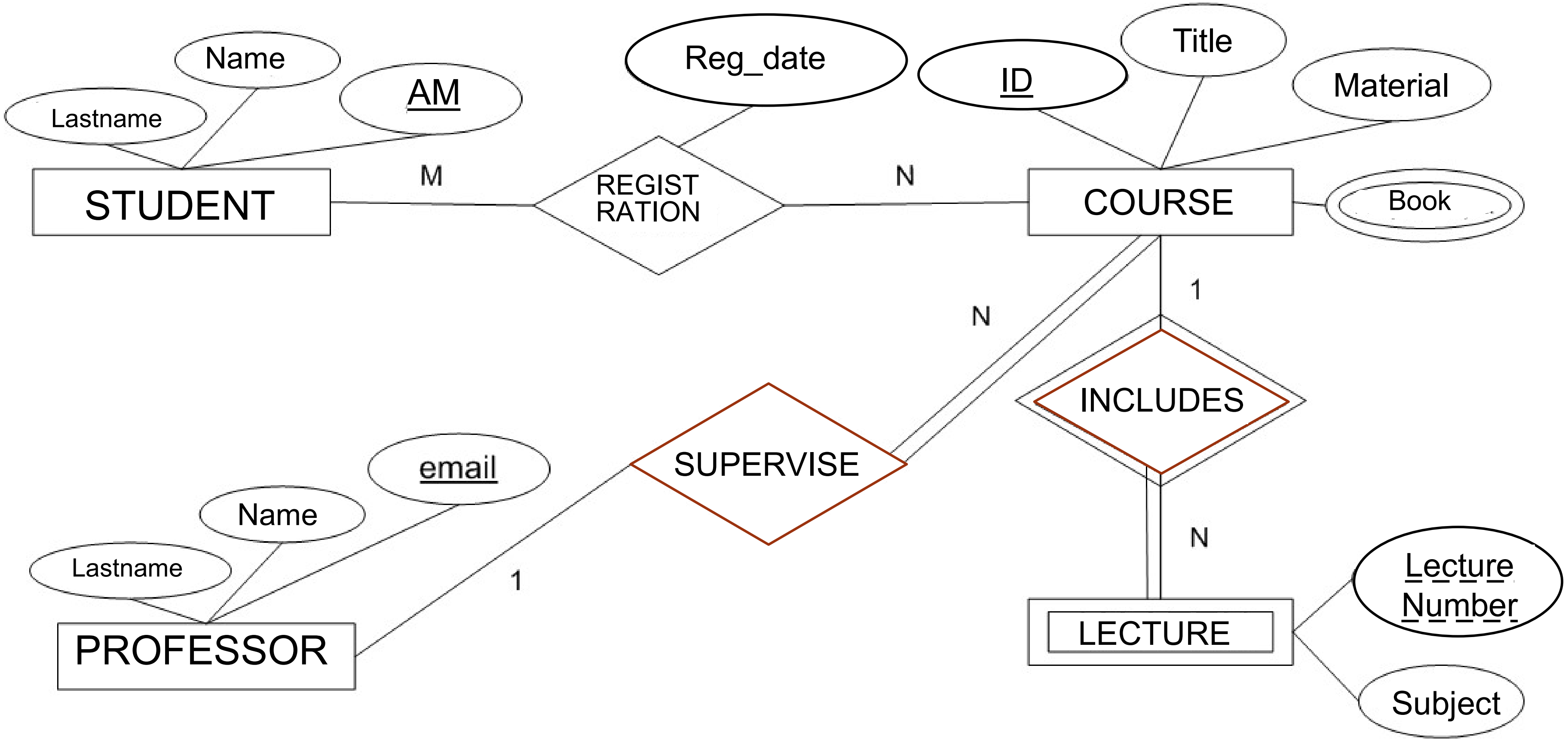
Database Systems Lab

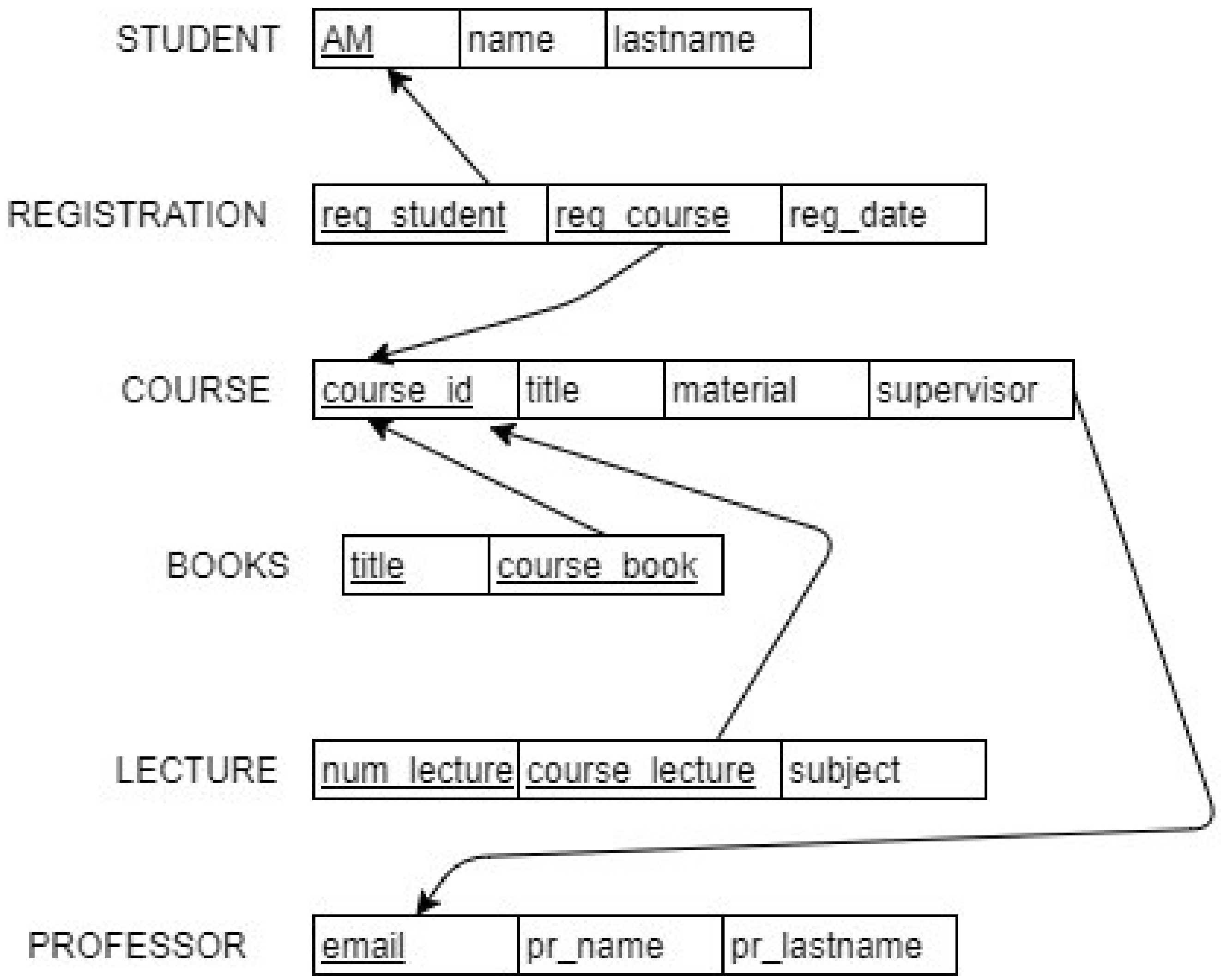
MySQL queries

ER-diagram example



AM is student's registration number

Relational-model example



STUDENT: AM | name | lastname

REGISTRATION: reg_student | reg_course | reg_date

COURSE: course_id | title | material | supervisor

BOOKS: title | course_book

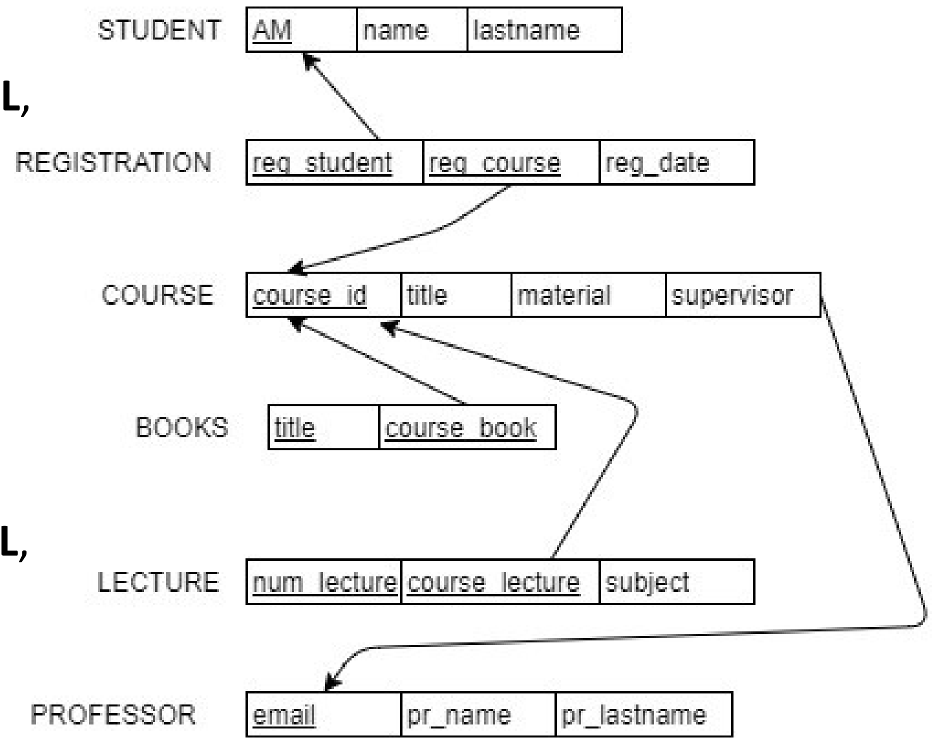
LECTURE: num_lecture | course_lecture | subject

