CSC375 / BIF375
Database Management Systems

Chapter 6
The Database Language SQL
Part 3

Correlated queries, Aggregations, Database modifications



Part 3.1 Correlated queries

EXISTS

NOT EXISTS

DISTINCT

EXISTS / NOT EXISTS

- These keywords can be used with a query that returns a table on its *right*.
- It **doesn't** have a *left* side.
- Returns true or false.

EXISTS

- **Syntax**

```
SELECT x from m
   WHERE EXISTS (SELECT a FROM t WHERE c)
```

- Returns **TRUE** if the table returned by the subquery on the right side has at least 1 row

- Returns **FALSE** if the subquery returns zero rows (empty)

`NOT EXISTS` works in the opposite way. Returns `TRUE` if table is empty, `FALSE` if not

Example

Students

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	18	134
milad	ghantous	21	1345
tarek	dib	18	12345
Karim	Amer	18	21231

TASK

Get all students who are not unique in their age. (other students have the same age as them)

Approach

- Loop over the students in **students** table
- For each one, check if there exists another student with same age but different ID.
- If yes, return it. if not, discard it.

Example

Students

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	18	134
milad	ghantous	21	1345
tarek	dib	18	12345
Karim	Amer	18	21231

Solution

```
SELECT *
FROM students
WHERE EXISTS
```

Replace with:

- a student having same age but different ID than the one we are currently ranging over
- **PROBLEM**: how to say `ID!=ID` or `age=age`?

Example

Students

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	18	134
milad	ghantous	21	1345
tarek	dib	18	12345
Karim	Amer	18	21231

Solution

```
SELECT *
FROM students S1
WHERE EXISTS
 (SELECT ID FROM students
   WHERE ID != S1.ID AND age = S1.age)
```

Result

name	lastname	age	ID
sara	kamel	18	134
tarek	dib	18	12345
Karim	Amer	18	21231

Explanation

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	18	134
milad	ghantous	21	1345
tarek	dib	18	12345
Karim	Amer	18	21231



```
SELECT *
FROM students S1
WHERE EXISTS
 (SELECT ID FROM students
   WHERE ID != S1.ID AND age = S1.age)
```

We renamed our table in the main query S1

We call these correlated queries

Iteration #	values for S1	subquery becomes	subquery result	EXISTS returns

1	S1.ID = 123
	S1.age=19
		`SELECT ID`
		`FROM students`
		`WHERE ID != 123 AND`
		`age = 19`
			empty set	FALSE

2	S1.ID = 134
	S1.age=18
		`SELECT ID`
		`FROM students`
		`WHERE ID != 134 AND`
		`age = 18`
			ID
			12345
			21231
				TRUE

and so on…

DISTINCT Keyword

- It’s used to eliminate duplicates

Students

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	18	134
milad	ghantous	21	1345
tarek	dib	18	12345
Karim	Amer	18	21231

```
SELECT DISTINCT age
FROM students
```

Result

age
19
18
21

DISTINCT Keyword

- When you specify a column that has NULL values in the DISTINCT clause, the DISTINCT clause will keep only one NULL value because it considers all NULL values are the same.

- Multiple columns in the DISTINCT clause:

Rosters

ID	major	absence
123	EE	3
134	EE	3
1345	EE	4
12345	ME	3

```
SELECT DISTNCT major, absence
FROM rosters
```

major	absence
EE	3
EE	4
ME	3


QUIZ
TIME!

Pop quiz

Roster

ID	major	absence
123	EE	4
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4

- Get all student IDs whose number of absences is not unique using EXISTS

```
SELECT ID
FROM roster R1
WHERE EXISTS
(_____________________________________________)
```

- Get all student IDs who have unique number of absences (no one else has the same number)

```
SELECT ID
FROM roster R1
WHERE NOT EXISTS
(_____________________________________________)
```

Part 3.2

Aggregations

Aggregations

- `SUM`, `AVG`, `COUNT`, `MIN`, and `MAX` can be applied to a column in a SELECT clause to produce that aggregation on the column.

- Also, `COUNT(*)` counts the number of tuples.

