

SQL – Data Manipulation Language

Carla Teixeira Lopes

Bases de Dados

Mestrado Integrado em Engenharia Informática e Computação, FEUP

Based on Jennifer Widom slides

Agenda

Introduction

The JOIN family of operators

Basic SQL Statement

Aggregation

Table Variables and Set Operators

Null values

Subqueries in WHERE clauses

Data Modification statements

Subqueries in FROM and SELECT clauses

SQL

Stands for **S**tructured **Q**uery **L**anguage

Pronounced “sequel”

Supported by all major commercial database systems

Standardized – many features over time

Interactive via GUI or prompt, or embedded in programs

Declarative, based on relational algebra

SQL History

1970 “A Relational Model of Data for Large Shared Data Banks” by Edgar Codd

Early **70's** SEQUEL Developed at IBM by Donald Chamberlin e Raymond Boyce

1979 First commercial version by Relational Software (now Oracle)

1986 SQL-86 and SQL-87. Ratified by ANSI and ISO

1989 SQL-891992 SQL-92. Also known as SQL2

1999 SQL:1999. Also known as SQL3. Includes regular expressions, recursive queries, triggers, non-scalar data types and some object-oriented expressions

2003 SQL:2003 XML support and auto-generated values

2006 SQL:2006 XQuery support

2008 SQL:2008

2011 SQL:2011

SQL is a ...

Data Definition Language (DDL)

- Define relational schemata

- Create/alter/delete tables and their attributes

Data Manipulation Language (DML)

- Insert/delete/modify tuples in tables

- Query one or more tables

Standard

Many standards out there

Database management systems implement something similar, but not identical to the standard for SQL

These slides will try to adhere to the standard as much as possible

Primarily the SQL2 standard and some constructs from the SQL3 standard

Sometimes we'll talk specifically about SQL as understood by SQLite

Agenda

Introduction

The JOIN family of operators

Basic SELECT Statement

Aggregation

Table Variables and Set Operators

Null values

Subqueries in WHERE clauses

Data Modification statements

Subqueries in FROM and SELECT clauses

The Basic SELECT Statement (SFW)

| | | | |
|--------|------------------------|---|-----------------------------------|
| SELECT | A_1, A_2, \dots, A_n | → | What to return |
| FROM | R_1, R_2, \dots, R_m | → | Identifies the relations to query |
| WHERE | condition | → | Combines and filters relations |

What's the equivalent in Relational Algebra?

$$\pi_{A_1, \dots, A_n} (\sigma_{condition} (R_1 \times \dots \times R_m))$$

The result is a relation with the schema A_1, A_2, \dots, A_n

SQL is compositional

Relational query languages are compositional

When a query is run over relations, the result is a relation

The schema of the obtained relation is the set of attributes that are returned

The output of one query can be used as the input to another (nesting)

This is extremely powerful

College Admission Database

Apply

| <u>sID</u> | <u>cName</u> | <u>major</u> | <u>dec</u> |
|------------|--------------|----------------|------------|
| 123 | Stanford | CS | Y |
| 123 | Stanford | EE | N |
| 123 | Berkeley | CS | Y |
| 123 | Cornell | EE | Y |
| 234 | Berkeley | biology | N |
| 345 | MIT | bioengineering | Y |
| 345 | Cornell | bioengineering | N |
| 345 | Cornell | CS | Y |
| 345 | Cornell | EE | N |
| 678 | Stanford | history | Y |
| 987 | Stanford | CS | Y |
| 987 | Berkeley | CS | Y |
| 876 | Stanford | CS | Y |
| 876 | MIT | biology | Y |
| 876 | MIT | marine biology | N |
| 765 | Stanford | history | Y |
| 765 | Cornell | history | N |
| 765 | Cornell | psychology | Y |
| 543 | MIT | CS | N |

College

| <u>cName</u> | <u>state</u> | <u>enr</u> |
|--------------|--------------|------------|
| Stanford | CA | 15000 |
| Berkeley | CA | 36000 |
| MIT | MA | 10000 |
| Cornell | NY | 21000 |

Student

| <u>sID</u> | <u>sName</u> | <u>GPA</u> | <u>HS</u> |
|------------|--------------|------------|-----------|
| 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 |

SQL query with one relation

```
SELECT sID, sName, GPA
FROM Student
WHERE GPA > 3.6;
```

| sID | sName | GPA |
|-----|-------|-----|
| 123 | Amy | 3.9 |
| 456 | Doris | 3.9 |
| 678 | Fay | 3.8 |
| 987 | Helen | 3.7 |
| 876 | Irene | 3.9 |
| 654 | Amy | 3.9 |

Not necessary to include GPA in the result even if we filter on the GPA

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

A Few Details

SQL commands are case insensitive

Same: SELECT, Select, select

Same: Student, student

Values are case sensitive

Different: 'Stanford', 'stanford'


Use single quotes for constants

'abc' - yes

"abc" - no

SQL query combining two relations

```
SELECT sName, major
FROM Student, Apply
WHERE Student.sID=Apply.sID;
```



This would happen automatically in a natural join of RA

What does it compute?