PROFESSOR: email | pr_name | pr_lastname

Table-creation statements

```
CREATE  TABLE student(
    name VARCHAR(25) DEFAULT 'unknown' NOT NULL,
    lastname VARCHAR(25) DEFAULT 'unknown' NOT NULL,
    AM INT(5) NOT NULL AUTO_INCREMENT,
    PRIMARY KEY(AM)
    );

CREATE  TABLE professor(
    pr_name VARCHAR(25) DEFAULT 'unknown' NOT NULL,
    pr_lastname VARCHAR(25) DEFAULT 'unknown' NOT NULL,
    email VARCHAR(255) NOT NULL,
    PRIMARY KEY(email)
    );
```



STUDENT AM name lastname

REGISTRATION reg_student reg_course reg_date

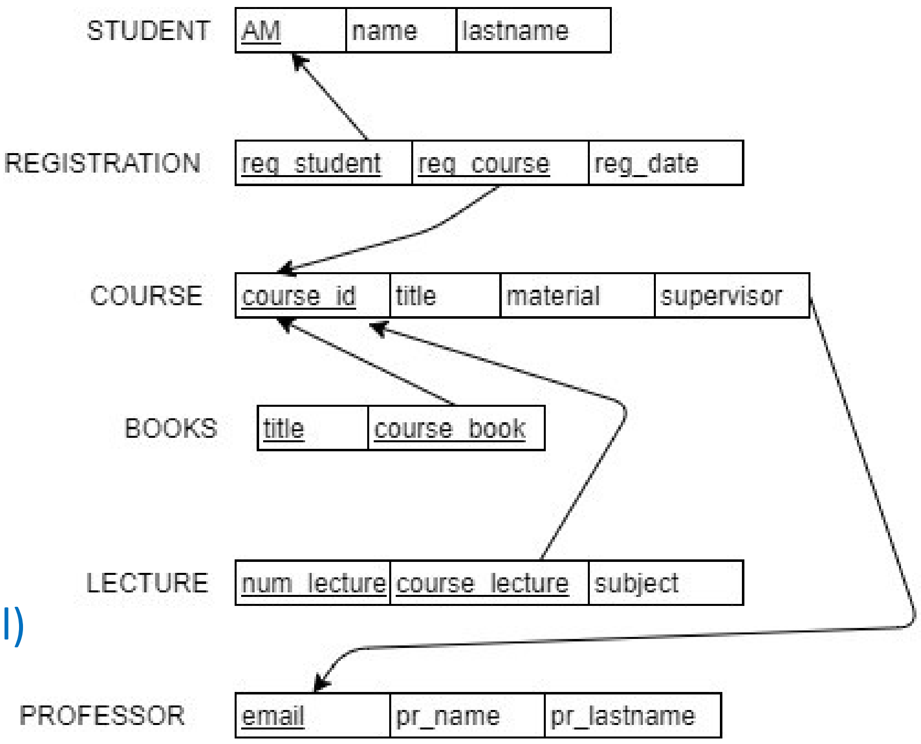
COURSE course_id title material supervisor

BOOKS title course_book

LECTURE num_lecture course_lecture subject

PROFESSOR email pr_name pr_lastname

```
CREATE TABLE course (
    title VARCHAR(255) DEFAULT 'unknown' NOT NULL,
    material  TEXT,
    course_id  INT(4) NOT NULL  AUTO_INCREMENT,
    supervisor VARCHAR(255) NOT  NULL,
    PRIMARY KEY(course_id),
    UNIQUE(title),
    CONSTRAINT SUPERVISED
    FOREIGN KEY (supervisor) REFERENCES professor(email)
    ON DELETE CASCADE ON UPDATE CASCADE);
```



STUDENT: AM | name | lastname

REGISTRATION: reg_student | reg_course | reg_date

COURSE: course_id | title | material | supervisor

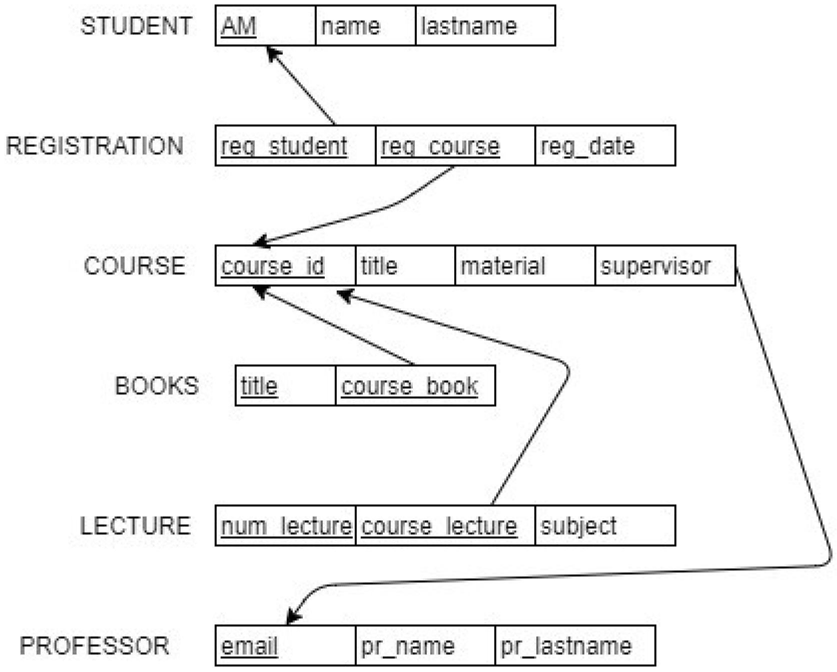
BOOKS: title | course_book

LECTURE: num_lecture | course_lecture | subject

PROFESSOR: email | pr_name | pr_lastname

```
CREATE TABLE books (
    title VARCHAR(128) DEFAULT 'Title' NOT NULL,
    course_book INT(4) NOT NULL,
    PRIMARY KEY(title,course_book),
    CONSTRAINT CRSBOOK
    FOREIGN KEY (course_book) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

```
CREATE TABLE lecture (
    subject VARCHAR(128),
    num_lecture INT(2) NOT NULL,
    course_lecture INT(4) NOT NULL,
    PRIMARY KEY(num_lecture,course_lecture),
    CONSTRAINT CRSLECTURE
    FOREIGN KEY (course_lecture) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE);
```



STUDENT: AM | name | lastname

REGISTRATION: reg_student | reg_course | reg_date

COURSE: course_id | title | material | supervisor

BOOKS: title | course_book

LECTURE: num_lecture | course_lecture | subject

PROFESSOR: email | pr_name | pr_lastname

```
CREATE TABLE registration (
    reg_date DATE NOT NULL,
    reg_student INT(5) NOT NULL,
    reg_course INT(4) NOT NULL,
    PRIMARY KEY(reg_student,reg_course),
    CONSTRAINT CRSREGISTRATION
    FOREIGN KEY (reg_course) REFERENCES course(course_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT STDNTREGISTRATION
    FOREIGN KEY (reg_student) REFERENCES student(AM)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

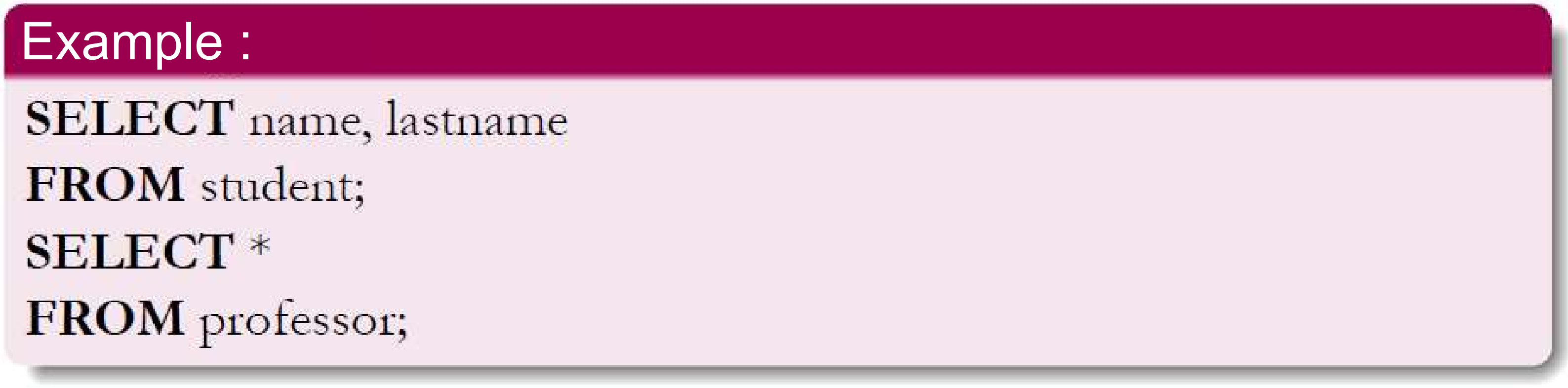
Data retrieval from a database

- The SELECT statement is used to retrieve data from one or more database tables.
- The SELECT statement returns this data in the form of a record (row) set that matches specific criteria (temporary table).
- The SELECT statement determines:
  - Which columns are included in the returned set
  - Which rows are included in the returned set (rows that fulfil certain conditions)
- The basic syntax of the SELECT statement is:

