CSCB20 – Week 4

Introduction to Database and Web Application Programming

Anna Bretscher
Winter 2017

Last Week

Intro to SQL and MySQL

Mapping Relational Algebra to SQL queries

Focused on queries to start – assumed tables and database exist.

Creating tables, setting constraints…

This Week

- Creating tables, setting constraints…
- Inserting and updating tables
- More query commands
  - HAVING clause
  - LIKE clause
  - IN clause
  - UNION, INTERSECT
  - CASE

This Week – Time Permitting

- Creating views
- Outer Joins
  - Left
  - Right
  - Full
- More on NULL values

Null Value

Every type can have the special value null.

A value of null indicates the value is unknown or that it may not exist at all.

Sometimes we do not want a null value at all – we can add such a constraint.

Creating a Table

SQL Notation:

```
CREATE TABLE table_name
            (col_name1 type1,
            col_name2 type2,
            … ,
            col_namen typen,
            <integrity-constraint1>,
            … ,
            <integrity-constraintk>);
```

Integrity Constraints

Primary key(list of attributes) :

These attributes form the primary keys for the relation. Primary keys must be *non-null* and *unique*.

Foreign key(list of attributes) references *s* :

The values of these attributes for any tuple in the relation must correspond to values of the *primary key attributes* of some tuple in relation *s*.

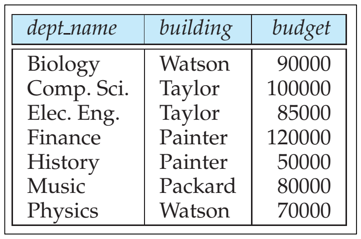
not null:

Specifies that this attribute may not have the *null value*. We list this constraint when defining the type of the attribute.

Examples

```
CREATE TABLE department
        (dept_name       VARCHAR(20),
         building        VARCHAR(15),
         budget          NUMERIC(12,2),
         PRIMARY KEY (dept_name));

CREATE TABLE course
        (course_id       VARCHAR(7),
         title           VARCHAR(50),
         dept_name       VARCHAR(20),
         credit          NUMERIC(2,0),
         PRIMARY KEY (course_id),
         FOREIGN KEY (dept_name) REFERENCES department );
```



dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Editing Tables

DROP TABLE table_name;	remove the table
DELETE FROM table_name WHERE predicate;	delete tuples satisfying the predicate
ALTER TABLE table_name ADD column type;	add a column
ALTER TABLE table_name DROP column;	remove a column

○ ○

Inserting

In MySQL we can insert into a table with the command:

```
INSERT INTO table_name
        VALUES (value1, value2, …, valuen);
```

OR

```
INSERT INTO table_name (col1, col2, …, coln)
        VALUES (value1, value2, …, valuen);
```

OR

```
INSERT INTO table_name
        SELECT QUERY
```

For example:

```
INSERT INTO instructor
        SELECT ID, name, dept_name, 18000
        FROM student
        WHERE dept_name = 'Music' AND tot_cred > 144;
```

○ ○

Updating

In MySQL we can update a table with the command:

```
        UPDATE table_name
                SET attribute = new_value
```

OR

```
        UPDATE table_name
                SET attribute = new_value
                WHERE predicate or select statement;
```

OR

```
        UPDATE table_name
                SET attribute = CASE
                                WHEN predicate_1 THEN result_1
                                WHEN predicate_2 THEN result_2
                                        …
                                WHEN predicate_n THEN result_n
                                ELSE   result_0
                              END
```

Views

- A view is a *virtual relation.*
- A view is defined in terms of stored tables (called base tables) and other views.
- Access a view like any base table.
- *Materialized views* exist, but are actually constructed and stored. Expensive to maintain!
- We'll use only virtual views.

Creating Views

CREATE VIEW view_name AS SELECT STATEMENT;

CREATE VIEW view_name(col_nam$_1$, col_name$_2$, …, col_name$_{k)}$
AS SELECT STATEMENT;

CREATE VIEW faculty AS SELECT ID, name, dept_name
FROM instructor;

We can now use view faculty as we would a table.

Every time the view is used, it is reconstructed.

Why Use Views

Allow us to break down a large query.

Make available specific category of data a particular user.

Gives another way to think about the data.

Q. Why is it good that views are virtual?

A. If a table is changed the corresponding view is changed appropriately.

Outer Joins

What does the following query return?

```
SELECT * FROM student  INNER JOIN takes
        ON student.id = takes.id;
```

We would like it to return every student and the courses they are taking.

Q. What about students who have not yet taken any courses?

A. They are left out.

Dangling Tuples

When JOINs require some attributes to match, tuples lacking a match are left out.

These tuples are said to be "*dangling*".

OUTER JOINs preserve dangling tuples by padding them with NULL in the other relation.

INNER JOINs do not pad with NULL.