Duplicate values



| |
|---|
| College(<u>cName</u> , state, enr) |
| Student(<u>sID</u> , sName, GPA, sizeHS) |
| Apply(<u>sID</u> , <u>cName</u> , <u>major</u> , decision) |

| sName | major |
|-------|----------------|
| Amy | CS |
| Amy | EE |
| Amy | CS |
| Amy | EE |
| Bob | biology |
| Craig | bioengineering |
| Craig | bioengineering |
| Craig | CS |
| Craig | EE |
| Fay | history |
| Helen | CS |
| Helen | CS |
| Irene | CS |
| Irene | biology |
| Irene | marine biology |
| Jay | history |
| Jay | history |
| Jay | psychology |
| Craig | CS |

SQL query excluding duplicate values

```
SELECT DISTINCT sName, major
FROM Student, Apply
WHERE Student.sID=Apply.sID;
```

No duplicate values



| sName | major |
|-------|----------------|
| Amy | CS |
| Amy | EE |
| Bob | biology |
| Craig | bioengineering |
| Craig | CS |
| Craig | EE |
| Fay | history |
| Helen | CS |
| Irene | CS |
| Irene | biology |
| Irene | marine biology |
| Jay | history |
| Jay | psychology |

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Another SQL query combining two relations

SELECT sName, GPA, decision

FROM Student, Apply

WHERE Student.sID=Apply.sID AND sizeHS<1000 AND
major='CS' AND cName='Stanford';

What does it compute?

| sName | GPA | decision |
|-------|-----|----------|
| Helen | 3.7 | Y |
| Irene | 3.9 | N |

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

One more SQL query combining two relations

SELECT cName

FROM College, Apply

WHERE College.cName=Apply.cName AND
enr>20000 AND major='CS';



SQLite Error: ambiguous
column name: cName

What does it compute?

How can we correct it?

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

One more SQL query combining two relations

```
SELECT College.cName
```

```
FROM   College, Apply
```

```
WHERE  College.cName=Apply.cName AND  
       enr>20000 AND major='CS';
```

| cName |
|----------|
| Berkeley |
| Berkeley |
| Cornell |



How can we eliminate duplicates?

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Order of the results

SQL is based on an unordered model

The order of the results may change each time we run a query

We can ask for a result to be sorted, in ascending or descending order, by an attribute or set of attributes

`ORDER BY <attributes> ASC`

`ORDER BY <attributes> DESC`

Ordering is ascending, unless you specify the DESC keyword

Order of the results

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

```
SELECT Student.sID, sName, GPA,  
       Apply.cName, enr  
FROM   Student, College, Apply  
WHERE  Student.sID=Apply.sID AND  
       Apply.cName=College.cName;
```

| sID | sName | GPA | cName | enr |
|-----|-------|-----|----------|-------|
| 123 | Amy | 3.9 | Stanford | 15000 |
| 123 | Amy | 3.9 | Stanford | 15000 |
| 123 | Amy | 3.9 | Berkeley | 36000 |
| 123 | Amy | 3.9 | Cornell | 21000 |
| 234 | Bob | 3.6 | Berkeley | 36000 |
| 345 | Craig | 3.5 | MIT | 10000 |
| 345 | Craig | 3.5 | Cornell | 21000 |
| 345 | Craig | 3.5 | Cornell | 21000 |
| 345 | Craig | 3.5 | Cornell | 21000 |
| 678 | Fay | 3.8 | Stanford | 15000 |

```
SELECT Student.sID, sName, GPA,  
       Apply.cName, enr  
FROM   Student, College, Apply  
WHERE  Student.sID=Apply.sID AND  
       Apply.cName=College.cName  
ORDER BY GPA DESC;
```

| sID | sName | GPA | cName | enr |
|-----|-------|-----|----------|-------|
| 123 | Amy | 3.9 | Stanford | 15000 |
| 123 | Amy | 3.9 | Stanford | 15000 |
| 123 | Amy | 3.9 | Berkeley | 36000 |
| 123 | Amy | 3.9 | Cornell | 21000 |
| 876 | Irene | 3.9 | Stanford | 15000 |
| 876 | Irene | 3.9 | MIT | 10000 |
| 876 | Irene | 3.9 | MIT | 10000 |
| 678 | Fay | 3.8 | Stanford | 15000 |
| 987 | Helen | 3.7 | Stanford | 15000 |
| 987 | Helen | 3.7 | Berkeley | 36000 |