  SELECT <column_name(s)>
  FROM <table_name(s)>
  WHERE <condition>

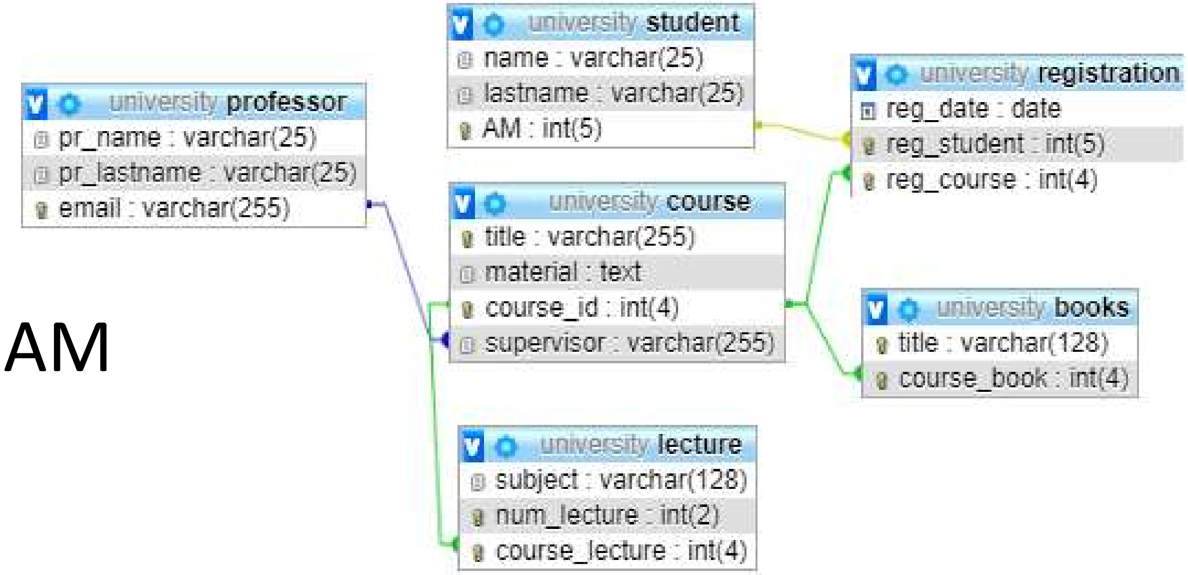
Select: Syntax

- **SELECT** <column_name(s)>:
  - The name of columns (fields) you want to retrieve data from
  - The column names are separated by comma
  - Only these columns will be included in the returned set.
  - The * asterisk sign can be used to include all the columns of a table
- **FROM** <table_name(s)>:
  - The name of tables you want to retrieve data from
  - If more than one tables are required, these can be combined in different ways (join)



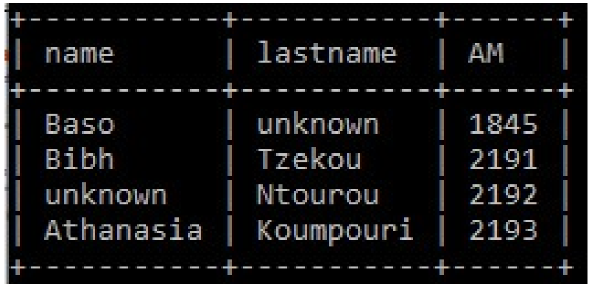
Example :

```
SELECT name, lastname
FROM student;
SELECT *
FROM professor;
```



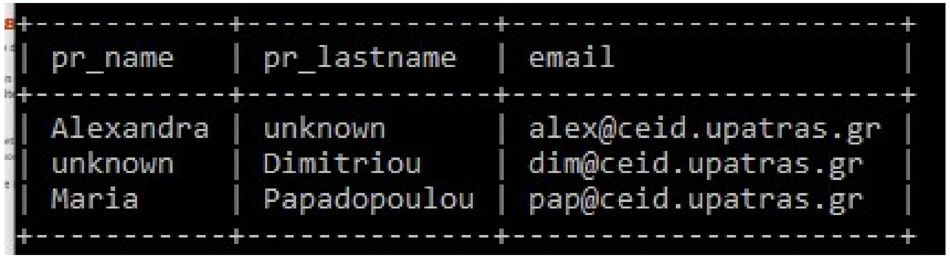
Select name, lastname, AM

From student;



```
+-----------+-----------+------+
| name      | lastname  | AM   |
+-----------+-----------+------+
| Baso      | unknown   | 1845 |
| Bibh      | Tzekou    | 2191 |
| unknown   | Ntourou   | 2192 |
| Athanasia | Koumpouri | 2193 |
+-----------+-----------+------+
```

Select * from professor;



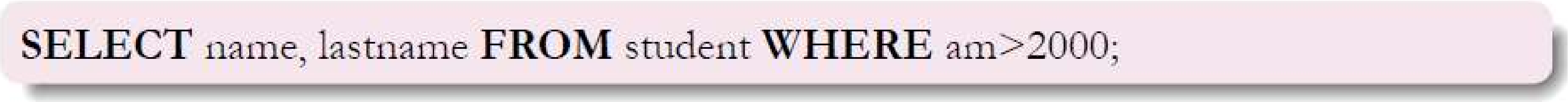
```
+-----------+-------------+----------------------+
| pr_name   | pr_lastname | email                |
+-----------+-------------+----------------------+
| Alexandra | unknown     | alex@ceid.upatras.gr |
| unknown   | Dimitriou   | dim@ceid.upatras.gr  |
| Maria     | Papadopoulou | pap@ceid.upatras.gr  |
+-----------+-------------+----------------------+
```

Select: WHERE

- In the WHERE clause a condition is specified that includes column names and SQL logical or comparison operators.
- Only the records (rows) that meet this condition are returned.
- The following operators can be used:
  - Logical-Boolean: AND, OR, NOT
  - Comparison: =, <>, >, <, >=, <=
  - Comparison with the NULL value: IS NULL, IS NOT NULL
  - Alphanumeric comparision: LIKE (using wildcards)
    - % zero or more characters
    - _ exactly one character

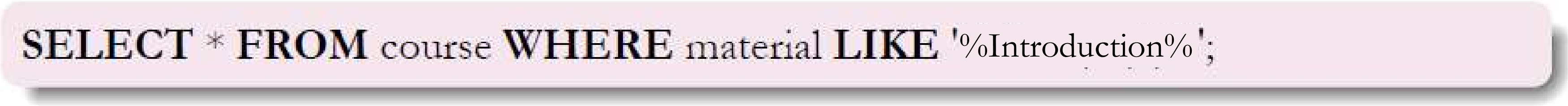
Select - examples

- Select all the names of student that have AM larger than 2000



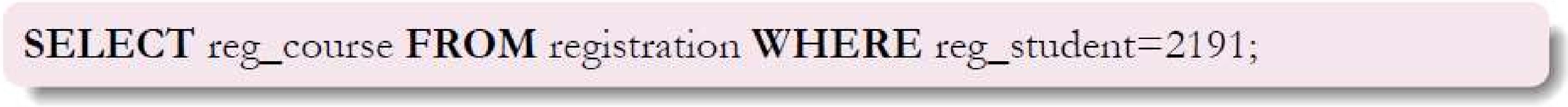
```
SELECT name, lastname FROM student WHERE am>2000;
```

- Select all the courses that include the word ‘Introduction’ in column *Material*



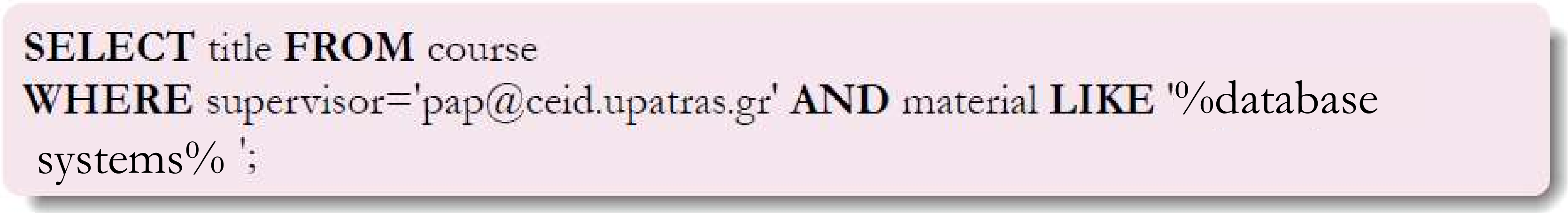
```
SELECT * FROM course WHERE material LIKE '%Introduction%';
```

- Select all the course ids that the student with AM equals to 2191 has registered to



```
SELECT reg_course FROM registration WHERE reg_student=2191;
```

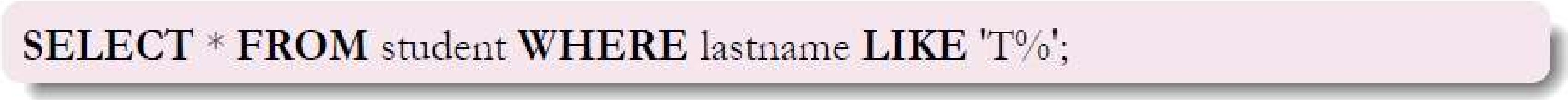
- Select all the course titles that are supervised by Papadopoulou (pap@ceid.upatras.gr) with material relevant to database systems



```
SELECT title FROM course
WHERE supervisor='pap@ceid.upatras.gr' AND material LIKE '%database
 systems% ';
```

