SQL: Part II

CPS 196.3
Introduction to Database Systems

Announcement

❖ Homework #1 due on September 12 in D327
❖ Project was assigned on September 8
❖ Sign up for the presentation!
  ❖ Have you checked news:duke.cs.cps196-3?

Incomplete information

❖ Example: Student (SID, name, age, GPA)
❖ Value unknown
  ❖ We do not know Nelson’s age
❖ Value not applicable
  ❖ Nelson has not taken any classes yet; what is his GPA?
Solution 1

- A dedicated special value for each domain (type)
  - GPA cannot be $-1$, so use $-1$ as a special value to indicate a missing or invalid GPA
  - Leads to incorrect answers if not careful
    - `SELECT AVG(GPA) FROM Student;`
  - Complicates applications
    - `SELECT AVG(GPA) FROM Student WHERE GPA <> -1;`
  - Remember the pre-Y2K bug?

Solution 2

- A valid-bit for every column
  - `Student (SID, name, name_is_valid, age, age_is_valid, GPA, GPA_is_valid)`
  - Too much overhead
  - Still complicates applications
    - `SELECT AVG(GPA) FROM Student WHERE GPA_is_valid;`

SQL’s solution

- A special value `NULL`
  - Same for every domain
  - Special rules for dealing with `NULL`’s

- Example: `Student (SID, name, age, GPA)`
  - `{ 789, "Nelson", NULL, NULL }`
Computing with NULL’s

- When we operate on a NULL and another value (including another NULL) using +, −, etc., the result is NULL
- Aggregate functions ignore NULL, except COUNT(∗) (since it counts rows)

Three-valued logic

- TRUE = 1, FALSE = 0, UKNOWN = 0.5
- x AND y = min(x, y)
- x OR y = max(x, y)
- NOT x = 1 − x
- When we compare a NULL with another value (including another NULL) using =, >, etc., the result is UKNOWN
- WHERE and HAVING clauses only select rows for output if the condition evaluates to TRUE
  - UKNOWN is in sufficient

Unfortunate consequences

- SELECT AVG(GPA) FROM Student;
  SELECT SUM(GPA)/COUNT(*) FROM Student;
- SELECT * FROM Student;
  SELECT * FROM Student WHERE GPA = GPA;
- Be careful: NULL breaks many equivalences
Another problem

- Example: Who has NULL GPA values?
  - SELECT * FROM Student WHERE GPA = NULL;

- Introduced built-in predicates IS NULL and IS NOT NULL
  - SELECT * FROM Student WHERE GPA IS NULL;

Outerjoin motivation

- Example: a master class list
  - SELECT c.CID, c.title, s_SID, s.name
    FROM Course c, Enroll e, Student s
    WHERE c.CID = e.CID AND e.SID = s.SID;
  - What if a class is empty?
  - It may be reasonable for the master class list to include empty classes as well
    - For these classes, SID and name columns would be NULL

Outerjoin flavors and definitions

- A full outerjoin between R and S (denoted \( R \bowtie S \)) includes all rows in the result of \( R \bowtie S \), plus
  - “Dangling” R rows (those that do not join with any S rows) padded with NULL’s for S’s columns
  - “Dangling” S rows (those that do not join with any R rows) padded with NULL’s for R’s columns
- A left outerjoin \( (R \bowtie S) \) includes rows in \( R \bowtie S \) plus dangling R rows padded with NULL’s
- A right outerjoin \( (R \bowtie S) \) includes rows in \( R \bowtie S \) plus dangling S rows padded with NULL’s
### Outerjoin examples

<table>
<thead>
<tr>
<th>Course</th>
<th>Enroll</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
<td>Title</td>
</tr>
<tr>
<td>CPS199</td>
<td>Independent Study</td>
</tr>
<tr>
<td>CPS130</td>
<td>Analysis of Algorithms</td>
</tr>
<tr>
<td>CPS114</td>
<td>Computer Networks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enroll</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>CID</td>
</tr>
<tr>
<td>142</td>
<td>CPS199 Independent Study</td>
</tr>
<tr>
<td>142</td>
<td>CPS114 Computer Networks</td>
</tr>
<tr>
<td>123</td>
<td>CPS196 NULL</td>
</tr>
<tr>
<td>142</td>
<td>NULL NULL</td>
</tr>
</tbody>
</table>

### Outerjoin syntax

- `SELECT * FROM Course LEFT OUTER JOIN Enroll ON Course.CID = Enroll.CID;`
- `SELECT * FROM Course RIGHT OUTER JOIN Enroll ON Course.CID = Enroll.CID;`
- `SELECT * FROM Course FULL OUTER JOIN Enroll ON Course.CID = Enroll.CID;`

- These queries return all columns in `Course` and `Enroll`, so they are not exactly `Course →* Enroll`, `Course ←* Enroll`, and `Course ↔* Enroll`, respectively.