Order of the results


```
SELECT Student.sID, sName, GPA, Apply.cName, enr
```

```
FROM Student, College, Apply
```

```
WHERE Student.sID=Apply.sID AND Apply.cName=College.cName
```

```
ORDER BY GPA DESC, enr;
```

| sID | sName | GPA | cName | enr |
|-----|-------|-----|----------|-------|
| 876 | Irene | 3.9 | MIT | 10000 |
| 876 | Irene | 3.9 | MIT | 10000 |
| 123 | Amy | 3.9 | Stanford | 15000 |
| 123 | Amy | 3.9 | Stanford | 15000 |
| 876 | Irene | 3.9 | Stanford | 15000 |
| 123 | Amy | 3.9 | Cornell | 21000 |
| 123 | Amy | 3.9 | Berkeley | 36000 |
| 678 | Fay | 3.8 | Stanford | 15000 |
| 987 | Helen | 3.7 | Stanford | 15000 |
| 987 | Helen | 3.7 | Berkeley | 36000 |



Descending GPA as primary sort order and, within each of those, ascending enrollment

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Like operator: pattern matching on strings

Built-in operator that allows string matching on attribute values

```
SELECT sID, major  
FROM Apply  
WHERE major like '%bio%';
```



Match any major containing bio

| sID | major |
|-----|----------------|
| 234 | biology |
| 345 | bioengineering |
| 345 | bioengineering |
| 876 | biology |
| 876 | marine biology |

% = any sequence of characters

_ = any single character

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

Selecting all attributes

SELECT *

FROM Apply

WHERE major like '%bio%';

| sID | cName | major | decision |
|-----|----------|----------------|----------|
| 234 | Berkeley | biology | N |
| 345 | MIT | bioengineering | Y |
| 345 | Cornell | bioengineering | N |
| 876 | MIT | biology | Y |
| 876 | MIT | marine biology | N |

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Arithmetic within SQL clauses

```
SELECT sID, sName, GPA, HS, GPA*(HS/1000)
FROM Student;
```

| sID | sName | GPA | HS | GPA*(HS/1000) |
|-----|--------|-----|------|---------------|
| 123 | Amy | 3.9 | 1000 | 3.9 |
| 234 | Bob | 3.6 | 1500 | 5.4 |
| 345 | Craig | 3.5 | 500 | 1.75 |
| 456 | Doris | 3.9 | 1000 | 3.9 |
| 567 | Edward | 2.9 | 2000 | 5.8 |
| 678 | Fay | 3.8 | 200 | 0.76 |
| 789 | Gary | 3.4 | 800 | 2.72 |
| 987 | Helen | 3.7 | 800 | 2.96 |
| 876 | Irene | 3.9 | 400 | 1.56 |
| 765 | Jay | 2.9 | 1500 | 4.35 |
| 654 | Amy | 3.9 | 1000 | 3.9 |
| 543 | Craig | 3.4 | 2000 | 6.8 |

Boosts GPA if student is from a big high school and reduces it, if he is from a small one

→ Can we improve this result?

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

Renaming columns

```
SELECT sID, sName, GPA, HS, GPA*(HS/1000) AS scaledGPA  
FROM Student;
```

| sID | sName | GPA | HS | scaledGPA |
|-----|--------|-----|------|-----------|
| 123 | Amy | 3.9 | 1000 | 3.9 |
| 234 | Bob | 3.6 | 1500 | 5.4 |
| 345 | Craig | 3.5 | 500 | 1.75 |
| 456 | Doris | 3.9 | 1000 | 3.9 |
| 567 | Edward | 2.9 | 2000 | 5.8 |
| 678 | Fay | 3.8 | 200 | 0.76 |
| 789 | Gary | 3.4 | 800 | 2.72 |
| 987 | Helen | 3.7 | 800 | 2.96 |
| 876 | Irene | 3.9 | 400 | 1.56 |
| 765 | Jay | 2.9 | 1500 | 4.35 |
| 654 | Amy | 3.9 | 1000 | 3.9 |
| 543 | Craig | 3.4 | 2000 | 6.8 |

| |
|---|
| College(<u>cName</u> , state, enr) Student(<u>sID</u> , sName, GPA, sizeHS) Apply(<u>sID</u> , <u>cName</u> , <u>major</u> , decision) |
|---|