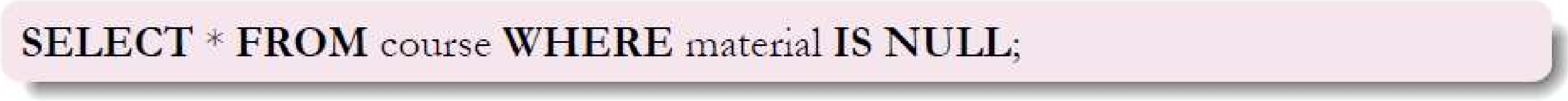
Select - examples

- Select all students whose lastname begins with a ‘T’



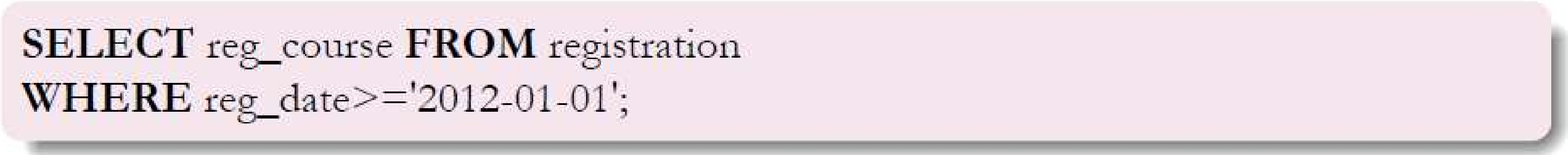
```
SELECT * FROM student WHERE lastname LIKE 'T%';
```

- Select all courses with no description (material)



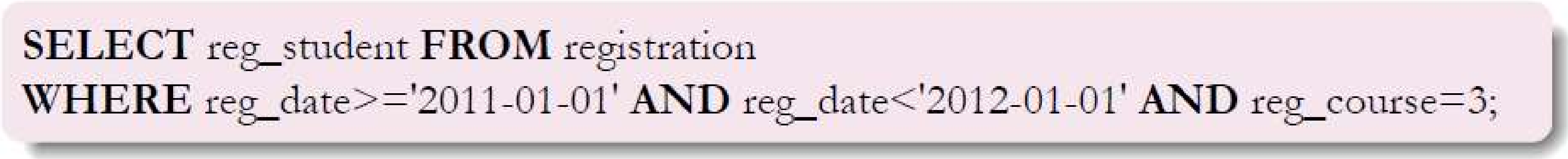
```
SELECT * FROM course WHERE material IS NULL;
```

- Select all the course id that students have registered to later than 2012



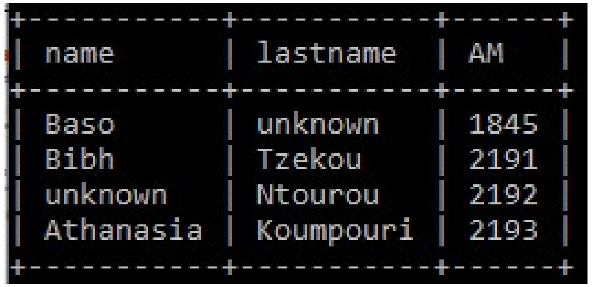
```
SELECT reg_course FROM registration
WHERE reg_date>='2012-01-01';
```

- Select all the AM of students that registered within 2011 to the course with id equals to 3



```
SELECT reg_student FROM registration
WHERE reg_date>='2011-01-01' AND reg_date<'2012-01-01' AND reg_course=3;
```

Select * from Student;

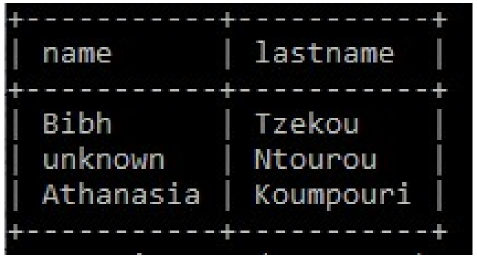


```
+-----------+-----------+------+
| name      | lastname  | AM   |
+-----------+-----------+------+
| Baso      | unknown   | 1845 |
| Bibh      | Tzekou    | 2191 |
| unknown   | Ntourou   | 2192 |
| Athanasia | Koumpouri | 2193 |
+-----------+-----------+------+
```

Select name, lastname

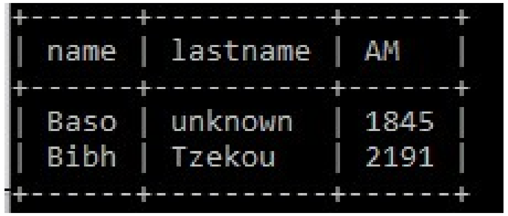
From student

Where AM>2000;



```
+-----------+-----------+
| name      | lastname  |
+-----------+-----------+
| Bibh      | Tzekou    |
| unknown   | Ntourou   |
| Athanasia | Koumpouri |
+-----------+-----------+
```

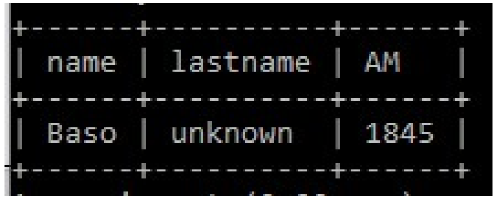
SELECT * FROM `student` WHERE name like "%b%";



```
+------+----------+------+
| name | lastname | AM   |
+------+----------+------+
| Baso | unknown  | 1845 |
| Bibh | Tzekou   | 2191 |
+------+----------+------+
```

SELECT * FROM `student` WHERE name like "_as%“;

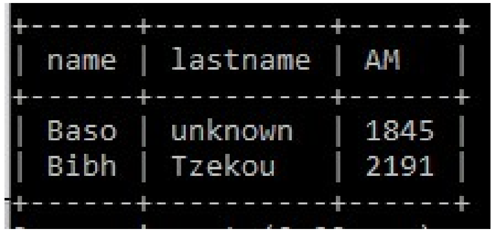
Only one character preceding as%



```
+------+----------+------+
| name | lastname | AM   |
+------+----------+------+
| Baso | unknown  | 1845 |
+------+----------+------+
```

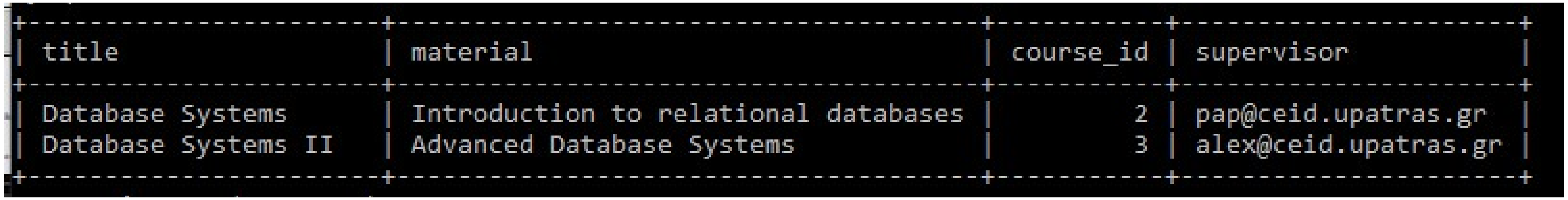
SELECT * FROM student WHERE name like “B___“;

Exactly 3 characters following B



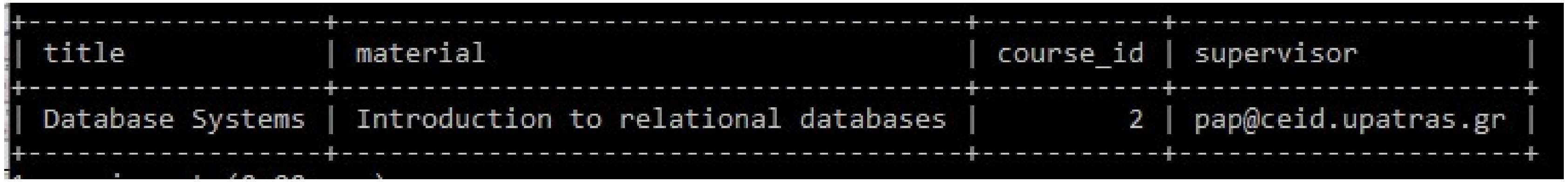
```
+------+----------+------+
| name | lastname | AM   |
+------+----------+------+
| Baso | unknown  | 1845 |
| Bibh | Tzekou   | 2191 |
+------+----------+------+
```

SELECT * FROM course;



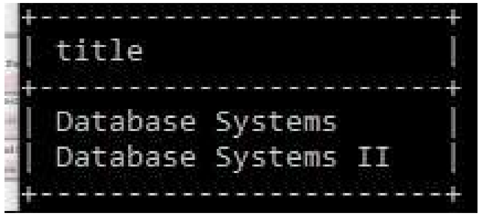
```
+------------------------+--------------------------------------+-----------+----------------------+
| title                  | material                             | course_id | supervisor           |
+------------------------+--------------------------------------+-----------+----------------------+
| Database Systems       | Introduction to relational databases |         2 | pap@ceid.upatras.gr  |
| Database Systems II    | Advanced Database Systems            |         3 | alex@ceid.upatras.gr |
+------------------------+--------------------------------------+-----------+----------------------+
```

SELECT * FROM course WHERE material like '%intro%';



```
+------------------+--------------------------------------+-----------+---------------------+
| title            | material                             | course_id | supervisor          |
+------------------+--------------------------------------+-----------+---------------------+
| Database Systems | Introduction to relational databases |         2 | pap@ceid.upatras.gr |
+------------------+--------------------------------------+-----------+---------------------+
```

SELECT title FROM course
WHERE supervisor like '%ceid%' and title like '%ase%';



```
+----------------------+
| title                |
+----------------------+
| Database Systems     |
| Database Systems II  |
+----------------------+
```

Full Syntax of SELECT Statement

Select a1, a2, .., an

From r1, r2, .., rm

Where P

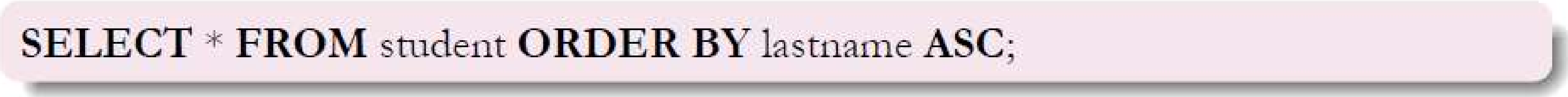
[**order by** .....]

