VirginiaTech

CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #4: SQL---Part 2

Overview - detailed - SQL

- DML
- other parts:
  - views
  - modifications
  - joins
  - DDL
  - constraints

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.

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');
```

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?

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

Same Problem

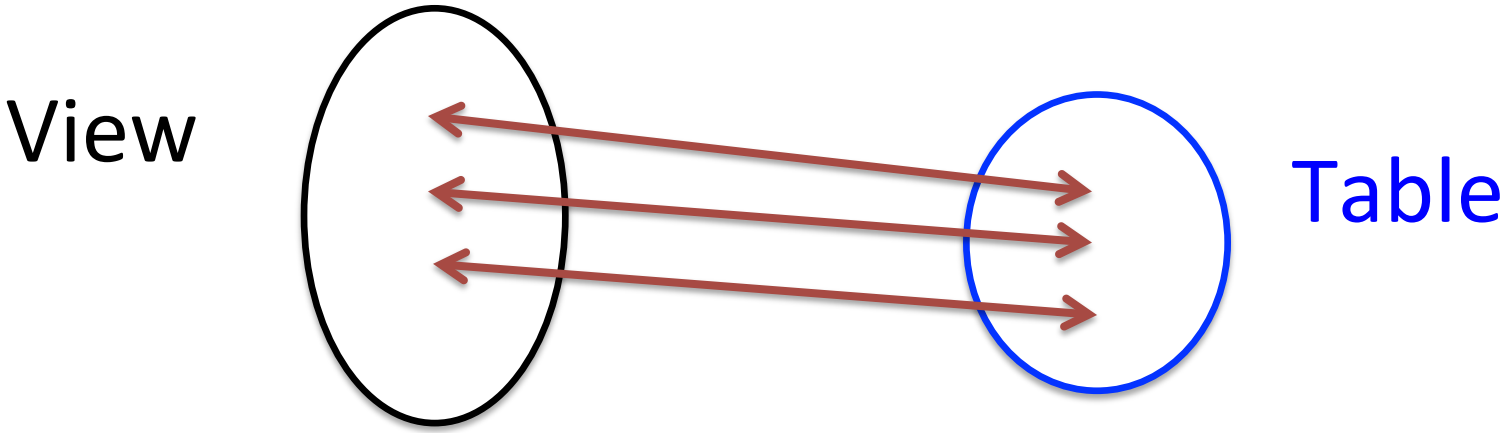
Inserting tuples into Views

- 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

EXTRA: NOT IN EXAM

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

EXTRA: NOT IN EXAM

- 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

EXTRA: NOT IN EXAM

- 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

EXTRA: NOT IN EXAM

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

- More info: Sections 25.9, and 25.10.1 from the textbook (~3 pages total)

MODIFICATIONS, JOINS, DDL

Reminder: mini-U db

STUDENT

Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

CLASS

c-id	c-name	units
4602	s.e.	2
4603	o.s.	2

TAKES

SSN	c-id	grade
123	4613	A
234	4613	B

DML - insertions etc

insert into student
values (“123”, “smith”, “main”)

insert into student(ssn, name, address)
values (“123”, “smith”, “main”)

DML - insertions etc

bulk insertion: how to insert, say, a table of 'foreign-student's, in bulk?

DML - insertions etc

bulk insertion:

insert into student

select ssn, name, address

from foreign-student

DML - deletion etc

delete the record of ‘smith’

DML - deletion etc

delete the record of ‘smith’:

delete from student

where name= ‘smith’

(careful - it deletes ALL the ‘smith’s!)

DML - update etc

record the grade ‘A’ for ssn=123 and course 4604

update takes

set grade=“A”

where ssn=“123” and c-id=“4604”

(will set to “A” ALL such records)

DML - joins

so far: ‘INNER’ joins, eg:

select ssn, c-name

from takes, class

where takes.c-id = class.c-id

DML - joins

Equivalently:

select ssn, c-name

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

Joins

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

Inner join

TAKES

SSN	c-id	grade
123	4613	A
234	4613	B

CLASS

c-id	c-name	units
4613	s.e.	2
4609	o.s.	2

SSN	c-name
123	s.e
234	s.e

o.s.: gone!

Outer join

TAKES

SSN	c-id	grade
123	4613	A
234	4613	B

CLASS

c-id	c-name	units
4613	s.e.	2
4609	o.s.	2

SSN	c-name
123	s.e
234	s.e.
null	o.s.

Outer join

select ssn, c-name

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

SSN	c-name
123	s.e
234	s.e.
null	o.s.

Outer join

- left outer join
- right outer join
- full outer join
- natural join

Null Values

- **null** -> unknown, or inapplicable, (or …)
- Complications:
  - 3-valued logic (true, false and *unknown*).
  - **null** = **null** : false!!

Overview - detailed - SQL

- DML
- other parts:
  - views
  - modifications
  - joins
  - DDL
  - constraints

Data Definition Language

