

# SQL – Data Manipulation Language

---

Carla Teixeira Lopes

Bases de Dados

Mestrado Integrado em Engenharia Informática e Computação, FEUP

Based on Jennifer Widom slides

# Operators summary

---

attribute [**NOT**] **IN** Relation

[**NOT**] attribute **IN** Relation

[**NOT**] **EXISTS** Relation

attribute <comparison> **ALL** Relation

attribute <comparison> **ANY** Relation

Comparison may be: <; >; <=; >=; =; <>

ALL and ANY not  
supported by SQLite

# Existential quantifier

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

An existential quantifier ( $\exists$ ) is a quantifier of the form “there exists”

Find colleges with some applications of students with a GPA higher than 3.8

```
SELECT DISTINCT cName
FROM Apply, Student
WHERE Apply.sID=Student.sID AND GPA>3.8;
```

Existential is easy!

cName
Stanford
Berkeley
MIT
Cornell

# Universal quantifier

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

A universal quantifier ( $\forall$ ) is a quantifier of the form “for all”

Find colleges with applications of students all having a GPA higher than 3.8



Equivalent

Find colleges that only have applications of students with a GPA higher than 3.8

**cName**

SELECT DISTINCT cName

FROM Apply

WHERE College.cName NOT IN (

SELECT Apply.cName FROM Apply, Student

WHERE Apply.sID=Student.sID AND GPA<=3.8);

# 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

# Subqueries in FROM and SELECT

---

SELECT  $A_1, A_2, \dots, A_n$   Expressions involving  
FROM  $R_1, R_2, \dots, R_m$   subqueries  
WHERE condition

Subqueries are nested SELECT statements

Subqueries in FROM generate a table to be used in the query

Subqueries in SELECT produce a value that comes out of the query

# Subqueries in the FROM clause

---

```
SELECT  sID, sName, GPA, GPA*(HS/1000) as scaledGPA
FROM    Student
WHERE   GPA*(HS/1000)-GPA>1.0 OR GPA - GPA*(HS/1000) > 1.0;
```

Return all students whose scaledGPA changes GPA by more than 1

Can we simplify this query?

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

sID	sName	GPA	scaledGPA
234	Bob	3.6	5.4
345	Craig	3.5	1.75
567	Edward	2.9	5.8
678	Fay	3.8	0.76
876	Irene	3.9	1.56
765	Jay	2.9	4.35
543	Craig	3.4	6.8

# Subqueries in the FROM clause

---

```
SELECT sID, sName, GPA, GPA*(HS/1000) as scaledGPA
FROM Student
WHERE GPA*(HS/1000)-GPA>1.0 OR GPA – GPA*(HS/1000) > 1.0;
```



```
SELECT sID, sName, GPA, GPA*(HS/1000) as scaledGPA
FROM Student
WHERE abs(GPA*(HS/1000)-GPA)>1.0;
```

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

<u>sID</u>	sName	GPA	scaledGPA
234	Bob	3.6	5.4
345	Craig	3.5	1.75
567	Edward	2.9	5.8
678	Fay	3.8	0.76
876	Irene	3.9	1.56
765	Jay	2.9	4.35
543	Craig	3.4	6.8



# Subqueries in the FROM clause

---

```
SELECT sID, sName, GPA, GPA*(HS/1000) as scaledGPA
FROM Student
WHERE abs(GPA*(HS/1000)-GPA) > 1.0;
```



```
SELECT *
FROM (select sID, sName, GPA, GPA*(HS/1000) as scaledGPA
      from Student) G
WHERE abs(scaledGPA - GPA) > 1.0;
```

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

sID	sName	GPA	scaledGPA
234	Bob	3.6	5.4
345	Craig	3.5	1.75
567	Edward	2.9	5.8
678	Fay	3.8	0.76
876	Irene	3.9	1.56
765	Jay	2.9	4.35
543	Craig	3.4	6.8

# Subqueries in the SELECT clause

---

```
SELECT DISTINCT College.cName, state, GPA
FROM   College, Apply, Student
WHERE  College.cName = Apply.cName
      AND Apply.sID = Student.sID
      AND GPA >= all
      (select GPA from Student, Apply
       where Student.sID = Apply.sID
       and Apply.cName = College.cName);
```

<u>cName</u>	state	GPA
Stanford	CA	3.9
Berkeley	CA	3.9
Cornell	NY	3.9
MIT	MA	3.9

How to rewrite it  
using a subquery  
in SELECT?

