SQL – Data Manipulation Language

Carla Teixeira Lopes

Bases de Dados
Mestrado Integrado em Engenharia Informática e Computação, FEUP

Based on Jennifer Widom slides

Group-by versus subqueries

```
SELECT cName, count(*) AS cnt
FROM Apply
GROUP BY cName;
```

College(cName, state, enr)
Student(sID, sName, GPA, sizeHS)
Apply(sID, cName, major, decision)

cName	cnt
Berkeley	3
Cornell	6
MIT	4
Stanford	6

```
SELECT DISTINCT cName,
        (SELECT count(*)
         FROM Apply A2
         WHERE A2.cName = A1.cName) AS cnt
FROM Apply A1;
```

Group-by versus subqueries

College(cName, state, enr)
Student(sID, sName, GPA, sizeHS)
Apply(sID, cName, major, decision)

```
SELECT cName
FROM Apply
GROUP BY cName
HAVING count(*) < 5;
```


No need for DISTINCT: automatically from GROUP BY

cName
Berkeley
MIT

```
SELECT DISTINCT cName
FROM Apply A1
WHERE 5 > (SELECT count(*) FROM Apply A2 WHERE A2.cName =
          A1.cName);
```

Every *group by* and *having* query can be written without using those clauses

Group-by versus subqueries

College(cName, state, enr)
Student(sID, sName, GPA, sizeHS)
Apply(sID, cName, major, decision)

```
SELECT cName
FROM Apply
GROUP BY cName
HAVING count(*) < 5;
```

This is SQL by an expert

```
SELECT DISTINCT cName
FROM Apply A1
WHERE 5 > (SELECT count(*)
           FROM Apply A2
           WHERE A2.cName = A1.cName);
```

This is SQL by a novice

Which way is more efficient?

- How many times do we do a SFW query in each case?
- With GROUP BY can be much more efficient!

Aggregation summary

SELECT	S	→ attributes $a_1,\ldots,a_k$ and/or aggregates over other attributes
FROM	$R_1,\ldots,R_n$	
WHERE	C_1	→ any condition on the attributes in $R_1,\ldots,R_n$
GROUP BY	$a_1,\ldots,a_k$	
HAVING	C_2	→ any condition on the aggregate expressions

Evaluation steps

1. Evaluate FROM-WHERE: apply condition C_1 on the attributes in $R_1,\ldots,R_n$
2. GROUP BY the attributes $a_1,\ldots,a_k$
3. Apply condition C_2 to each group
4. Compute aggregates in S and return the result

Agenda

~~Introduction~~

~~Basic SQL Statement~~

~~Table Variables and Set Operators~~

~~Subqueries in WHERE clauses~~

~~Subqueries in FROM and SELECT clauses~~

~~The JOIN family of operators~~

~~Aggregation~~

Null values

Data Modification statements

NULL values

Unless specified otherwise, any value in an attribute can take on the special value NULL

NULL usually means that the value is undefined or unknown or not applicable

We will see what happens when we have NULL values and we run queries over the database

NULL values for numerical operations

NULL -> NULL

If x = NULL then

4*(3-x)/7 is NULL

NULL values for Boolean operations

Conditions are evaluated using a three value logic: TRUE, FALSE, UNKNOWN

If x= NULL then x='Joe' is UNKNOWN

Considering TRUE is 1.0, UNKNOWN is 0.5 and FALSE is 0.0

C1 AND C2 = min (C1, C2)

C1 OR C2 = max (C1, C2)

NOT C1 = 1 – C1

Rule in SQL: include only tuples that yield TRUE (1.0)

A first query with NULL values

College(cName, state, enr)
Student(sID, sName, GPA, sizeHS)
Apply(sID, cName, major, decision)

```
SELECT sID, sName, GPA
FROM Student
WHERE GPA > 3.5;
```

sID	sName	GPA
123	Amy	3.9
234	Bob	3.6
456	Doris	3.9
678	Fay	3.8
987	Helen	3.7
876	Irene	3.9
654	Amy	3.9

2 additional students

Student

sID	sName	GPA	HS
123	Amy	3.9	1000
234	Bob	3.6	1500
345	Craig	3.5	500
456	Doris	3.9	1000
567	Edward	2.9	2000
678	Fay	3.8	200
789	Gary	3.4	800
987	Helen	3.7	800
876	Irene	3.9	400
765	Jay	2.9	1500
654	Amy	3.9	1000
543	Craig	3.4	2000
432	Kevin	NULL	1500
321	Lori	NULL	2500

A second query with NULL values

```
SELECT sID, sName, GPA
FROM Student
WHERE GPA <= 3.5;
```

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

sID	sName	GPA
345	Craig	3.5
567	Edward	2.9
789	Gary	3.4
765	Jay	2.9
543	Craig	3.4