```
create table student
(ssn char(9) not null,
  name char(30),
  address char(50),
primary key (ssn) )
```

Data Definition Language

create table r(A1 D1, …, An Dn,

integrity-constraint1,

…

integrity-constraint-n)

Data Definition Language

Domains:

- **char**(n), **varchar**(n)
- **int**, **numeric**(p,d), **real**, **double precision**
- **float, smallint**
- **date**, **time**

Data Definition Language

delete a table: difference between

drop table student

delete from student

Data Definition Language

modify a table:

alter table student **drop** address

alter table student **add** major char(10)

CONSTRAINTS

Data Definition Language

integrity constraints:

- **primary key**
- **foreign key**
- **check**(P)

Data Definition Language

create table takes

(ssn **char**(9) **not null**,

  c-id **char**(5) **not null**,

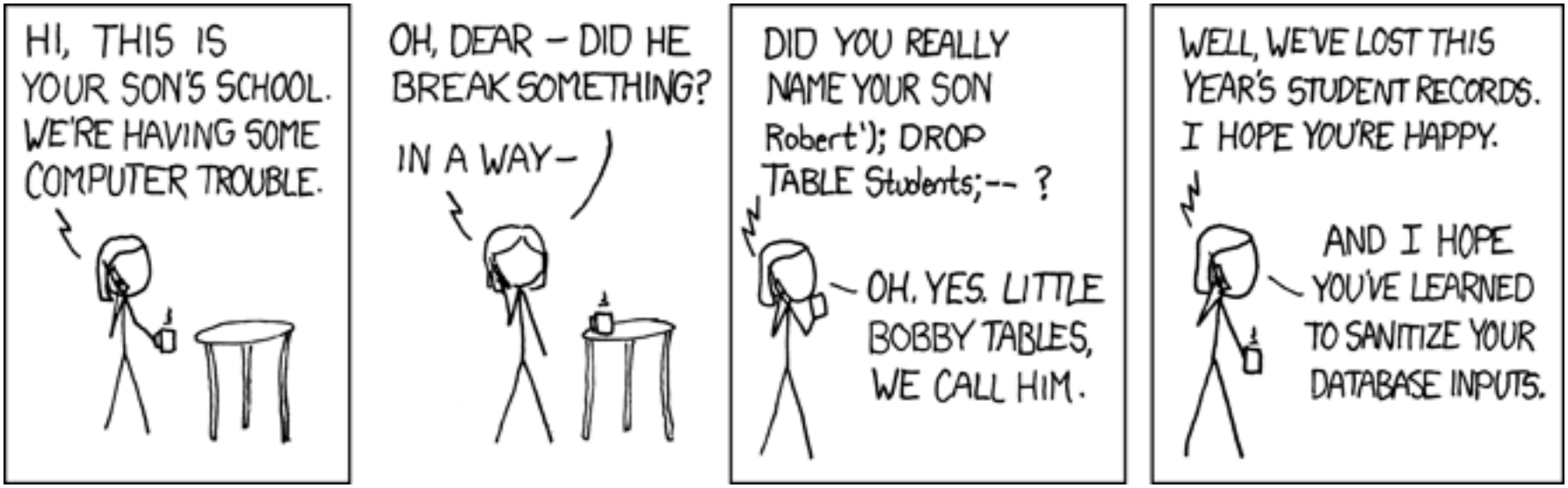
  grade **char**(1),

primary key (ssn, c-id),

check grade **in** (“A”, “B”, “C”, “D”, “F”))

Maintaining Integrity of Data

- Data is dirty.
- How does an application ensure that a database modification does not corrupt the tables?



Maintaining Integrity of Data

- Data is dirty.
- How does an application ensure that a database modification does not corrupt the tables?

- Two approaches:
  - Application programs check that database modifications are consistent.
  - Use the features provided by SQL.

Integrity Checking in SQL

- PRIMARY KEY and UNIQUE constraints.
- FOREIGN KEY constraints.
- Constraints on attributes and tuples.
- Triggers (schema-level constraints).

- How do we express these constraints?
- How do we check these constraints?
- What do we do when a constraint is violated?

Keys in SQL

- A set of attributes S is a key for a relation R if every pair of tuples in R disagree on at least one attribute in S.
- Select one key to be the PRIMARY KEY; declare other keys using UNIQUE.

Primary Keys in SQL

- Modify the schema of Students to declare PID to be the key.