Outer Joins

Use OUTER JOINS to prevent this loss of information.

The LEFT OUTER JOIN preserves tuples only in the relation to the left of the JOIN.

The RIGHT OUTER JOIN preserves tuples only in the relation to the right of the JOIN.

The FULL OUTER JOIN preserves tuples in both relations.*

* MySQL does not support FULL OUTER JOIN, but we can emulate by doing the UNION of a LEFT and a RIGHT.

JOIN Examples

A	B
1	2
4	5

R

B	C
2	3
6	7

S

R NATURAL JOIN S

A	B	C
1	2	3

JOIN Examples

A	B
1	2
4	5

R

B	C
2	3
6	7

S

R NATURAL LEFT JOIN S

A	B	C
1	2	3
4	5	NULL

JOIN Examples

A	B
1	2
4	5

R

B	C
2	3
6	7

S

R NATURAL RIGHT JOIN S

A	B	C
1	2	3
NULL	6	7

JOIN Examples

A	B
1	2
4	5

R

B	C
2	3
6	7

S

R NATURAL FULL JOIN S

OR

(R NATURAL LEFT JOIN S) UNION (R NATURAL RIGHT JOIN S)

A	B	C
1	2	3
4	5	NULL
NULL	6	7

JOIN Recap

A JOIN B ON C	inner join
A {LEFT | RIGHT | FULL} JOIN B ON C	outer join
A NATURAL JOIN B	natural inner join
A NATURAL {LEFT | RIGHT | FULL} JOIN B	natural outer join

NULL

We can check for NULL values using:

IS NULL

IS NOT NULL

Because we have NULL, we need three truth values for comparisons:

TRUE, FALSE and UNKNOWN

If one or both operands is NULL, the comparison always evaluates to UNKNOWN.

Otherwise, comparisons evaluate to TRUE and FALSE.

Booleans and UNKNOWN

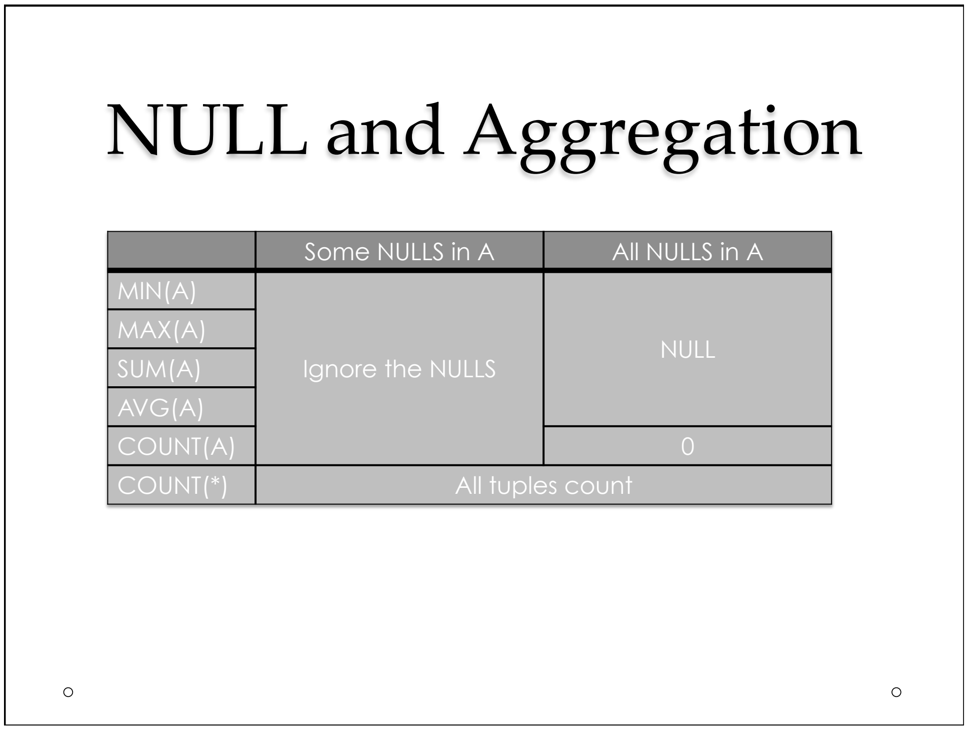
What is NOT UNKNOWN?
UNKNOWN.

What is TRUE AND UNKNOWN?
UNKNOWN.

What is TRUE OR UNKNOWN?
TRUE.

WHAT IS FALSE AND UNKNOWN?
FALSE.

WHAT IS FALSE OR UNKNOWN?
UNKNOWN.



NULL and Aggregation

	Some NULLS in A	All NULLS in A
MIN(A)	Ignore the NULLS	NULL
MAX(A)		
SUM(A)		
AVG(A)		
COUNT(A)		0
COUNT(*)	All tuples count	