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

A third query with NULL values

```
SELECT sID, sName, GPA
FROM Student
WHERE GPA > 3.5 OR GPA <= 3.5;
```

Tautology

Even with a tautology, we might not get all the data

sID	sName	GPA
123	Amy	3.9
234	Bob	3.6
345	Craig	3.5
456	Doris	3.9
567	Edward	2.9
678	Fay	3.8
789	Gary	3.4
987	Helen	3.7
876	Irene	3.9
765	Jay	2.9
654	Amy	3.9
543	Craig	3.4

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

A fourth query with NULL values

To make this query return all the students

```
SELECT sID, sName, GPA
FROM Student
WHERE GPA > 3.5 OR GPA <= 3.5 OR
GPA IS NULL;
```

Can test for NULL explicitly

- attribute IS NULL
- attribute IS NOT NULL

sID	sName	GPA
123	Amy	3.9
234	Bob	3.6
345	Craig	3.5
456	Doris	3.9
567	Edward	2.9
678	Fay	3.8
789	Gary	3.4
987	Helen	3.7
876	Irene	3.9
765	Jay	2.9
654	Amy	3.9
543	Craig	3.4
432	Kevin	
321	Lori	

A fifth query with NULL values

```
SELECT sID, sName, GPA, HS
FROM Student
WHERE GPA > 3.5 OR HS < 1600;
```

Although Kevin has a NULL GPA, he is retrieved because it has a HS<1600

sID	sName	GPA	HS
123	Amy	3.9	1000
234	Bob	3.6	1500
345	Craig	3.5	500
456	Doris	3.9	1000
678	Fay	3.8	200
789	Gary	3.4	800
987	Helen	3.7	800
876	Irene	3.9	400
765	Jay	2.9	1500
654	Amy	3.9	1000
432	Kevin		1500

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

NULL values and aggregate functions

SELECT distinct GPA

FROM Student;

GPA
NULL
2.9
3.4
3.5
3.6
3.7
3.8
3.9

8 tuples

SELECT count(distinct GPA)

FROM Student;

count(distinct GPA)
7

When counting (the distinct) values, NULLs are not included

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

NULL values

When using a database with NULL values

- Be careful when writing queries
- Understand how the NULL values are going to influence the result

Agenda

~~Introduction~~

~~Basic SQL Statement~~

~~Table Variables and Set Operators~~

~~Subqueries in WHERE clauses~~

~~Subqueries in FROM and SELECT clauses~~

~~The JOIN family of operators~~

~~Aggregation~~

~~Null values~~

Data Modification statements

Inserting new data

Insert Into Table (A_1, A_2, …, A_n)

Values (v_1, v_2, …, v_n) → If we provide values for all attributes, we may omit the list of attributes.

Insert Into Table

Select-Statement → Same schema as the table

Inserting new data

```
Insert Into College
    Values ('Carnegie Mellon', 'PA', 11500);
```

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

College

cName	state	enr
Stanford	CA	15000
Berkeley	CA	36000
MIT	MA	10000
Cornell	NY	21000
Carnegie Mellon	PA	11500

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Inserting new data

Have all students who didn't apply anywhere apply to CS at Carnegie Mellon

```
SELECT *
FROM Student
WHERE sID not in (select sID FROM Apply);
```

sID	sName	GPA	HS
456	Doris	3.9	1000
567	Edward	2.9	2000
789	Gary	3.4	800
654	Amy	3.9	1000

```
INSERT INTO Apply
SELECT sID, 'Carnegie Mellon', 'CS', NULL
FROM Student
WHERE sID not in (select sID FROM Apply);
```

Apply

sID	cName	major	dec
…	…	…	…
543	MIT	CS	N
456	Carnegie Mellon	CS	NULL
567	Carnegie Mellon	CS	NULL
789	Carnegie Mellon	CS	NULL
654	Carnegie Mellon	CS	NULL

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Inserting new data

Admit to Carnegie Mellon EE all students who were turned down in EE elsewhere

sID	sName	GPA	HS
123	Amy	3.9	1000
345	Craig	3.5	500

SELECT *

FROM Student

WHERE sID in (select sID FROM Apply WHERE major='EE' AND decision='N' AND cName<>'Carnegie Mellon');

Apply

sID	cName	major	dec
…	…	…	…
654	Carnegie Mellon	CS	NULL
123	Carnegie Mellon	EE	Y
345	Carnegie Mellon	EE	Y

INSERT INTO Apply

SELECT sID, 'Carnegie Mellon', 'EE', 'Y'

FROM Student

WHERE sID in (select sID FROM Apply WHERE major='EE' AND decision='N' AND cName<>'Carnegie Mellon');