Examples

Students

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	18	134
milad	ghantous	21	1345
tarek	dib	18	12345
Karim	Amer	18	21231

```
SELECT MAX(ID)
FROM students
```

MAX(ID)
21231

```
SELECT SUM(age)
FROM students
WHERE age>18
```

SUM(age)
40

```
SELECT AVG(age) as av_age
FROM students
```

av_age
18.8000

```
SELECT count(*)
FROM students
```

count(*)
5

```
SELECT count(DISTINCT age)
FROM students
```

count(DISTINCT age)
3

NULL’s Ignored in Aggregation

- NULL never **contributes** to a sum, average, or count, and can never be the minimum or maximum of a column, so it’s ignored.

- But **if all the values in some column are NULLs**, then the result of the aggregation is NULL.

  - Exception: COUNT of the above case (ALL NULL) is equal to 0.

Examples

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	*NULL*	134
milad	ghantous	*NULL*	1345
tarek	dib	*NULL*	12345
Karim	Amer	18	21231

```
select SUM(age)
from students
```

sum(age)
37

```
select COUNT(age)
from students
```

count(age)
2

```
select AVG(age)
from students
```

average(age)
18.5

---

name	lastname	age	ID
rim	ismail	*NULL*	123
sara	kamel	*NULL*	134
milad	ghantous	*NULL*	1345
tarek	dib	*NULL*	12345
Karim	Amer	*NULL*	21231

```
select SUM(age)
from students
```

SUM(age)
NULL

```
select COUNT(age)
from students
```

COUNT(age)
0

Examples

Rosters

ID	major	absence
123	EE	1
134	EE	2
1345	EE	3
12345	ME	10

```
select AVG(absence)
from rosters
Where major = ‘EE”
```

AVG(absence)
2

```
select *
from rosters
Where absence
  >(select avg(absence) from rosters)
```

ID	major	absence
12345	ME	10


QUIZ
TIME!

Pop quiz

- What's the output?

```
SELECT AVG(absence)
FROM roster
```

Roster

ID	major	absence
123	EE	4
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4

- Write a query to calculate the number of majors we have? (hint: use count)

Part 3.3

GROUP BY / HAVING

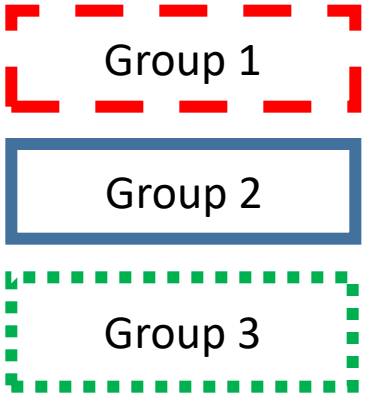
GROUP BY keyword

- This is used in the query to divide the tuples into groups

Roster

name	major	absence
john	CCE	2
khaled	EE	1
milad	CCE	2
rim	EE	4
sara	EE	7
tarek	ME	4

```
select *
from roster
group by major
```



- This is what we get, a little different than we expected.
- Each group is represented by **1 tuple not all.**

name	major	absence
john	CCE	2
khaled	EE	1
tarek	ME	4

Why did this happen?

- This is because GROUP BY should be **always** used with Aggregations and not alone!

- Aggregations like sum, max, min will be applied on a group by group basis.

Examples

name	major	absence
john	CCE	2
khaled	EE	1
milad	CCE	2
rim	EE	4
sara	EE	7
tarek	ME	4

```
SELECT major, MAX(absence)
FROM roster
GROUP BY major
```

Step 1: Grouping

Step 2: for each group, apply aggregation

major	max(absence)
CCE	2
EE	7
ME	4

Let's try this

name	lastname	age	ID
rim	ismail	19	123
sara	kamel	18	134
milad	ghantous	21	1345
tarek	dib	18	12345
Karim	Amer	19	21231

Query

```
SELECT name, MAX(ID)
FROM students
GROUP BY age
```

Result

name	MAX(ID)
sara	12345
rim	21231
milad	1345

- This is such a weird result and doesn't make sense.
- We are grouping by age, but projecting the "name" for each group

Why did this happen?

- If any aggregation is used, then each element of the SELECT list must be either:
  1. Aggregated (sum, min, max…), or
  2. An attribute on the GROUP BY list.

  Otherwise, it **should not** be in the SELECT list.

Examples

Do the queries below give the same result?

```
SELECT major                    ?       SELECT distinct major
FROM roster                     =       FROM roster
GROUP BY major
```

name	major	absence
john	CCE	2
khaled	EE	1
milad	CCE	2
rim	EE	4
sara	EE	7
tarek	ME	4

Answer: yes

GROUP BY and NULL values

- If the **grouping** column contains one or more **null values**, all **null values** form a single **group**.