[**group by** ....]

[**having**...]

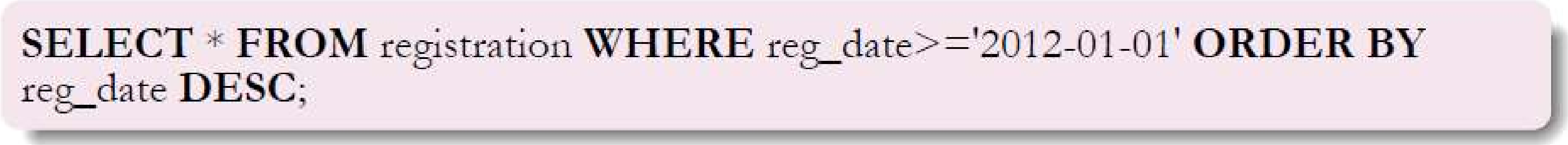
Select – Order By

- The ORDER BY clause is used to sort the returned set of records based on one or more fields.
- Sorting precedence is from left to right.
- ASC and DESC keywords define ascending or descending order, respectively.
- Select all students in ascending order based on their lastname



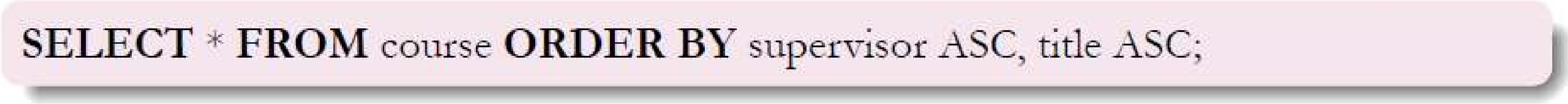
```
SELECT * FROM student ORDER BY lastname ASC;
```

- Select all registrations of 2012 sorted from newest to oldest



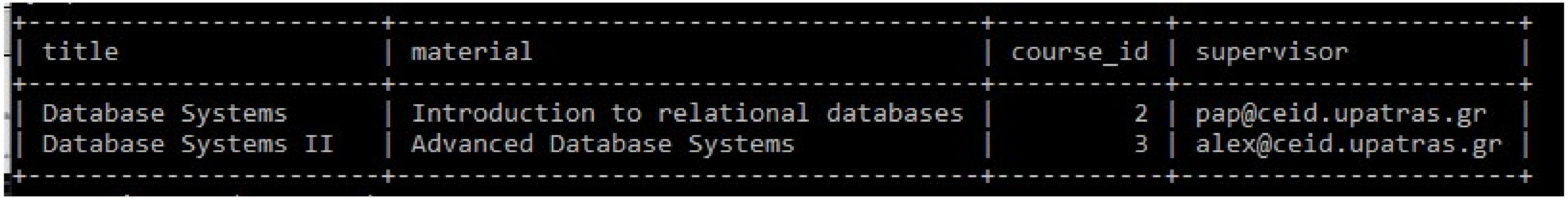
```
SELECT * FROM registration WHERE reg_date>='2012-01-01' ORDER BY
reg_date DESC;
```

- Select all courses sorted first by professors’ email then by title



```
SELECT * FROM course ORDER BY supervisor ASC, title ASC;
```

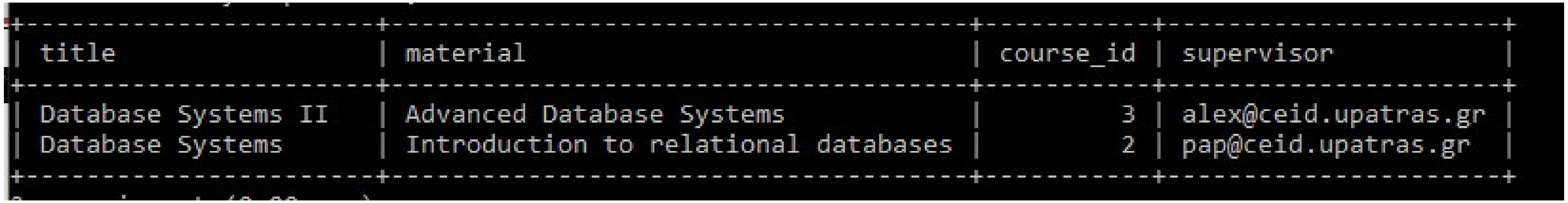
Select * from course



```
+-----------------------+---------------------------------------+------------+-----------------------+
| title                 | material                              | course_id  | supervisor            |
+-----------------------+---------------------------------------+------------+-----------------------+
| Database Systems      | Introduction to relational databases |          2 | pap@ceid.upatras.gr   |
| Database Systems II   | Advanced Database Systems             |          3 | alex@ceid.upatras.gr  |
+-----------------------+---------------------------------------+------------+-----------------------+
```

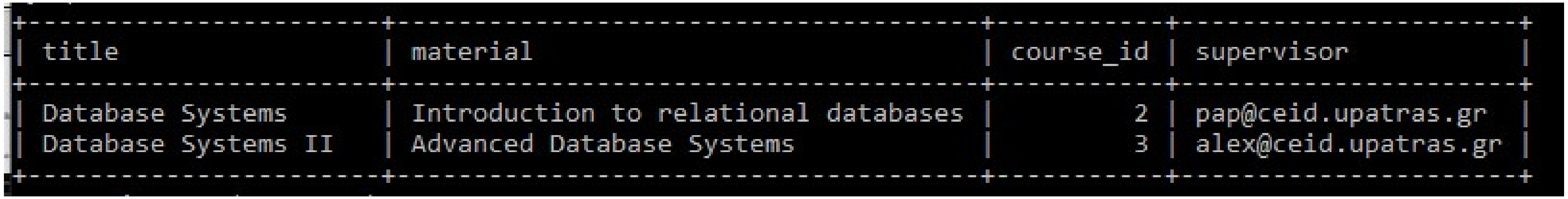
Sort based on supervisor

select * from course
order by supervisor;   **By default ASC**



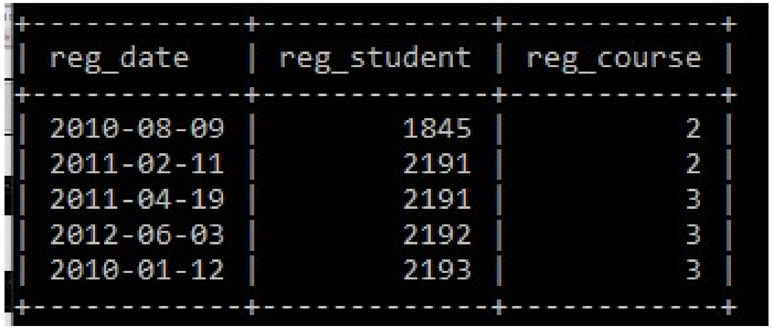
```
+-----------------------+---------------------------------------+------------+-----------------------+
| title                 | material                              | course_id  | supervisor            |
+-----------------------+---------------------------------------+------------+-----------------------+
| Database Systems II   | Advanced Database Systems             |          3 | alex@ceid.upatras.gr  |
| Database Systems      | Introduction to relational databases |          2 | pap@ceid.upatras.gr   |
+-----------------------+---------------------------------------+------------+-----------------------+
```

select * from course
order by supervisor
DESC;



```
+-----------------------+---------------------------------------+------------+-----------------------+
| title                 | material                              | course_id  | supervisor            |
+-----------------------+---------------------------------------+------------+-----------------------+
| Database Systems      | Introduction to relational databases |          2 | pap@ceid.upatras.gr   |
| Database Systems II   | Advanced Database Systems             |          3 | alex@ceid.upatras.gr  |
+-----------------------+---------------------------------------+------------+-----------------------+
```