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

Return colleges paired with the highest GPA of their applicants

# Subqueries in the SELECT clause

---

```
SELECT cName, state,  
(SELECT DISTINCT GPA  
FROM   Apply, Student  
WHERE  College.cName = Apply.cName  
       AND Apply.sID = Student.sID  
       AND GPA >= all  
       (select GPA from Student, Apply  
        where Student.sID = Apply.sID  
        and Apply.cName = College.cName)) AS GPA
```

Computes the  
highest GPA for  
the college

```
FROM College;
```

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

# Subqueries in the SELECT clause

---

Query that returns colleges paired with the name of the applicants

```
SELECT cName, state,  
(SELECT DISTINCT sName  
FROM   Apply, Student  
WHERE  College.cName = Apply.cName  
       AND Apply.sID = Student.sID) AS sName  
FROM College;
```



Error. Why?



subquery returns  
more than 1 row



subquery in SELECT can only  
return 1 column of 1 tuple

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~~

# The JOIN family of operators

---

SELECT  $A_1, A_2, \dots, A_n$

FROM  $R_1, R_2, \dots, R_m$

WHERE condition



Implicit join

## Explicit Joins

Inner join on *condition*

$\bowtie_{\theta}$

Natural join

$\bowtie$

Inner join using (*attrs*)

$\bowtie$  explicitly listing the attributes to be equated

Left | Right | Full Outer Join

Combines tuples as in  $\bowtie_{\theta}$  but when they don't match they are added to the result with NULL values

None of these operators adds expressive power to SQL

# Inner join on *condition*

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

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



Equivalent to:

```
SELECT DISTINCT sName, major
FROM Student INNER JOIN Apply
ON Student.sID = Apply.sID;
```



INNER JOIN is the DEFAULT JOIN operator in SQL

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

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

# Inner join on *condition*: example 2

---

```
SELECT sName, GPA
FROM   Student, Apply
WHERE  Student.sID=Apply.sID AND HS<1000 AND major='CS' AND cName='Stanford';
```

↓  
Equivalent to:

```
SELECT sName, GPA
FROM   Student JOIN Apply
ON     Student.sID=Apply.sID
WHERE  HS<1000 AND major='CS' AND cName='Stanford';
```

sName	GPA
Helen	3.7
Irene	3.9

↓  
Equivalent to:

```
SELECT sName, GPA
FROM   Student JOIN Apply
ON     Student.sID=Apply.sID AND HS<1000 AND major='CS' AND cName='Stanford';
```

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



# Inner join on *condition* with 3 relations

---

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

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

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

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

```
FROM   Apply JOIN Student JOIN College
```

```
ON      Apply.sID = Student.sID AND Apply.cName = College.cName;
```

Runs on SQLite but  
not in every system

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

```
FROM   (Apply JOIN Student ON Apply.sID = Student.sID) JOIN College
```

```
ON      Apply.cName = College.cName;
```

Some systems  
(e.g.: Postgres)  
requires the join  
operators to be  
binary

# Interaction between query and processor

---

SQL systems tend to follow the structure that is provided by the JOIN operators and parenthesis

```
SELECT Apply.sID, sName, GPA, Apply.cName, enr  
FROM   (Apply JOIN Student ON Apply.sID = Student.sID) JOIN College  
ON Apply.cName = College.cName;
```

Typically, Apply will be  
joined with Student  
first

The order by things are done affects query performance

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

# Natural join

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

It joins tables based on the attributes with the same name and keeps only one of those attributes

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



Equivalent to:

```
SELECT DISTINCT sName, major
FROM Student NATURAL JOIN Apply;
```

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

# Natural join with additional conditions

---

```
SELECT sName, GPA
FROM   Student JOIN Apply
ON Student.sID=Apply.sID
WHERE HS<1000 AND major='CS' AND cName='Stanford';
```