Deleting existing data

Delete From Table

Where Condition

Condition can be complicated

- Can include subqueries and aggregation over other tables

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Deleting existing data

Delete all students who applied to more than two different majors

```
SELECT sID
FROM Apply
GROUP BY sID
HAVING count(distinct major) > 2;

DELETE FROM Student
WHERE sID in (
SELECT sID FROM Apply GROUP BY sID
HAVING count(distinct major) > 2);
```

sID
345
876

Student

sID	sName	GPA	HS
123	Amy	3.9	1000
234	Bob	3.6	1500
456	Doris	3.9	1000
567	Edward	2.9	2000
678	Fay	3.8	200
789	Gary	3.4	800
987	Helen	3.7	800
765	Jay	2.9	1500
654	Amy	3.9	1000
543	Craig	3.4	2000

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Deleting existing data

Delete those students also from Apply

```
DELETE FROM Apply
WHERE sID in (
        SELECT sID FROM Apply GROUP BY sID
        HAVING count(distinct major) > 2);
```

Not all database systems allow deletion commands where the subquery includes the same relation that you're deleting from

- In systems that don't allow, a temporary table would have to be created

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Deleting existing data

Delete colleges with no CS applicants

```
SELECT *
FROM College
WHERE cName not in
(SELECT cName from Apply where major='CS');
```

cName	state	enr
Cornell	NY	21000

```
DELETE FROM College
WHERE cName not in
(SELECT cName from Apply where major='CS');
```

cName	state	enr
Stanford	CA	15000
Berkeley	CA	36000
MIT	MA	10000
Carnegie Mellon	PA	11500

College

Updating existing data

Update Table

Set Attr = Expression

Where Condition

Update Table

Set A_1 = $Expr_1$, A_2 = $Expr_2$, …, A_n = $Expr_n$

Where Condition

Conditions and expressions can include subqueries and queries over other tables or the same table

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Updating existing data

Accept applicants to Carnegie Mellon with GPA<3.6 but turn them into economics majors

```
SELECT *
FROM Apply
WHERE cName = 'Carnegie Mellon' and sID in
(SELECT sID from Student where GPA<3.6);
```

sID	cName	major	dec
567	Carnegie Mellon	CS	
789	Carnegie Mellon	CS	

```
UPDATE Apply
SET decision = 'Y', major='economics'
WHERE cName = 'Carnegie Mellon' and sID in
(SELECT sID from Student where GPA<3.6);
```

Apply

sID	cName	major	dec
…	…	…	…
654	Carnegie Mellon	CS	NULL
123	Carnegie Mellon	EE	Y
567	Carnegie Mellon	economics	Y
789	Carnegie Mellon	economics	Y

Updating existing data

Turn the highest GPA EE applicant into a CSE applicant

```
SELECT *
FROM Apply
WHERE major='EE' AND sID in
(select sID from Student where GPA>=all
        (select GPA from Student where sID in
         (select sID from Apply where major='EE')));
```

sID	cName	major	dec
123	Stanford	EE	N
123	Cornell	EE	Y
123	Carnegie Mellon	EE	Y

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Updating existing data

Turn the highest GPA EE applicant into a CSE applicant

```
UPDATE Apply
SET major = 'CSE'
WHERE major='EE' AND sID in
(select sID from Student where GPA>=all
        (select GPA from Student where sID in
         (select sID from Apply where major='EE')));
```

sID	cName	major	dec
…	…	…	…
789	Carnegie Mellon	economics	Y
123	Stanford	CSE	N
123	Cornell	CSE	Y
123	Carnegie Mellon	CSE	Y

Apply

College(<u>cName</u>, state, enr)

Student(<u>sID</u>, sName, GPA, sizeHS)

Apply(<u>sID</u>, <u>cName</u>, <u>major</u>, decision)

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Updating existing data

Give every student the highest GPA and the smallest high school in the database

UPDATE Student

SET GPA = (select max(GPA) from Student),

sizeHS = (select min(sizeHS) from Student);

In the SET command, the right-hand side of the equals can itself be a subquery

Student

sID	sName	GPA	HS
123	Amy	3.9	200
234	Bob	3.9	200
456	Doris	3.9	200
567	Edward	3.9	200
678	Fay	3.9	200
789	Gary	3.9	200
987	Helen	3.9	200
765	Jay	3.9	200
654	Amy	3.9	200
543	Craig	3.9	200

Summary

SQL is a rich programming language that handles the way data is processed declaratively

Kahoot time!

Any doubts?

Readings

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3rd Edition

- Section 6.1 – Simple Queries in SQL
- Section 6.2 – Queries Involving More Than One Relation
- Section 6.3 - Subqueries
- Section 6.4 – Full-Relation Operations
- Section 6.5 – Database Modifications

Philip Greenspun, SQL for Web Nerds, http://philip.greenspun.com/sql/