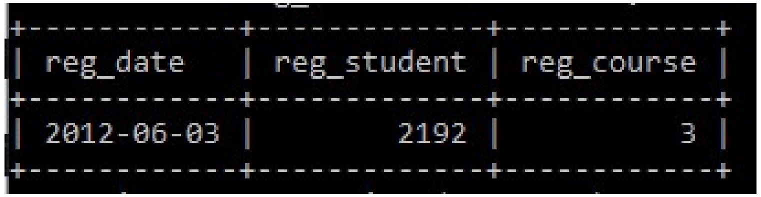
Select * from registration;



```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

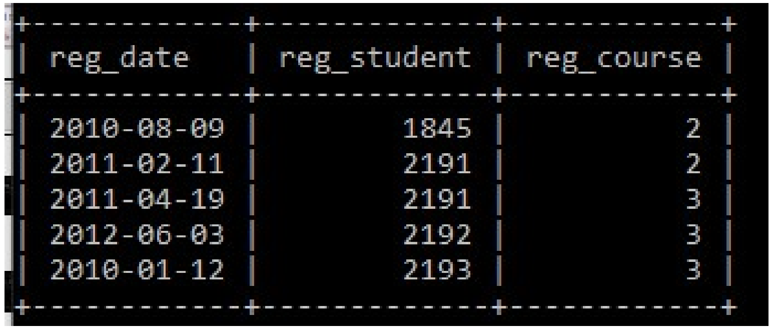
SELECT * FROM `registration`

WHERE reg_date>' 2011-10-01';



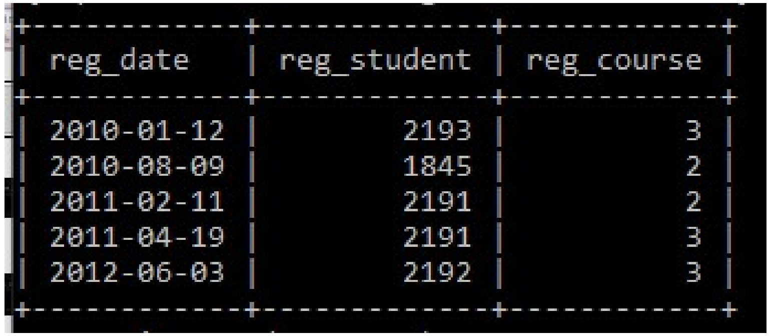
```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2012-06-03 |        2192 |          3 |
+------------+-------------+------------+
```

Select * from registration;



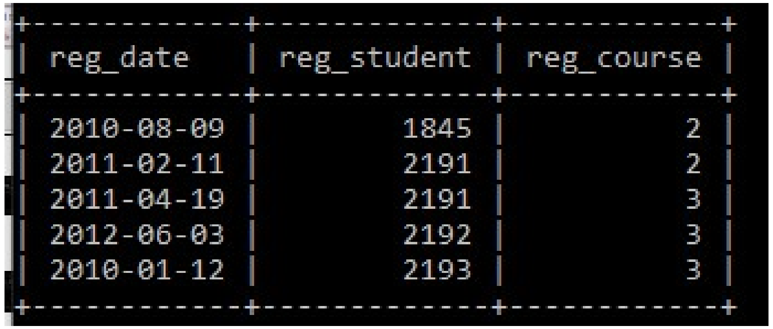
```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

SELECT * FROM registration order by reg_date;



```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-01-12 |        2193 |          3 |
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
+------------+-------------+------------+
```

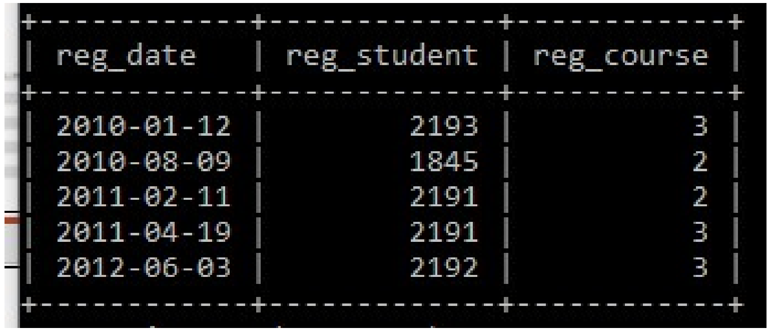
Select * from registration;



```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

SELECT * FROM registration

order by reg_date, reg_student;

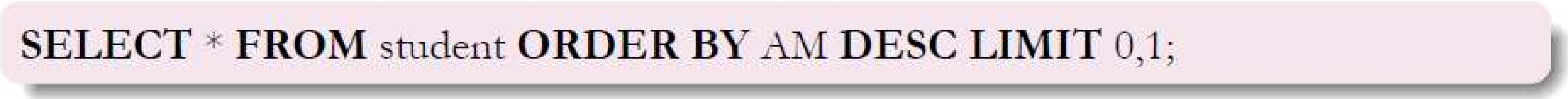


```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-01-12 |        2193 |          3 |
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
+------------+-------------+------------+
```

Select - Limit

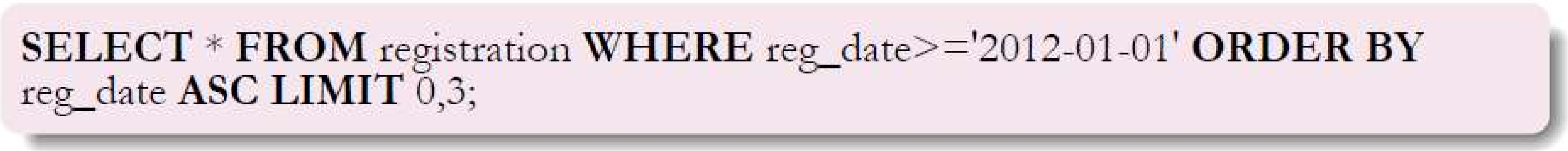
- Limits the number of records returned by a SELECT statement
- Syntax: **LIMIT** m,n
  - **m** is the offset, namely the number of records (rows) that are skipped before beginning to return rows (starting from 0).
  - **n** determines the number of records (rows) returned by the query.
  - The offset is optional. If omitted, the query will return n rows from the first row returned by the SELECT statement.
  - LIMIT is often used with ORDER BY for ensuring the order of the result.

- Select the student that has the largest AM



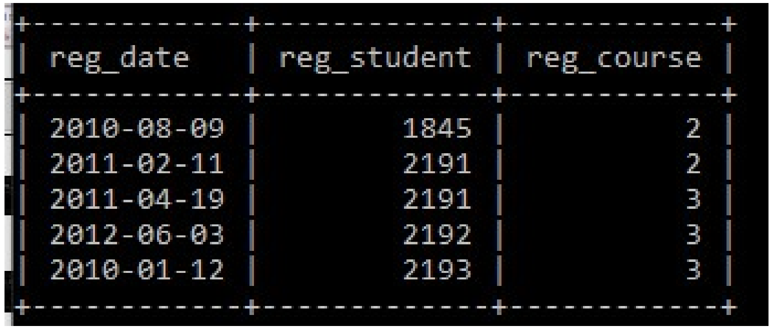
```
SELECT * FROM student ORDER BY AM DESC LIMIT 0,1;
```

- Select the first three registrations since 2012



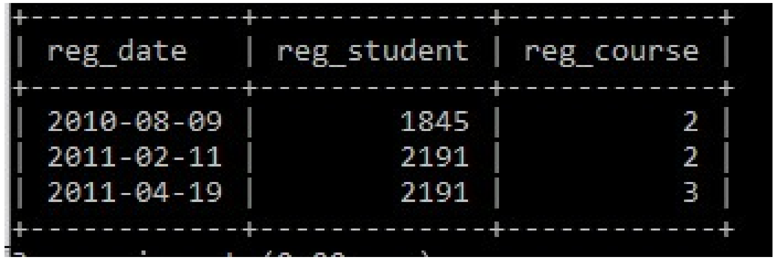
```
SELECT * FROM registration WHERE reg_date>='2012-01-01' ORDER BY
reg_date ASC LIMIT 0,3;
```

Select * from registration;



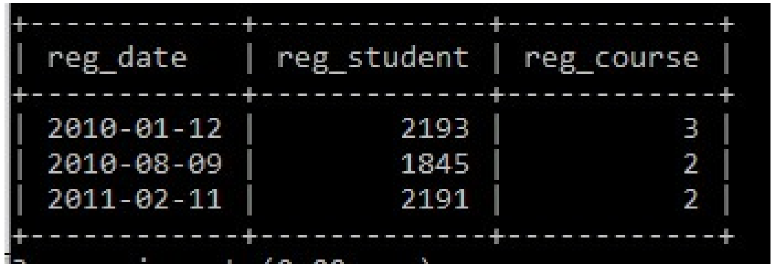
```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

SELECT * FROM registration LIMIT 3;



```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
+------------+-------------+------------+
```

SELECT * FROM `registration` order by reg_date, reg_student Limit 3;

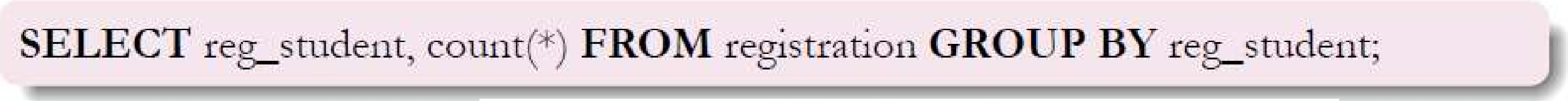


```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-01-12 |        2193 |          3 |
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
+------------+-------------+------------+
```

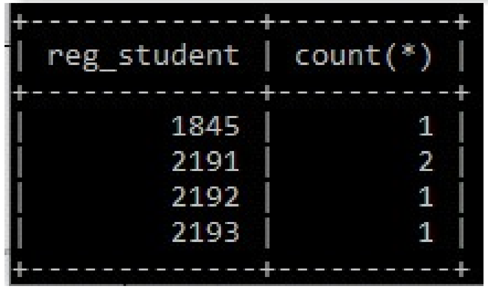
Select - Group By

- The GROUP BY clause groups records(rows) based on one or more columns.
- Different rows will be arranged in a group, if these rows have the same value in the column(s) included in the GROUP BY clause
- If more than one column is included in the GROPU BY, the groups are arranged for the rows with the similar values on all columns.

- Select the AM and the number of registrations of all students.

