CMSC424: Database Design
SQL

February 10, 2020

Instructor: Amol Deshpande
amol@cs.umd.edu

Today's Plan

- Reading Homework 2

- SQL (Chapter 3)
  - Null values (3.6)
  - Aggregates (3.7)
  - Views (4.2)
  - Transactions (4.3)
  - Integrity Constraints (4.4)
  - Triggers (5.3)

SQL: Nulls

The “dirty little secret” of SQL

(major headache for query optimization)

Can be a value of any attribute

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

What does this mean?

(unknown) We don’t know Waltham’s assets?

(inapplicable) Waltham has a special kind of account without assets

(withheld) We are not allowed to know

SQL: Nulls

Arithmetic Operations with `Null`

`n + NULL = NULL` (similarly for all *arithmetic ops*: `+, -, *, /, mod, …`)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT bname, assets * 2 as a2
FROM branch
```

=

bname	a2
Downtown	18M
Perry	3.4M
Mianus	.8M
Waltham	NULL

SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets = NULL
```

=

bname	bcity	assets

Counter-intuitive: NULL * 0 = NULL

Counter-intuitive: select * from movies
where length >= 120 or length <= 120

SQL: Nulls

Boolean Operations with `Null`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

e.g: branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

```
SELECT *
FROM branch
WHERE assets IS NULL
```

=

bname	bcity	assets
Waltham	Boston	NULL

SQL: Unknown

Boolean Operations with `Unknown`

`n < NULL = UNKNOWN` (similarly for all *boolean ops*: `>, <=, >=, <>, =, …`)

```
FALSE OR UNKNOWN = UNKNOWN
TRUE AND UNKNOWN = UNKNOWN
```

Intuition: substitute each of TRUE, FALSE for unknown. If different answer results, results is unknown

```
UNKNOWN OR UNKNOWN = UNKNOWN
UNKNOWN AND UNKNOWN = UNKNOWN
NOT (UNKNOWN) = UNKNOWN
```

Can write:

```
SELECT …
FROM …
WHERE booleanexp IS UNKNOWN
```

UNKNOWN tuples are not included in final result

Today's Plan

- Reading Homework 2

- SQL (Chapter 3)
  - Null values (3.6)
  - Aggregates (3.7)
  - Views (4.2)
  - Transactions (4.3)
  - Integrity Constraints (4.4)
  - Triggers (5.3)

Aggregates

Other common aggregates:
max, min, sum, count, stdev, …

select count (**distinct** *ID*)
from *teaches*
where *semester* = ’Spring’ **and** *year* = 2010

Find the average salary of instructors in the Computer Science
select *avg(**salary)***
from *instructor*
where *dept_name = ‘Comp. Sci’;*

Can specify aggregates in any query.

Find max salary over instructors teaching in S’10
select max(*salary)*
from *teaches* **natural join** *instructor*
where *semester* = ’Spring’ **and** *year* = 2010;

Aggregate result can be used as a scalar.
Find instructors with max salary:
select *
from *instructor*
where *salary* **= (select max***(salary)* **from** *instructor);*

Aggregates

Aggregate result can be used as a scalar.
Find instructors with max salary:

select *

from *instructor*

where *salary* **= (select max**(*salary*) **from** *instructor*);

Following doesn't work:

select *

from *instructor*

where *salary* **= max**(*salary*);

select *name*, **max**(*salary*)

From instructor;

Aggregates: Group By

Split the tuples into groups, and computer the aggregate for each group

select *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*;

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregates: Group By

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 3.8 The natural join of the *instructor* relation with the *teaches* relation.

Output will have 3 tuples:

Summer, ….
Fall, ….
Spring, …

Aggregates: Group By

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 3.8 The natural join of the *instructor* relation with the *teaches* relation.

Output will have 2 tuples:

2009,
2010,

Aggregates: Group By

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Figure 3.8 The natural join of the *instructor* relation with the *teaches* relation.