Employees

ID	name	position	department
1	Jad	Technician	Maintenance
2	Samer	Accountant	Accounting
3	Sara	*NULL*	*NULL*
4	Lea	*NULL*	IT
5	Tarek	*NULL*	*NULL*

```
SELECT department,
count(*) from
employees GROUP BY
department;
```

department	count(*)
NULL	2
Accounting	1
IT	1
Maintenance	1

GROUP BY and NULL values

Employees

ID	name	position	department
1	Jad	Technician	Maintenance
2	Samer	Accountant	Accounting
3	Sara	*NULL*	*NULL*
4	Lea	*NULL*	IT
5	Tarek	*NULL*	*NULL*

```
SELECT department, count(position) from employees GROUP BY department;
```

department	count(position)
NULL	0
Accounting	1
IT	0
Maintenance	1

Examples

name	major	absence
john	CCE	2
khaled	EE	1
milad	CCE	2
rim	EE	4
sara	EE	7
tarek	ME	4

What it the output of the following query?

```
SELECT major, absence, count(*)
FROM roster
GROUP BY major, absence
```

Major	Absence	Count(*)
CCE	2	2
EE	1	1
EE	4	1
EE	7	1
ME	4	1

HAVING keyword

- `HAVING` <*condition*> follows a **`GROUP BY`** clause.
- The *condition* is checked for **each** group
  - Groups **not satisfying** the *condition* are **eliminated**.
  - Groups satisfying the *condition* are kept.
- The HAVING *condition* is **always** aggregated.

Example

ID	major	absence
123	EE	4
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4

```
SELECT major
FROM roster
GROUP BY major
HAVING max(absence)<6
```

- Here we are looking for groups based on majors, if a group has a max absence **less than** 6.
- **We have 3 groups**:
  - EE with max = 5
  - ME with max = 6
  - CCE with max = 7

Result

major
EE

- Only EE group satisfies the condition


QUIZ
TIME!

Pop quiz

- Write a query to output all the majors that have the **total number** of absences above 10.

Roster

ID	major	absence
123	EE	4
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4

Order of execution

```
SELECT column_1, column_2, aggregation(column_3)
FROM table_1 NATURAL JOIN table_2
WHERE <….>
GROUP BY <….>
HAVING <….>
ORDER BY <….>
```

1. **FROM** is executed first to select the table(s) we are working with
2. **WHERE** is executed second to return only the rows satisfying some conditions
3. **GROUP BY** is then executed to group the filtered data from the previous step by a specific column or multiple columns
4. **HAVING** is then executed to filter the groups
5. **SELECT** is executed to select a subset of columns from the filtered grouped result
6. **ORDER BY** finally orders the result

Part 3.4

Database modifications

Insertion

Syntax

```
INSERT INTO T(A1,A2,…AN) VALUES (’V1’, ‘V2’, … ‘VN’)
```

Example

```
INSERT INTO roster(ID, major, absence)
VALUES (999, ‘EE’, 5)
```

Insertion

Create table roster(
ID integer primary key,
major char(20),
absence integer default 0);

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query.

Example

```
INSERT INTO roster
VALUES (999, 'EE', 5)
```

Insertion

It is possible to only insert data in specific columns. The column names must be specified.

Example

```
INSERT INTO roster(ID, major)
VALUES (999, 'EE')
-- the value of absence will be 0 (default value)
```

Deletion

ID	major	absence
123	EE	4
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4

Syntax

```
DELETE FROM T [WHERE C]
```

Example

```
DELETE FROM roster
    WHERE ABSENCE <=5
```

All rows are deleted if no condition is provided

Result

ID	major	absence
125	ME	6
126	CCE	7

Update

ID	major	absence
123	EE	4
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4

Syntax

```
UPDATE T SET ATT=V
[WHERE C]
```

Example 1

```
UPDATE ROSTER SET ABSENCE = 5
```

Result

ID	major	absence
123	EE	5
124	EE	5
125	ME	5
126	CCE	5
127	CCE	5
128	ME	5

Update

ID	major	absence
123	EE	4
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4

Example 2

```
UPDATE ROSTER SET ABSENCE = 5
WHERE MAJOR='EE'
```

Result

ID	major	absence
123	EE	5
124	EE	5
125	ME	6
126	CCE	7
127	CCE	5
128	ME	4