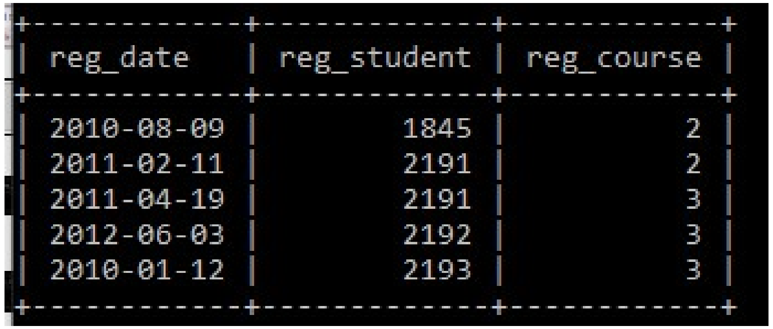


```
SELECT reg_student, count(*) FROM registration GROUP BY reg_student;
```



```
+-------------+----------+
| reg_student | count(*) |
+-------------+----------+
|        1845 |        1 |
|        2191 |        2 |
|        2192 |        1 |
|        2193 |        1 |
+-------------+----------+
```

Select * from registration;

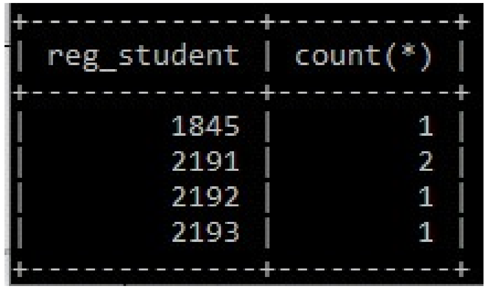


```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

SELECT reg_student, count(*)

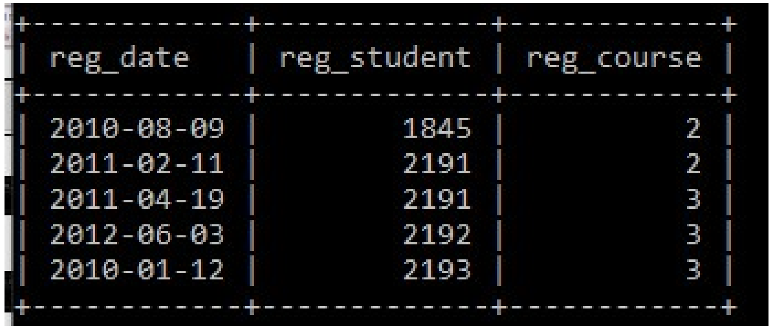
FROM `registration`

group by reg_student;



```
+-------------+----------+
| reg_student | count(*) |
+-------------+----------+
|        1845 |        1 |
|        2191 |        2 |
|        2192 |        1 |
|        2193 |        1 |
+-------------+----------+
```

Select * from registration;

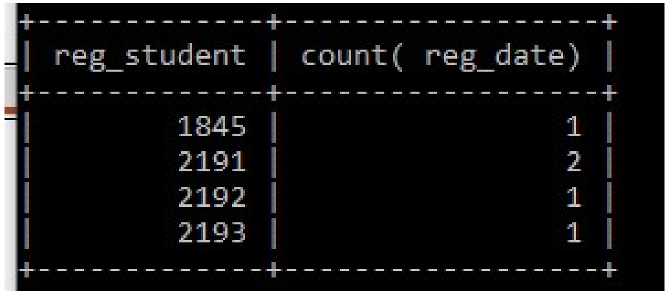


```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

SELECT reg_student, count(reg_date)

FROM `registration`

group by reg_student;



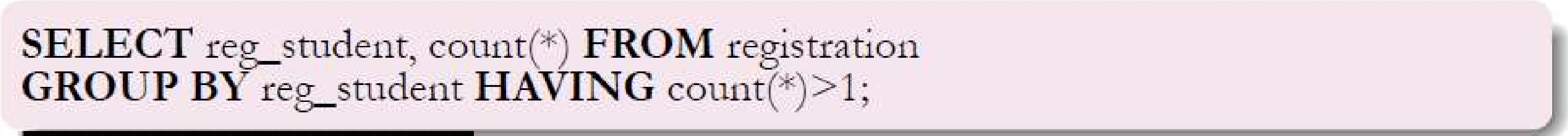
```
+-------------+-----------------+
| reg_student | count( reg_date) |
+-------------+-----------------+
|        1845 |               1 |
|        2191 |               2 |
|        2192 |               1 |
|        2193 |               1 |
+-------------+-----------------+
```

Aggregate Functions

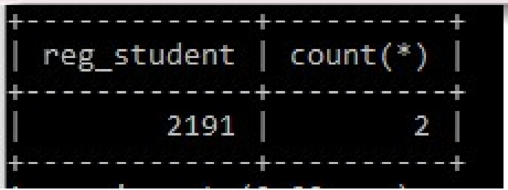
- These functions perform a calculation on values of records across one or more columns and return an aggregated value
- These column(s) are passed as arguments
- Can be used with Group BY clause to calculate an aggregated value per group of records
  - SUM: returns the total sum of non-NULL values
  - COUNT: returns the number of records
    - The keyword DISTINCT can be used in the SELECT clause for getting unique values of a column. It can also be used in COUNT() to get the number of unique values in a column.
  - MAX: returns the maximum value of non-NULL values
  - MIN: returns the minimum value of non-NULL values
  - AVG: returns the average of non-NULL values

Select - Having

- The HAVING clause defines a condition for filtering group of records (formed by the GROUP BY clause)
- If the condition must be applied on the data of the whole table – not on groups of records – should be in the WHERE clause
- **Having** : performs similar operation to the **Where clause BUT for groups of records** *(not individual records)*
- Select all students with more than one registration

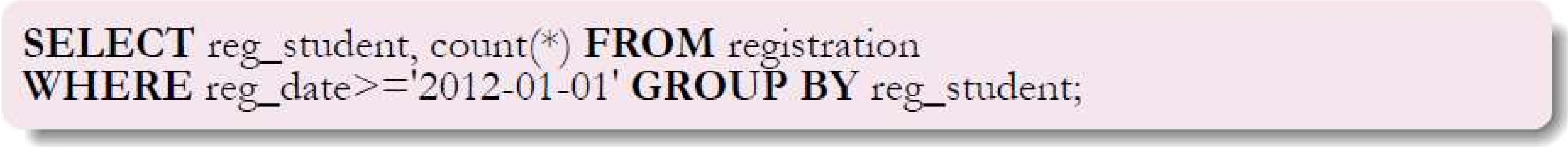


```
SELECT reg_student, count(*) FROM registration
GROUP BY reg_student HAVING count(*)>1;
```



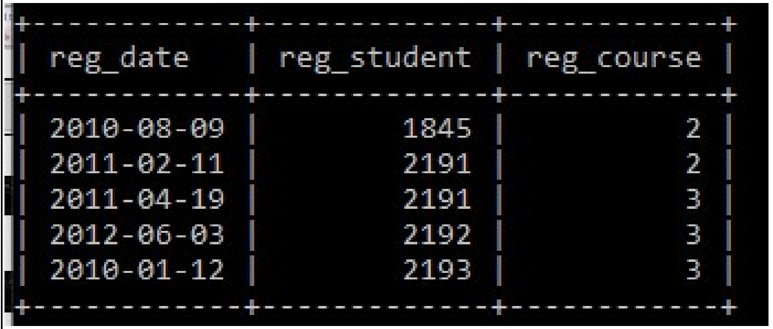
```
+-------------+----------+
| reg_student | count(*) |
+-------------+----------+
|        2191 |        2 |
+-------------+----------+
```

- Select each students’ AM and number of registration with registration date after 2012.



```
SELECT reg_student, count(*) FROM registration
WHERE reg_date>='2012-01-01' GROUP BY reg_student;
```

Table *Registration*

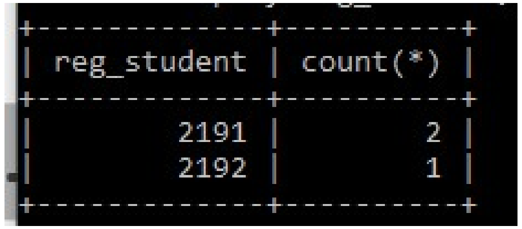


```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

SELECT reg_student, count(*) FROM registration

WHERE reg_date>= '2010-10-01'

Group by reg_student;



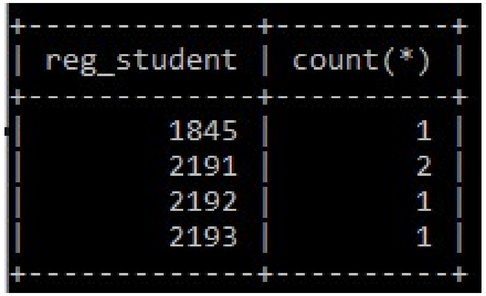
```
+-------------+----------+
| reg_student | count(*) |
+-------------+----------+
|        2191 |        2 |
|        2192 |        1 |
+-------------+----------+
```

This cannot be accomplished using the HAVING clause because the column reg_date is not included in the SELECT clause

SELECT reg_student, count(*)

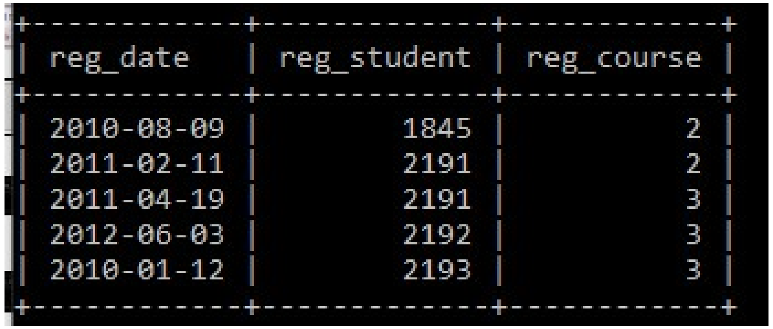
FROM registration

Group by reg_student



```
+-------------+----------+
| reg_student | count(*) |
+-------------+----------+
|        1845 |        1 |
|        2191 |        2 |
|        2192 |        1 |
|        2193 |        1 |
+-------------+----------+
```

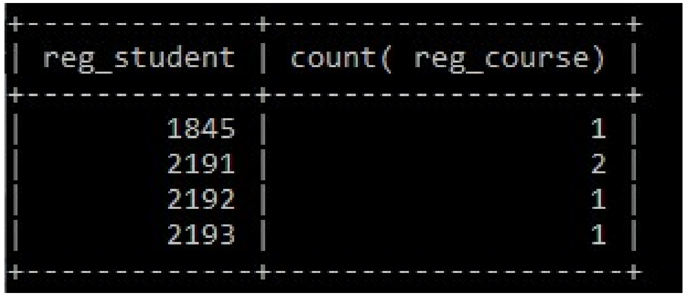
Table *Registration*