Output will have 7 tuples:

Comp. Sci,
Finance,
Music,
Physics,
History,
Biology,
Elec. Eng.,

Aggregates: Group By

Attributes in the select clause must be aggregates, or must appear in the group by clause. Following wouldn't work

select *dept_name*, ID, **avg** (*salary*)
from *instructor*
group by *dept_name*;

"having" can be used to select only some of the groups.

select *dept_name*
from *instructor*
group by *dept_name*
having avg(*salary***)** *> 42000*

Aggregates and NULLs

Given

branch =

bname	bcity	assets
Downtown	Boston	9M
Perry	Horseneck	1.7M
Mianus	Horseneck	.4M
Waltham	Boston	NULL

Aggregate Operations

```
SELECT SUM (assets) =
FROM branch
```

SUM
11.1 M

`NULL` *is ignored for SUM*

Same for `AVG` *(3.7M),* `MIN` *(0.4M),* `MAX` *(9M)*

Also for COUNT(assets) -- returns 3

But `COUNT (*)` *returns*

COUNT
4

Aggregates and NULLs

Given

branch =

bname	bcity	assets

```
SELECT SUM (assets) =
FROM branch
```

SUM
NULL

- *Same as* `AVG, MIN, MAX`
- *But* `COUNT (assets)` *returns*

COUNT
0

Today's Plan

- Reading Homework 2

- SQL (Chapter 3)
  - Null values (3.6)
  - Aggregates (3.7)
  - Views (4.2)
  - Transactions (4.3)
  - Integrity Constraints (4.4)
  - Triggers (5.3)

Today's Plan

- Reading Homework 2

- SQL (Chapter 3)
  - Null values (3.6)
  - Aggregates (3.7)
  - Views (4.2)
  - Transactions (4.3)
  - Integrity Constraints (4.4)
  - Triggers (5.3)

Views

- Provide a mechanism to hide certain data from the view of certain users. To create a view we use the command:

create view *v* **as** <query expression>

where:

<query expression> is any legal expression

The view name is represented by *v*

- Can be used in any place a normal table can be used
- For users, there is no distinction in terms of using it

Example Queries

- A view consisting of branches and their customers

create view *all-customers* **as**
(**select** *branch-name, customer-name*
from *depositor, account*
where *depositor.account-number = account.account-number)*
union
(**select** *branch-name, customer-name*
from *borrower, loan*
where *borrower.loan-number = loan.loan-number)*

Find all customers of the Perryridge branch

select *customer-name*
from *all-customers*
where *branch-name =* 'Perryridge'

Today's Plan

- Reading Homework 2

- SQL (Chapter 3)
  - Null values (3.6)
  - Aggregates (3.7)
  - Views (4.2)
  - Transactions (4.3)
  - Integrity Constraints (4.4)
  - Triggers (5.3)

Next:

- Integrity constraints
- ??
- Prevent semantic inconsistencies

IC's

- Predicates on the database
- Must always be true (checked whenever db gets updated)

- There are the following 4 types of IC's:
  - Key constraints (1 table)

    e.g., *2 accts can't share the same acct_no*
  - Attribute constraints (1 table)

    e.g., *accts must have nonnegative balance*
  - Referential Integrity constraints (2 tables)

    E.g. *bnames* associated w/ *loans* must be names of real branches
  - Global Constraints (*n* tables)

    E.g., all *loans* must be carried by at least 1 *customer* with a savings acct

Key Constraints

Idea: specifies that a relation is a set, not a bag

SQL examples:

1. Primary Key:

```
CREATE TABLE branch(
        bname  CHAR(15)  PRIMARY KEY,
        bcity      CHAR(20),
        assets     INT);
```

or

```
CREATE TABLE depositor(
        cname   CHAR(15),
        acct_no  CHAR(5),
         PRIMARY KEY(cname, acct_no));
```

2. Candidate Keys:

```
CREATE TABLE customer (
        ssn     CHAR(9)    PRIMARY KEY,
        cname  CHAR(15),
        address CHAR(30),
        city          CHAR(10),
         UNIQUE (cname, address, city));
```