  - CREATE TABLE Students(
    PID VARCHAR(8) PRIMARY KEY,
    Name CHAR(20), Address VARCHAR(255));

- What about Courses, which has two attributes in its key?
  - CREATE TABLE Courses(Number integer, DeptName:
    VARCHAR(8), CourseName VARCHAR(255), Classroom
    VARCHAR(30), Enrollment integer,
    PRIMARY KEY (Number, DeptName)
    );

Effect of Declaring PRIMARY KEYs

- Two tuples in a relation cannot agree on all the attributes in the key. DBMS will reject any action that inserts or updates a tuple in violation of this rule.

- A tuple cannot have a NULL value in a key attribute.

Other Keys in SQL

- If a relation has other keys, declare them using the UNIQUE keyword.
- Use UNIQUE in exactly the same places as PRIMARY KEY.

- There are two differences between PRIMARY KEY and UNIQUE:
  - A table may have only one PRIMARY KEY but more than one set of attributes declared UNIQUE.
  - A tuple may have NULL values in UNIQUE attributes.

Enforcing Key Constraints

- Upon which actions should an RDBMS enforce a key constraint?
- Only tuple update and insertion.
- RDMBS searches the tuples in the table to find if any tuple exists that agrees with the new tuple on all attributes in the primary key.
- To speed this process, an RDBMS automatically creates an efficient search index on the primary key.
- User can instruct the RDBMS to create an index on one or more attributes

Foreign Key Constraints

- Referential integrity constraint: in the relation Teach (that "connects" Courses and Professors), if Teach relates a course to a professor, then a tuple corresponding to the professor must exist in Professors.
- How do we express such constraints in Relational Algebra?
- Consider the Teach(ProfessorPID, Number, DeptName) relation.

  every non-NULL value of ProfessorPID inTeach must be a valid ProfessorPID in Professors.

- RA $\pi_{\text{ProfessorPID}}(\text{Teach}) \subseteq \pi_{\text{PID}}(\text{Professors})$.

Referential Integrity constraints

'foreign keys' - eg:

