SQL

CPS 216
Advanced Database Systems

Review

SELECT [DISTINCT]...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...;

ORDER BY

• SELECT [DISTINCT] $E_1$, $E_2$, $E_3$...
FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY $E_i$ [ASC | DESC], $E_{i+1}$ [ASC | DESC], ...;
• ASC = ascending, DESC = descending
• Operational semantics
  – After SELECT list has been computed and optional duplicate elimination has been carried out, sort the output according to ORDER BY specification
ORDER BY example

- List all students, sort them by GPA (descending) and then name (ascending)
  
  - ASC is the default option
  - Technically, only output columns can appear in ORDER BY clause (some DBMS support more)
  - Can use output index instead

Data modification: INSERT

- Insert one row
  Example: Student 456 takes CPS 216
  - INSERT INTO Enroll VALUES (456, 'CPS 216');

- Insert the result of a query
  Example: Force everybody to take CPS 216
  - INSERT INTO Enroll
    (SELECT SID, 'CPS 216' FROM Student)

Data modification: DELETE

- Delete everything
  - DELETE FROM Enroll;

- Delete according to a WHERE condition
  Example: Student 456 drops CPS 216
  - DELETE FROM Enroll
    WHERE SID = 456 AND CID = 'CPS 216';

  Example: Drop students with GPA lower than 1.0 from all CPS classes
  -
Data modification: 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);

Views

- A view is like a virtual table
  - Defined by a query, which describes how to compute
    the view contents on the fly
  - DBMS stores the view definition query instead of
    view contents
  - Can be used in queries just like a regular table

Creating and dropping views

- Example: CPS 216 roster
  - CREATE VIEW CPS216Roster AS
    SELECT SID, name, age, GPA
    FROM Student
    WHERE SID IN (SELECT SID FROM Enroll
    WHERE CID = 'CPS 216');
- To drop a view (or table)
  - DROP VIEW view_name;
  - DROP TABLE table_name;
Using views in queries

- Example: find the average GPA of CPS 216 students
  - SELECT AVG(GPA) FROM CPS216Roster;
  - To process the query, replace the reference to the view by its definition

Why use views?

- To hide data from users
- To hide complexity from users
- Logical data independence
  - If applications deal with views, we can change the underlying schema without affecting applications
  - Recall physical data independence: change the physical organization of data without affecting applications
- Real database applications use tons of views

Modifying views

- Doesn’t seem to make sense since views are virtual
- But does make sense if that’s how users view the database
- Goal: modify the base tables such that the modification would appear to have been accomplished on the view
A simple case

CREATE VIEW StudentGPA AS
  SELECT SID, GPA FROM Student;
DELETE FROM StudentGPA WHERE SID = 123;
translates to:

An impossible case

CREATE VIEW HighGPAStudent AS
  SELECT SID, GPA FROM Student
  WHERE GPA > 3.7;
INSERT INTO HighGPAStudent
  VALUES(987, 2.5);

A case with too many possibilities

CREATE VIEW AverageGPA(GPA) AS
  SELECT AVG(GPA) FROM Student;
  -- Note that you can rename columns in view definition
UPDATE AverageGPA SET GPA = 2.5;
SQL92 updatable views

- Single-table SFW
  - No aggregation
  - No subqueries

- Overly restrictive
- Still gets it wrong in some cases
  - See the slide titled “An impossible case”

Incomplete information

- Example: Student (SID, name, age, GPA)

- Value unknown

- Value not applicable

Solution 1

- A dedicated special value for each domain
  - GPA cannot be –1, so use –1 as a special value
  - SELECT AVG(GPA) FROM Student;

  SELECT AVG(GPA) FROM Student
  WHERE GPA <> 0;
  • Complicates applications
  - Remember the pre-Y2K bug?
    • 09/09/99 was used as an invalid or missing date value
    • It’s tricky to make these assumptions!
Solution 2

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

SQL’s solution

- A special value NULL
  - Same for every domain
  - Special rules for dealing with NULLs

- Example: Student (SID, name, age, GPA)
  - <789, 'Nelson', NULL, NULL>

Computing with NULLs

- When we operate on a NULL and another value (including another NULL) using +, -, etc., the result is NULL

- Aggregate functions ignore NULL, except COUNT(*)
Three-valued logic

- TRUE = 1, FALSE = 0, UNKNOWN = 0.5
- $x \text{ AND } y = \min(x, y)$
- $x \text{ OR } y = \max(x, y)$
- $\text{NOT}(x) = 1 - x$
- When we compare a NULL with another value (including another NULL) using $=, >, \text{etc.}$, the result is UNKNOWN
- WHERE and HAVING clauses only select tuples if the condition evaluates to TRUE
  – UNKNOWN is insufficient

Unfortunate consequences

- select avg(GPA) from Student;
  select sum(GPA) / count(*) from Student;

- select * from Student;
  select * from Student
  where GPA > 3.0 or GPA <= 3.0;

- Be careful: NULL breaks many equivalences

Another problem

- Example: Who has NULL GPA values?

  – New built-in predicates IS NULL and IS NOT NULL
  select * from Student where GPA is null;
Recap

- Covered
  - ORDER BY
  - Data modification statements
  - Views
  - NULLs
- Skipped
  - Outerjoin
  - Alternative join syntax
  - Schema modification statements
- Next
  - Constraints

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
  - Tell the DBMS about the data

Types of constraints

- NOT NULL
- Key
- Referential integrity
- General assertion
- Tuple- and attribute-based CHECKs
NOT NULL constraint example

• 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
    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
   (SID integer not null primary key,
    name varchar(30) not null,
    email varchar(30) unique,
    age integer, GPA float);
• create table Course
   (CID char(10) not null primary key,
    title varchar(100) not null);
• create table Enroll
   (SID integer not null, CID char(10) not null,
    primary key(SID, CID));

Works on Oracle
but not DB2:
DB2 requires UNIQUE
key columns
Referential integrity example

– Enroll.SID references Student.SID
– Enroll.CID references Course.SID
– If an SID appears in Enroll, it must appear in Student
– If a CID appears in Enroll, it must appear in Course
– That is, no “dangling pointers”

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>Enroll.CID</th>
<th>Course.CID</th>
<th>Course.CID title</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>CPS 216</td>
<td>CPS 210</td>
<td>Advanced Data</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>CPS 214</td>
<td>CPS 214</td>
<td>Analysis of Alg.</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>CPS 216</td>
<td>CPS 214</td>
<td>Computer Net.</td>
</tr>
<tr>
<td>406</td>
<td>Ralph</td>
<td>CPS 216</td>
<td>CPS 214</td>
<td>...</td>
</tr>
</tbody>
</table>

Referential integrity in SQL

- Referenced column must be PRIMARY KEY
- Referencing column is called FOREIGN KEY
- Example declaration
  - 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 a Enroll tuple so it refers to a non-existent SID
  - Reject
- Delete or update a Student tuple whose SID is referenced by some Enroll tuple
  - All three options can be specified in SQL
Deferred constraint checking

- No-chicken-no-egg problem
  - 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 ()
- SQL3, but not all (perhaps no) DBMS supports it

Tuple- and attribute-based CHECKs

- 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?
Next time

Transactions!