Equivalent to:

```
SELECT sName, GPA
FROM   Student NATURAL JOIN Apply
WHERE HS<1000 AND major='CS' AND cName='Stanford';
```

sName	GPA
Helen	3.7
Irene	3.9

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

## Inner join using (*attrs*)

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Using lists the attributes that should be acquainted across the 2 relations

Only attributes that appear in both relations

```
SELECT sName, GPA
FROM Student NATURAL JOIN Apply
WHERE HS<1000 AND major='CS' AND cName='Stanford';
```

sName	GPA
Helen	3.7
Irene	3.9



Equivalent to:

```
SELECT sName, GPA
FROM Student JOIN Apply using(sID)
WHERE HS<1000 AND major='CS' AND cName='Stanford';
```

Better practice than using the natural join

# Join with more than 1 instance of a relation

---

```
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;
```

Finds pairs of students with the same GPA

```
SELECT S1.sID, S1.sName, S1.GPA, S2.sID, S2.sName, S2.GPA
FROM   Student S1 join Student S2 using(GPA)
ON      S1.sID < S2.sID;
```

→ Error



Using and ON  
cannot be used  
in combination

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

# Join with more than 1 instance of a relation

---

```
SELECT S1.sID, S1.sName, S1.GPA, S2.sID, S2.sName, S2.GPA
FROM   Student S1 join Student S2 using(GPA)
WHERE  S1.sID < S2.sID;
```

sID	sName	GPA	sID1	sName1	GPA1
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

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

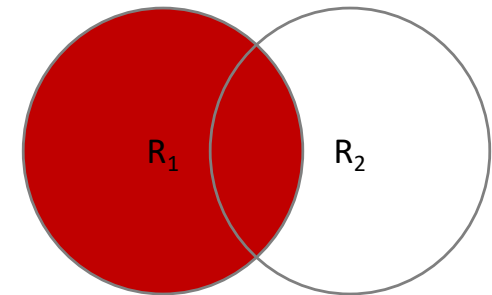
Apply(sID, cName, major, decision)

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

## Left outer join

Takes any tuples on the left side and if they don't have a match on a tuple from the right, it is still added to the result and padded with NULL values

Tuples with no matches are *dangling tuples*



```
SELECT sName, sID, cName, major  
FROM Student LEFT OUTER JOIN Apply using(sID);
```

LEFT JOIN



abbreviation



# Left outer join

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

SELECT sName, sID, cName, major  
FROM Student INNER JOIN Apply  
using(sID);

sName	sID	cName	major
Amy	123	Cornell	EE
Amy	123	Berkeley	CS
Amy	123	Stanford	EE
Amy	123	Stanford	CS
Bob	234	Berkeley	biology
Craig	345	Cornell	EE
Craig	345	Cornell	CS
Craig	345	Cornell	bioengineering
Craig	345	MIT	bioengineering
Fay	678	Stanford	history
...	...	...	...

Students who have applied somewhere

SELECT sName, sID, cName, major  
FROM Student LEFT JOIN Apply  
using(sID);

sName	sID	cName	major
Amy	123	Cornell	EE
Amy	123	Berkeley	CS
Amy	123	Stanford	EE
Amy	123	Stanford	CS
Bob	234	Berkeley	biology
Craig	345	Cornell	EE
Craig	345	Cornell	CS
Craig	345	Cornell	bioengineering
Craig	345	MIT	bioengineering
Doris	456	NULL	NULL
...	...	...	...

Students who have applied somewhere plus  
students who haven't yet applied anywhere

# Natural left outer join

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

```
SELECT sName, sID, cName, major
FROM Student NATURAL LEFT JOIN Apply;
```

sName	sID	cName	major
Amy	123	Cornell	EE
Amy	123	Berkeley	CS
Amy	123	Stanford	EE
Amy	123	Stanford	CS
Bob	234	Berkeley	biology
Craig	345	Cornell	EE
Craig	345	Cornell	CS
Craig	345	Cornell	bioengineering
Craig	345	MIT	bioengineering
Doris	456	NULL	NULL
...	...	...	...

