * RESTRICT (default): drop statement fails if the view is referenced by other views
    or integrity constraints

Materialized Views
• Some DBMS allows to “precompute” or “materialize” a view
• Example: MultiClass view again. Students who take multiple classes
CREATE VIEW MultiClass AS
SELECT sid, name
FROM Student, Enroll
WHERE Student.sid = Enroll.sid
GROUP BY Student.sid
HAVING COUNT(*) > 1

- Q: Why do we want to materialize this view?

- Q: Why don’t we always materialize views?

* Q: When should we refresh MultiClass?

- refresh of materialized view can be costly
- incremental refresh of materialized view is sometimes difficult

- Q: What views to materialize? Both? Just one? Pros and cons of each?

CREATE VIEW MultiClass AS
SELECT sid, name
FROM Student, Enroll
WHERE Student.sid = Enroll.sid
GROUP BY Student.sid
HAVING COUNT(*) > 1

CREATE VIEW StudDeptCount AS
SELECT sid, name, dept, COUNT(*)
FROM Enroll
GROUP BY sid, dept

* Deciding views to materialize is a difficult optimization problem

- Many commercial DBMS supports materialized views
  - used for “data warehouse” for OLAP (online analytical processing) queries
  - limited support for incremental refresh
  - materialized view selection is a difficult optimization problem
Authorization

- Make sure users only see what they are allowed to see
- Do not let an unauthorized user to modify database

Privileges

- For a relation \( R \) and user \( U \), \( U \) may be authorized for:
  - \( \text{SELECT} \) on \( R \)
  - \( \text{INSERT}(A_1, A_2, \ldots, A_n) \) on \( R \)
    * the rest of the columns should take NULL or DEFAULT
  - \( \text{UPDATE}(A_1, A_2, \ldots, A_n) \) on \( R \)
  - \( \text{DELETE} \) on \( R \)

- Q: When will it be useful to limit insert privileges to certain columns?

- \textbf{GRANT} \( \langle \text{privileges} \rangle \) on \( \langle R \rangle \) to \( \langle \text{users} \rangle \) [ WITH GRANT OPTION ]
  - Give the privilege(s) to the user(s)
  - \( \langle \text{privileges} \rangle \): SELECT, INSERT, ... separated by commas (or ALL PRIVILEGES)
  - \( \langle R \rangle \): table, view, ...
  - \( \langle \text{users} \rangle \): list of users/groups, or PUBLIC
  - more about WITH GRANT OPTION later

  \textbf{EXAMPLE}: Grant SELECT privilege on Student to \( u_2 \)

- \textbf{REVOKE} \( \langle \text{privileges} \rangle \) on \( \langle R \rangle \) from \( \langle \text{users} \rangle \) [ CASCADE | RESTRICT ]
  - Revoke the privilege(s) from the user(s).
  - More about CASCADE/DEFAULT later. Default: RESTRICT

  \textbf{EXAMPLE}: revoke SELECT privilege on Student from \( u_2 \)

- \textbf{EXAMPLES}
- UPDATE Student
  SET GPA = 4.0
  WHERE sid IN (SELECT sid FROM Enroll WHERE dept = 'CS')
  * Q: What privileges are needed for this statement?

- DELETE FROM Student
  WHERE sid NOT IN (SELECT sid FROM Enroll)
  * Q: What privileges are needed for this statement?

GRANT OPTION and cascading revoke

- Q: Who can grant privileges?
  - database administrator (DBA in oracle, SUPER in MySQL)
  - owner (= creator) of the table/view/
  - Q: What if we want to delegate privilege management to someone else?

  - GRANT ⟨privileges⟩ ON R TO u₂ WITH GRANT OPTION
    - Delegate the privilege management task to u₂.
    - User u₂ can now grant the given privilege(s) to others.
    - In addition, u₂ can further delegate privilege management to other users.

- Authorization graph.
  - Nodes: users
  - Edges:
    * When uᵢ grants privilege to uⱼ, an edge is added from uᵢ to uⱼ.
    * When u₁ creates a table/view . . ., edge is created from DBA to u₁

  - Example: What does the following authorization graph mean?
    
    DBA → u₁ → u₂ → u₃

- CASCADE/RESTRICT option in REVOKE statement
Q: Given the following authorization graph, what should happen when \( u_1 \) revokes privilege from \( u_2 \)?

\[
\text{DBA} \rightarrow u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4
\]

* CASCADE: revoke the privileges passed on from \( u_2 \) as well.
* RESTRICT (default): reject the REVOKE command

Interpretation of REVOKE

Q: Assume the following authorization graph. If \( u_1 \) revokes the privilege from \( u_2 \), what will happen to the privilege that \( u_2 \) has?

\[
\text{DBA} \rightarrow u_1 \rightarrow u_2 \\
\downarrow \quad \uparrow \\
\quad u_3
\]

In SQL92, REVOKE command removes the appropriate “edge(s)” in the authorization graph. The revoked privilege may still remain if there is another valid path for the privilege.

Q: If DBA revokes the privilege from \( u_2 \), what edge should we remove?

\[
\text{DBA} \rightarrow u_1 \rightarrow u_2 \\
\downarrow \quad \uparrow \\
\quad u_3
\]

NOTES:

* Under SQL92, \( u_1 \) can revoke privilege from \( u_2 \) only if \( u_1 \) granted the privilege to \( u_2 \). Even DBA cannot revoke a privilege from a user unless DBA granted the privilege to the user. This restriction is to avoid privilege revocation ambiguity.

* Unfortunately this restriction is too strict in practice, so most commercial systems do not enforce this. In case of ambiguity, they just do whatever is “reasonable” according to their policy.
Privileges and views

• Q: How can we allow user u₁ to see only (sid, name) of Student?

• Q: How to allow user u₁ modify a student record only if their age < 18?

• Consider the following sequence of commands. Each command is prefixed by the user who is trying to execute the command.

  
  u₁: CREATE TABLE R(A INT);
  u₂: CREATE VIEW V AS SELECT * FROM R;

  – Q: Should we allow u₂ to create such a view?

  – Q: Assume u₁ executed

  
  u₁: GRANT SELECT ON R TO ’u₂’;

  before “u₂: CREATE VIEW ...”, so V has been successfully created by u₂. Then can u₂ execute the following command?

  
  u₂: GRANT SELECT ON V TO ’u₃’;

  – Q: Now if u₁ revokes SELECT privilege on R from u₂, what should happen to V?

NOTE:

* In order to create a view, the user needs SELECT privilege(s) on all base table(s).
* To grant a privilege on a view, the user has to have GRANT OPTION privilege on the base table(s) of the view
* When the privilege(s) for the base table(s) of a view are revoked, the view may be automatically dropped as well.
Other Privileges

• Q: Who should be allowed to create tables/views/... in a database?

• Unfortunately, no standard for (table) creation/drop privilege in SQL
  – MySQL: CREATE, DROP, ALTER
  – Oracle: CREATE TABLE, CREATE VIEW, DROP ANY TABLE, ...
  – DB2: CREATETAB (table creation), CREATEALIAS (view creation), DROP, ALTER, ...

• Other privileges in (later) SQL standards
  – SQL92: REFERENCES (reference in constraints), USAGE (domain)
  – SQL99: TRIGGER, EXECUTE (function call), and UNDER (define subtype)