Agenda

Introduction

The JOIN family of operators

~~Basic SQL Statement~~

Aggregation

Table Variables and Set Operators

Null values

Subqueries in WHERE clauses

Data Modification statements

Subqueries in FROM and SELECT clauses

Table variables

Used in the FROM clause for two purposes

Make queries more readable

Rename relations when we have two instances of the same relation

```
SELECT Student.sID, sName, GPA, Apply.cName, enr
FROM   Student, College, Apply
WHERE  Student.sID=Apply.sID AND Apply.cName=College.cName;
```

```
SELECT S.sID, sName, GPA, A.cName, enr
FROM   Student S, College C, Apply A
WHERE  S.sID=A.sID AND A.cName=C.cName;
```

→ Not changing the
result, only making
the query more
readable

Table variables

How to list the pairs of students who have the same GPA?

Student(sID, sName, GPA, sizeHS)

```
SELECT S1.sID, S1.sName, S1.GPA, S2.sID, S2.sName, S2.GPA
FROM   Student S1, Student S2
WHERE  S1.GPA=S2.GPA;
```

How to list only pairs
of different students?



| sID | sName | GPA | sID | sName | GPA |
|-----|-------|-----|-----|-------|-----|
| 123 | Amy | 3.9 | 123 | Amy | 3.9 |
| 123 | Amy | 3.9 | 456 | Doris | 3.9 |
| 123 | Amy | 3.9 | 876 | Irene | 3.9 |
| 123 | Amy | 3.9 | 654 | Amy | 3.9 |
| 234 | Bob | 3.6 | 234 | Bob | 3.6 |
| 345 | Craig | 3.5 | 345 | Craig | 3.5 |
| 456 | Doris | 3.9 | 123 | Amy | 3.9 |
| 456 | Doris | 3.9 | 456 | Doris | 3.9 |
| ... | ... | ... | ... | ... | ... |

Table variables

```
SELECT S1.sID, S1.sName, S1.GPA, S2.sID, S2.sName, S2.GPA
FROM   Student S1, Student S2
WHERE  S1.GPA=S2.GPA AND S1.sID <> S2.sID;
```

How to exclude the same
pairs in different order?



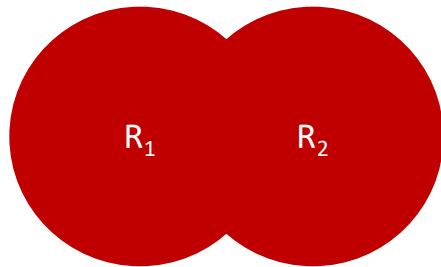
| sID | sName | GPA | sID | sName | GPA |
|-----|--------|-----|-----|-------|-----|
| 123 | Amy | 3.9 | 456 | Doris | 3.9 |
| 123 | Amy | 3.9 | 876 | Irene | 3.9 |
| 123 | Amy | 3.9 | 654 | Amy | 3.9 |
| 456 | Doris | 3.9 | 123 | Amy | 3.9 |
| 456 | Doris | 3.9 | 876 | Irene | 3.9 |
| 456 | Doris | 3.9 | 654 | Amy | 3.9 |
| 567 | Edward | 2.9 | 765 | Jay | 2.9 |
| ... | ... | ... | ... | ... | ... |

Table variables

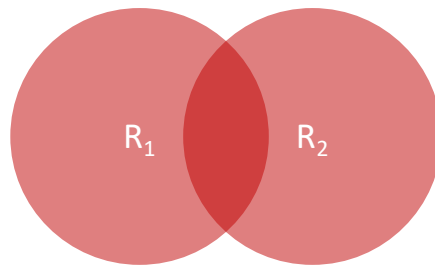
```
SELECT S1.sID, S1.sName, S1.GPA, S2.sID, S2.sName, S2.GPA
FROM   Student S1, Student S2
WHERE  S1.GPA=S2.GPA AND S1.sID < S2.sID;
```

| sID | sName | GPA | sID | sName | GPA |
|-----|--------|-----|-----|-------|-----|
| 123 | Amy | 3.9 | 456 | Doris | 3.9 |
| 123 | Amy | 3.9 | 876 | Irene | 3.9 |
| 123 | Amy | 3.9 | 654 | Amy | 3.9 |
| 456 | Doris | 3.9 | 876 | Irene | 3.9 |
| 456 | Doris | 3.9 | 654 | Amy | 3.9 |
| 567 | Edward | 2.9 | 765 | Jay | 2.9 |
| 654 | Amy | 3.9 | 876 | Irene | 3.9 |
| 543 | Craig | 3.4 | 789 | Gary | 3.4 |