## Left outer join

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

How to rewrite the left outer join without using it?

```
SELECT sName, Student.sID, cName, major
FROM Student, Apply
Where Student.sID=Apply.sID
UNION
SELECT sName, sID, NULL, NULL
FROM Student
WHERE sID NOT IN (select sID from Apply)
```

sName	sID	cName	major
Amy	123	Cornell	EE
Amy	123	Berkeley	CS
Amy	123	Stanford	EE
Amy	123	Stanford	CS
Bob	234	Berkeley	biology
Craig	345	Cornell	EE
Craig	345	Cornell	CS
Craig	345	Cornell	bioengineering
Craig	345	MIT	bioengineering
Doris	456	NULL	NULL
...	...	...	...

## Right outer join

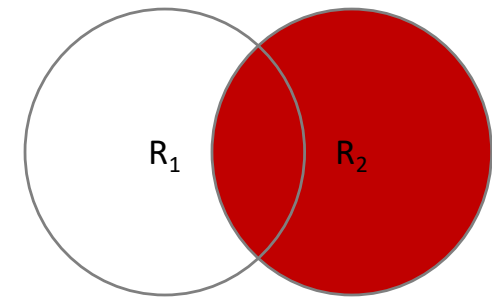
College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

Takes any tuples on the right side and if they don't have a match on a tuple from the left, it is still added to the result and padded with NULL values

We can also use the left outer join for the same effect swapping the order of the relations



```
SELECT sName, sID, cName, major  
FROM Student RIGHT OUTER JOIN Apply using(sID);
```

## Full outer join

---

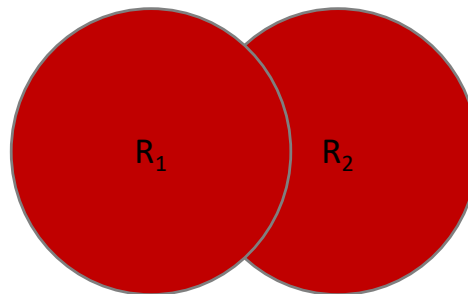
College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

To include unmatched tuples from both sides of a join

```
SELECT sName, sID, cName, major  
FROM Student FULL OUTER JOIN Apply using(sID);
```



## Full outer join

---

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

How can we express a full outer join without using it?

```
SELECT sName, Student.sID, cName, major  
FROM Student LEFT JOIN Apply using(sID)
```

```
UNION
```

```
SELECT sName, Student.sID, cName, major  
FROM Student RIGHT JOIN Apply using(sID);
```

Automatically  
eliminates duplicates

College(cName, state, enr)

Student(sID, sName, GPA, sizeHS)

Apply(sID, cName, major, decision)

## Full outer join

---

How can we express a full outer join without using joins?

```
SELECT sName, Student.sID, cName, major
```

```
FROM Student, Apply
```

```
Where Student.sID=Apply.sID
```

```
UNION
```

```
SELECT sName, sID, NULL, NULL
```

```
FROM Student
```

```
WHERE sID NOT IN (select sID from Apply)
```

```
UNION
```

```
SELECT NULL, sID, cName, major
```

```
FROM Apply
```

```
WHERE sID NOT IN (select sID from Student)
```

# Outer joins and commutativity

---

Commutativity  $(A \text{ op } B) = (B \text{ op } A)$

Left and Right Outer Joins are not commutative

Full Outer Join is commutative



# Outer joins and associativity

Associativity  $(A \text{ op } B) \text{ op } C = A \text{ op } (B \text{ op } C)$

T1		T2	
A	B	B	C
1	2	2	3

T3	
A	C
4	5

SELECT A, B, C


FROM (T1 natural full outer join T2) natural full outer join T3;



A	B	C
1	2	3
4	NULL	5

SELECT A, B, C

FROM T1 natural full outer join (T2 natural full outer join T3);



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

Left and right outer joins are not associative either

# Outer joins summary

---

## Left outer join

Include the left tuple even if there's no match

## Right outer join

Include the right tuple even if there's no match

## Full outer join

Include the both left and right tuples even if there's no match

# Kahoot time!

---

Any doubts?

# Readings

---

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3<sup>rd</sup> 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/>