

## SQL: Part II

CPS 196.3  
Introduction to Database Systems

## 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?
    - 09/09/99 was used as a missing or invalid date value

## 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*)
  - $\langle 789, \text{"Nelson"}, \text{NULL}, \text{NULL} \rangle$

## 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

7

- ❖ TRUE = 1, FALSE = 0, UNKNOWN = 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 UNKNOWN
- ❖ WHERE and HAVING clauses only select rows for output if the condition evaluates to TRUE
  - UNKNOWN is insufficient

## Unfortunate consequences

8

- ❖ SELECT AVG(GPA) FROM Student;  
SELECT SUM(GPA)/COUNT(\*) FROM Student;
    - Not equivalent
    - Although  $AVG(GPA) = SUM(GPA)/COUNT(GPA)$  still
  - ❖ SELECT \* FROM Student;  
SELECT \* FROM Student  
WHERE GPA > 3.0 OR GPA <= 3.0;
    - Not equivalent
- ☞ Be careful: NULL breaks many equivalences

## Another problem

9

- ❖ Example: Who has NULL GPA values?
  - SELECT \* FROM Student WHERE GPA = NULL;
    - Does not work; never returns anything
  - (SELECT \* FROM Student)  
EXCEPT ALL  
(SELECT \* FROM Student  
WHERE GPA = 0 OR GPA <> 0)
    - Works, but ugly
  - Introduced built-in predicates IS NULL and IS NOT NULL
    - SELECT \* FROM Student WHERE GPA IS NULL;

## Outerjoin motivation

10

- ❖ 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

11

- ❖ 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 ( $R \ltimes S$ ) includes rows in  $R \bowtie S$  plus dangling  $R$  rows padded with NULL’s
- ❖ A right outerjoin ( $R \rtimes S$ ) includes rows in  $R \bowtie S$  plus dangling  $S$  rows padded with NULL’s

## Outerjoin examples

12

Course		Course $\bowtie$ Enroll		
CID	title	CID	title	SID
142	CPS196	CPS199	Independent Study	NULL
142	CPS114	CPS130	Analysis of Algorithms	857
123	CPS196	CPS114	Computer Networks	142
857	CPS196	CPS114	Computer Networks	456
857	CPS130	CID	title	SID
456	CPS114	CPS196	NULL	142
		CPS114	Computer Networks	142
		CPS196	NULL	123
		CPS196	NULL	857
		CPS130	Analysis of Algorithms	857
		CPS114	Computer Networks	456
		CPS114	Computer Networks	456
		CPS196	NULL	142
		CPS196	NULL	123
		CPS196	NULL	857

## Outerjoin syntax

13

- ❖ `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*  $\bowtie$  *Enroll*, *Course*  $\ltimes$  *Enroll*, and *Course*  $\ltimes$  *Enroll*, respectively

## Summary of SQL features covered so far

14

- ❖ 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

15

- ❖ Insert one row
  - `INSERT INTO Enroll VALUES (456, 'CPS196');`
    - Student 456 takes 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'));`
    - Force everybody to take CPS196

## DELETE

16

- ❖ 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

17

- ❖ 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);`
    - But update of every row causes average GPA to change!
    - Average GPA is computed over the old Student table

## Constraints

18

- ❖ 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

19

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

## NOT NULL constraint examples

20

- ❖ 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

21

- ❖ 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

22

- ❖ CREATE TABLE Student  
(SID INTEGER NOT NULL PRIMARY KEY,  
name VARCHAR(30) NOT NULL,  
email VARCHAR(30) UNIQUE, ← Works on Oracle  
but not DB2:  
DB2 requires UNIQUE  
key columns  
to be NOT NULL
  - ❖ 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));
- ↑  
This form is required for multi-attribute keys

## Referential integrity example

23

- ❖ 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"

Student				Enroll		Course	
SID	name	age	GPA	SID	CID	CID	title
142	Barb	10	2.3	142	CPS196	CPS196	Intro. to Database Systems
123	Milhouse	10	3.1	142	CPS114	CPS130	Analysis of Algorithms
857	Lisa	8	4.3	123	CPS196	CPS114	Computer Networks
456	Ralph	8	2.3	857	CPS196	...	...
...	...	...	...	857	CPS130	...	...
...	...	...	...	456	CPS114	...	...
...	...	...	...	...	...	...	...

## Referential integrity in SQL

24

- ❖ Referenced column(s) must be PRIMARY KEY
- ❖ Referencing column(s) form a FOREIGN KEY
- ❖ Example
  - 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

25

Example: *Enroll.SID* references *Student.SID*

- ❖ Insert or update an *Enroll* row so it refers to a non-existent SID
  - Reject
- ❖ Delete or update a *Student* row whose SID is referenced by some *Enroll* row
  - Reject
  - Cascade: ripple changes to all referring rows
  - Set NULL: set all references to NULL
  - All three options can be specified in SQL

## Deferred constraint checking

26

- ❖ 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));
```
  - The first INSERT will always violate a constraint
- ❖ Deferred constraint checking is necessary
  - Check only at the end of a transaction
  - Allowed in SQL as an option

## General assertion

27

- ❖ 

```
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 (NOT EXISTS
(SELECT * FROM Enroll
WHERE SID NOT IN
(SELECT SID FROM Student)));
```
- ☞ In SQL3, but not all (perhaps no) DBMS supports it

## Tuple- and attribute-based CHECK's

28

- ❖ 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?
  - Not quite; not checked when *Student* is modified

## Summary of SQL features covered so far

29

- ❖ Query
  - SELECT-FROM-WHERE statements
  - Set and bag operations
  - Table expressions, subqueries
  - Aggregation and grouping
  - Ordering
  - Outerjoins
- ❖ Modification
  - INSERT/DELETE/UPDATE
- ❖ Constraints

☞ Next: triggers, views, indexes