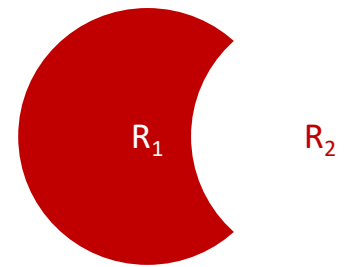
Set operators



Union



Intersect



Except

Sets, Bags and Lists

Sets

Only one occurrence of each element

Unordered elements

Bags (or multisets)

More than one occurrence of an element

Unordered elements and their occurrences

Lists

More than one occurrence of an element

Occurrences are ordered

Recall Bags

$\lambda(X)$ = “Count of tuple in X ”

Multiset X

| Tuple |
|--------|
| (1, a) |
| (1, a) |
| (1, b) |
| (2, c) |
| (2, c) |
| (2, c) |
| (1, d) |
| (1, d) |



Multiset X

| Tuple | $\lambda(X)$ |
|--------|--------------|
| (1, a) | 2 |
| (1, b) | 1 |
| (2, c) | 3 |
| (1, d) | 2 |

In a set all counts are (0,1).

Union as a bag operation

Multiset X

| Tuple | $\lambda(X)$ |
|--------|--------------|
| (1, a) | 2 |
| (1, b) | 0 |
| (2, c) | 3 |
| (1, d) | 0 |

U

Multiset Y

| Tuple | $\lambda(Y)$ |
|--------|--------------|
| (1, a) | 5 |
| (1, b) | 1 |
| (2, c) | 2 |
| (1, d) | 2 |

=

Multiset Z

| Tuple | $\lambda(Z)$ |
|--------|--------------|
| (1, a) | 7 |
| (1, b) | 1 |
| (2, c) | 5 |
| (1, d) | 2 |

$$\lambda(Z) = \lambda(X) + \lambda(Y)$$

Intersect as a bag operation

Multiset X

| Tuple | $\lambda(X)$ |
|--------|--------------|
| (1, a) | 2 |
| (1, b) | 0 |
| (2, c) | 3 |
| (1, d) | 0 |

\cap

Multiset Y

| Tuple | $\lambda(Y)$ |
|--------|--------------|
| (1, a) | 5 |
| (1, b) | 1 |
| (2, c) | 2 |
| (1, d) | 2 |

$=$

Multiset Z

| Tuple | $\lambda(Z)$ |
|--------|--------------|
| (1, a) | 2 |
| (1, b) | 0 |
| (2, c) | 2 |
| (1, d) | 0 |

$$\lambda(Z) = \min(\lambda(X), \lambda(Y))$$

Except as a bag operation

Multiset X

| Tuple | $\lambda(X)$ |
|--------|--------------|
| (1, a) | 2 |
| (1, b) | 0 |
| (2, c) | 3 |
| (1, d) | 0 |

—

Multiset Y

| Tuple | $\lambda(Y)$ |
|--------|--------------|
| (1, a) | 5 |
| (1, b) | 1 |
| (2, c) | 2 |
| (1, d) | 2 |

=

Multiset Z

| Tuple | $\lambda(Z)$ |
|--------|--------------|
| (1, a) | 0 |
| (1, b) | 0 |
| (2, c) | 1 |
| (1, d) | 0 |

If $\lambda(X) > \lambda(Y)$

$$\lambda(Z) = \lambda(X) - \lambda(Y)$$

Else

0

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Union operator

```
SELECT cName FROM College
```

```
UNION
```

```
SELECT sName FROM Student;
```

In SQL, the two sides of the union don't have to be the same

How to unify the two schemas?

```
SELECT cName AS name FROM College
```

```
UNION
```

```
SELECT sName AS name FROM Student;
```

| cName |
|----------|
| Amy |
| Berkeley |
| Bob |
| Cornell |
| Craig |
| Doris |
| Edward |
| Fay |
| Gary |
| Helen |
| Irene |
| Jay |
| MIT |
| Stanford |

Union operator

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

By default, in SQL, the union operator eliminates duplicates

If we want to have duplicates in our result

```
SELECT cName AS name FROM College
```

UNION ALL

```
SELECT sName AS name FROM Student;
```

Result is not sorted anymore

Why?

How can we sort the result?

```
SELECT cName AS name FROM College
```

```
UNION ALL
```

```
SELECT sName AS name FROM Student
```

```
ORDER BY name;
```

| name |
|----------|
| Stanford |
| Berkeley |
| MIT |
| Cornell |
| Amy |
| Bob |
| Craig |
| Doris |
| Edward |
| Fay |
| Gary |
| Helen |
| Irene |
| Jay |
| Amy |
| Craig |

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/>