```
create table takes(
    ssn char(9) not null,
    c-id char(5) not null,
    grade integer,
    primary key(ssn, c-id),
    foreign key ssn references student,
    foreign key c-id references class)
```

Referential Integrity constraints

…

foreign key ssn **references** student,

foreign key c-id **references** class)

Effect:

- expects that ssn to exist in ‘student’ table
- blocks ops that violate that - how??
  - insertion?
  - deletion/update?

Requirements for FOREIGN KEYs

- If a relation R declares that some of its attributes refer to foreign keys in another relation S, then these attributes must be declared UNIQUE or PRIMARY KEY in S.

- Values of the foreign key in R must appear in the referenced attributes of some tuple in S.

Enforcing Referential Integrity

- Three policies for maintaining referential integrity.
- **Default policy**: reject violating modifications.
- **Cascade policy**: mimic changes to the referenced attributes at the foreign key.
- **Set-NULL policy**: set appropriate attributes to NULL.

Default Policy for Enforcing Referential Integrity

- Reject violating modifications. There are four cases.

- Insert a new Teach tuple whose ProfessorPID is not NULL and is not the PID of any tuple in Professors.
- Update the ProfessorPID attribute in a tuple in Teach to a value that is not the PID value of any tuple in Professors.
- Delete a tuple in Professors whose PID value is the ProfessorPID value for one or more tuples in Teach.
- Update the PID value of a tuple in Professors when the old PID value is the value of ProfessorPID in one or more tuples of Teach.

Cascade Policy for Enforcing Referential Integrity

- Only applies to deletions of or updates to tuples in the referenced relation (e.g., Professors).

- If we delete a tuple in Professors, delete all tuples in Teach that refer to that tuple.
- If we update the PID value of a tuple in Professors from p1 to p2, update all value of ProfessorPID in Teach that are p1 to p2.

Set-NULL Policy for Enforcing Referential Integrity

- Also applies only to deletions of or updates to tuples in the referenced relation (e.g., Professors).

- If we delete a tuple in Professors, set the ProfessorPID attributes of all tuples in Teach that refer to the deleted tuple to NULL.
- If we update the PID value of a tuple in Professors from p1 to p2, set all values of ProfessorPID in Teach that are p1 to NULL

Referential Integrity constraints in SQL

…

foreign key ssn **references** student

on delete cascade

on update cascade,

…

- -> eliminate all student enrollments
- other options (set to null, to default etc)

Constraining Attributes and Tuples

- SQL also allows us to specify constraints on attributes in a relation and on tuples in a relation.
  - Disallow courses with a maximum enrollment greater than 100.
  - A chairperson of a department must teach at most one course every semester.
- How do we express such constraints in SQL?
- How can we change our minds about constraints?
- A simple constraint: NOT NULL
  - Declare an attribute to be NOT NULL after its type in a CREATE TABLE statement.
  - Effect is to disallow tuples in which this attribute is NULL.

Attribute-Based CHECK Constraints

- Disallow courses with a maximum enrollment greater than 100.

- CREATE TABLE Courses(...

  Enrollment INT CHECK (Enrollment <= 100) ...);

- The condition can be any condition that can appear in a WHERE clause.
- CHECK statement may use a subquery to mention other attributes of the same or other relations.
- An attribute-based CHECK constraint is checked only when the value of that attribute changes.

Tuple-Based CHECK Constraints

- Tuple-based CHECK constraints are checked whenever a tuple is **inserted into or updated in a relation**.

- A chairperson of a department teaches at most one course in any semester.

```
CREATE TABLE Teach(...
        CHECK ProfessorPID NOT IN
            ((SELECT ProfessorPID FROM Teach)
            INTERSECT
            (SELECT ChairmanPID FROM Departments)
            )
            );
```

Weapons for IC:

- assertions
  - **create assertion** <assertion-name> **check** <predicate>
- triggers (~ assertions with ‘teeth’)
  - on operation, if condition, then action

Assertions: Example

- Can't have more courses than students ('Pigeonhole Principle')

```
CREATE ASSERTION FewStudents CHECK (
  (SELECT COUNT(*) FROM Students) <=
  (SELECT COUNT(*) FROM Courses)
);
```

Triggers: Motivation

- triggers (~ assertions with ‘teeth’)
  - on operation, if condition, then action

Triggers - example

define trigger zerograde **on update** takes

(**if new** takes.grade < 0

then takes.grade = 0)

Triggers - discussion

- more complicated: "managers have higher salaries than their subordinates" - a trigger can automatically boost mgrs salaries
- triggers: tricky (infinite loops…)

OK, what could have been done?