### Summary of SQL features covered so far

- `SELECT-FROM-WHERE` statements
- Set and bag operations
- Table expressions, subqueries
- Aggregation and grouping
- Ordering
- NULL's and outerjoins

- Next: data modification statements, constraints
INSERT

- Insert one row
  - `INSERT INTO Enroll VALUES (456, 'CPS196');`

- Insert the result of a query
  - `INSERT INTO Enroll
    (SELECT SID, 'CPS196' FROM Student
     WHERE SID NOT IN (SELECT SID FROM Enroll
      WHERE CID = 'CPS196'));`

DELETE

- Delete everything
  - `DELETE FROM Enroll;`

- Delete according to a WHERE condition
  - Example: Student 456 drops CPS196
    - `DELETE FROM Enroll
      WHERE SID = 456 AND CID = 'CPS196';`
  - Example: Drop students with GPA lower than 1.0 from all CPS classes
    - `DELETE FROM Enroll
      WHERE SID IN (SELECT SID FROM Student
       WHERE GPA < 1.0)
      AND CID LIKE 'CPS%';`

UPDATE

- Example: Student 142 changes name to "Barney"
  - `UPDATE Student
    SET name = 'Barney'
    WHERE SID = 142;`

- Example: Let's be “fair”?
  - `UPDATE Student
    SET GPA = (SELECT AVG(GPA) FROM Student);`
Constraints

- Restrictions on allowable data in a database
  - In addition to the simple structure and type restrictions imposed by the table definitions
  - Declared as part of the schema
  - Enforced by the DBMS
- Why use constraints?
  - Protect data integrity (catch errors)
  - Tell the DBMS about the data (so it can optimize better)

Types of SQL constraints

- **NOT NULL**
- **Key**
- **Referential integrity (foreign key)**
- **General assertion**
- **Tuple- and attribute-based** CHECK's

**NOT NULL constraint examples**

- CREATE TABLE Student
  (SID INTEGER NOT NULL,
   name VARCHAR(30) NOT NULL,
   email VARCHAR(30),
   age INTEGER,
   GPA FLOAT);
- CREATE TABLE Course
  (CID CHAR(10) NOT NULL,
   title VARCHAR(100) NOT NULL);
- CREATE TABLE Enroll
  (SID INTEGER NOT NULL,
   CID CHAR(10) NOT NULL);
Key declaration

- At most one PRIMARY KEY per table
  - Typically implies a primary index
  - Rows are stored inside the index, typically sorted by the primary key value
- Any number of UNIQUE keys per table
  - Typically implies a secondary index
  - Pointers to rows are stored inside the index

Key declaration examples

- CREATE TABLE Student
  
  ```sql
  CREATE TABLE Student
  (SID INTEGER NOT NULL PRIMARY KEY,
   name VARCHAR(30) NOT NULL,
   email VARCHAR(30) UNIQUE,
   age INTEGER,
   GPA FLOAT);
  ```

- CREATE TABLE Course
  
  ```sql
  CREATE TABLE Course
  (CID CHAR(10) NOT NULL PRIMARY KEY,
   title VARCHAR(100) NOT NULL);
  ```

- CREATE TABLE Enroll
  
  ```sql
  CREATE TABLE Enroll
  (SID INTEGER NOT NULL,
   CID CHAR(10) NOT NULL,
   PRIMARY KEY(SID, CID));
  ```

Referential integrity example

- Enroll.SID references Student.SID
  - If an SID appears in Enroll, it must appear in Student
- Enroll.CID references Course.CID
  - If a CID appears in Enroll, it must appear in Course

  That is, no "dangling pointers"
### Referential integrity in SQL
- Referenced column(s) must be PRIMARY KEY
- Referencing column(s) form a FOREIGN KEY
- Example
  ```sql
  CREATE TABLE Enroll
  (SID INTEGER NOT NULL REFERENCES Student(SID),
   CID CHAR(10) NOT NULL,
   PRIMARY KEY(SID, CID),
   FOREIGN KEY CID REFERENCES Course(CID));
  ```

### Enforcing referential integrity
- Example: `Enroll.SID` references `Student.SID`
- Insert or update an `Enroll` row so it refers to a non-existent SID
- Delete or update a `Student` row whose SID is referenced by some `Enroll` row

### Deferred constraint checking
- No-chicken-no-egg problem
  ```sql
  CREATE TABLE Dept
  (name CHAR(20) NOT NULL PRIMARY KEY,
   chair CHAR(30) NOT NULL REFERENCES Prof(name));
  CREATE TABLE Prof
  (name CHAR(30) NOT NULL PRIMARY KEY,
   dept CHAR(20) NOT NULL REFERENCES Dept(name));
  ```
- Deferred constraint checking is necessary
  - Check only at the end of a transaction
  - Allowed in SQL as an option
General assertion

- CREATE ASSERTION assertion_name
  
  CHECK assertion_condition;

- *assertion_condition* is checked for each modification that could potentially violate it

- Example: *Enroll.SID* references *Student.SID*
  
  - CREATE ASSERTION EnrollStudentRefIntegrity
    
    CHECK (
    );

  - In SQL3, but not all (perhaps no) DBMS supports it

Tuple- and attribute-based CHECK's

- Associated with a single table

- Only checked when a tuple or an attribute is inserted or updated

- Example:
  
  - CREATE TABLE Enroll
    
    (SID INTEGER NOT NULL
    
    CHECK (SID IN (SELECT SID FROM Student)),
    CID ...);

  - Is it a referential integrity constraint?

Summary of SQL features covered so far

- SELECT-FROM-WHERE statements

- Set and bag operations

- Table expressions, sub-queries

- Aggregation and grouping

- Ordering

- NULL's and outerjoins

- INSERT/DELETE/UPDATE

- Constraints

  - Next: triggers, views, indexes