```
+------------+-------------+------------+
| reg_date   | reg_student | reg_course |
+------------+-------------+------------+
| 2010-08-09 |        1845 |          2 |
| 2011-02-11 |        2191 |          2 |
| 2011-04-19 |        2191 |          3 |
| 2012-06-03 |        2192 |          3 |
| 2010-01-12 |        2193 |          3 |
+------------+-------------+------------+
```

Count each student's courses

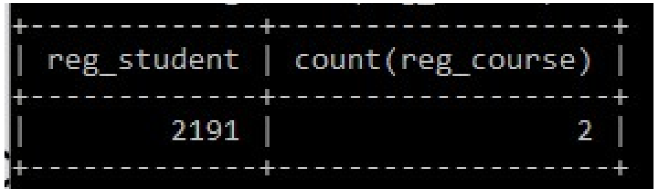
```
SELECT reg_student, count( reg_course)
FROM `registration`
group by reg_student;
```



```
+-------------+--------------------+
| reg_student | count( reg_course) |
+-------------+--------------------+
|        1845 |                  1 |
|        2191 |                  2 |
|        2192 |                  1 |
|        2193 |                  1 |
+-------------+--------------------+
```

From the previous step, select only the records with more than 1 courses

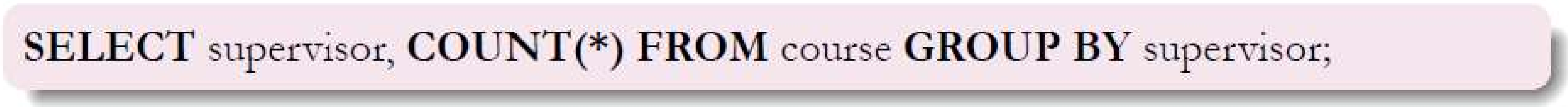
```
SELECT reg_student, count(reg_course)
FROM `registration`
group by reg_student
having count(reg_course)>1;
```



```
+-------------+-------------------+
| reg_student | count(reg_course) |
+-------------+-------------------+
|        2191 |                 2 |
+-------------+-------------------+
```

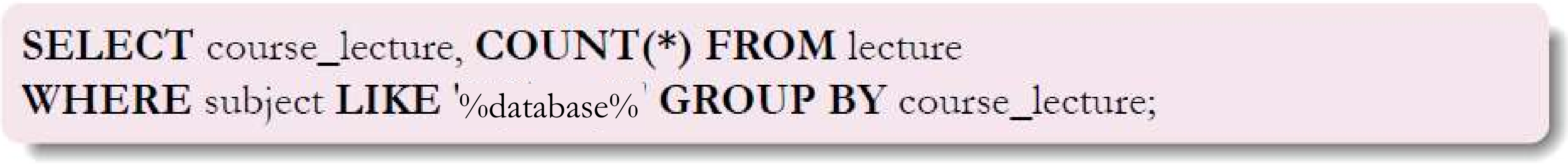
Select – Group By

- Select the number of courses supervised by each professor



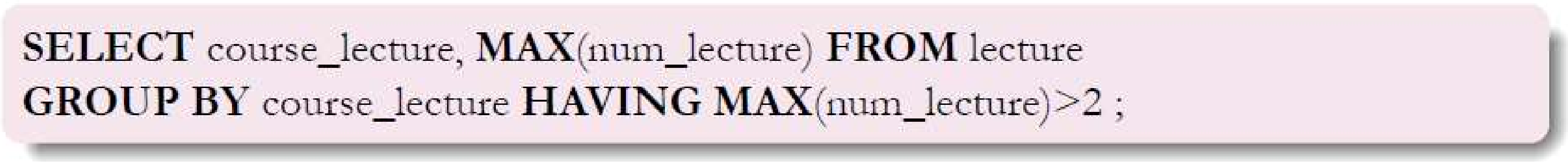
```
SELECT supervisor, COUNT(*) FROM course GROUP BY supervisor;
```

- For each course, select the number of lectures with subject relevant to databases



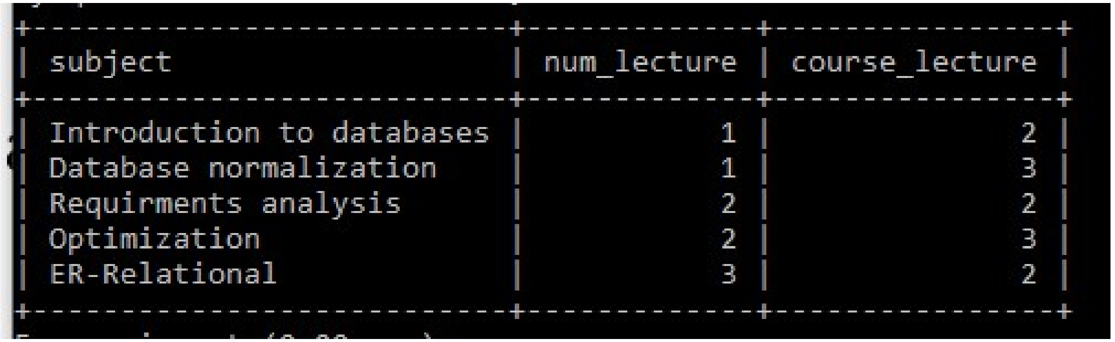
```
SELECT course_lecture, COUNT(*) FROM lecture
WHERE subject LIKE '%database%' GROUP BY course_lecture;
```

- Select course id and the maximum num_lecture for each course that its num_lecture is greater than 2.



```
SELECT course_lecture, MAX(num_lecture) FROM lecture
GROUP BY course_lecture HAVING MAX(num_lecture)>2 ;
```

Select * from lecture;



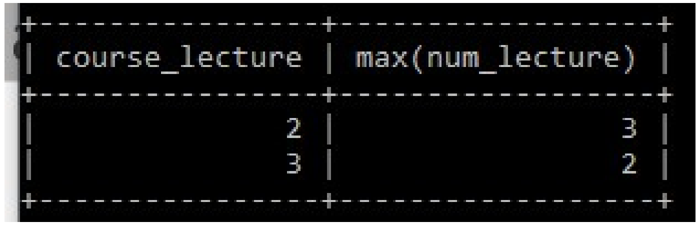
```
+---------------------------+--------------+----------------+
| subject                   | num_lecture  | course_lecture |
+---------------------------+--------------+----------------+
| Introduction to databases |            1 |              2 |
| Database normalization    |            1 |              3 |
| Requirments analysis      |            2 |              2 |
| Optimization              |            2 |              3 |
| ER-Relational             |            3 |              2 |
+---------------------------+--------------+----------------+
```

SELECT course_lecture, max(num_lecture)

from lecture

group by course_lecture;

Find the maximum num_lecture



```
+----------------+------------------+
| course_lecture | max(num_lecture) |
+----------------+------------------+
|              2 |                3 |
|              3 |                2 |
+----------------+------------------+
```

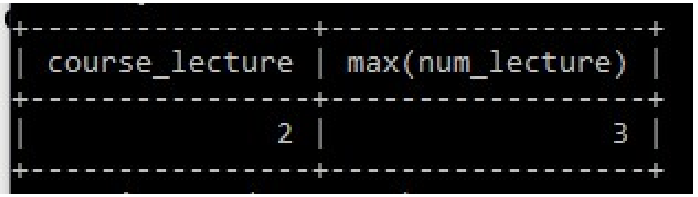
Select the records with max(num_lecture)>2

SELECT course_lecture, max(num_lecture)

from lecture

group by course_lecture

having max(num_lecture)>2;



```
+----------------+------------------+
| course_lecture | max(num_lecture) |
+----------------+------------------+
|              2 |                3 |
+----------------+------------------+
```

Select – Group By

- Correct use of GROUP BY in a SELECT statement
  1. The columns in the SELECT clause must
     - either also be included in the GROUP BY clause
     - or be functionally determined by the columns in the GROPU BY clause

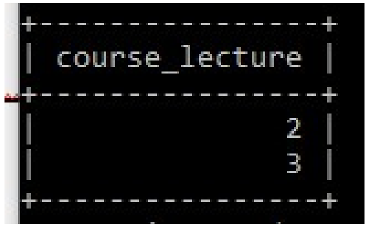
  2. Fields (columns) that
     - are not included in the GROUP BY clause
     - are not functionally determined by fields that are not included in the GROUP BY clause

     can only be present in the SELECT clause as arguments of an aggregate function, i.e., count .

Select – Group By

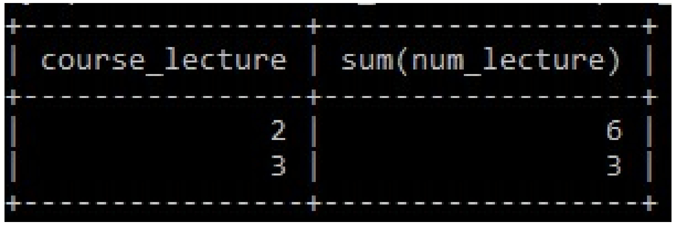
- Correct use of a GROUP BU clause in a SELECT statement

1. SELECT course_lecture FROM lecture group by course_lecture;



```
+----------------+
| course_lecture |
+----------------+
|              2 |
|              3 |
+----------------+
```

2. SELECT course_lecture, sum(num_lecture) FROM lecture group by course_lecture;

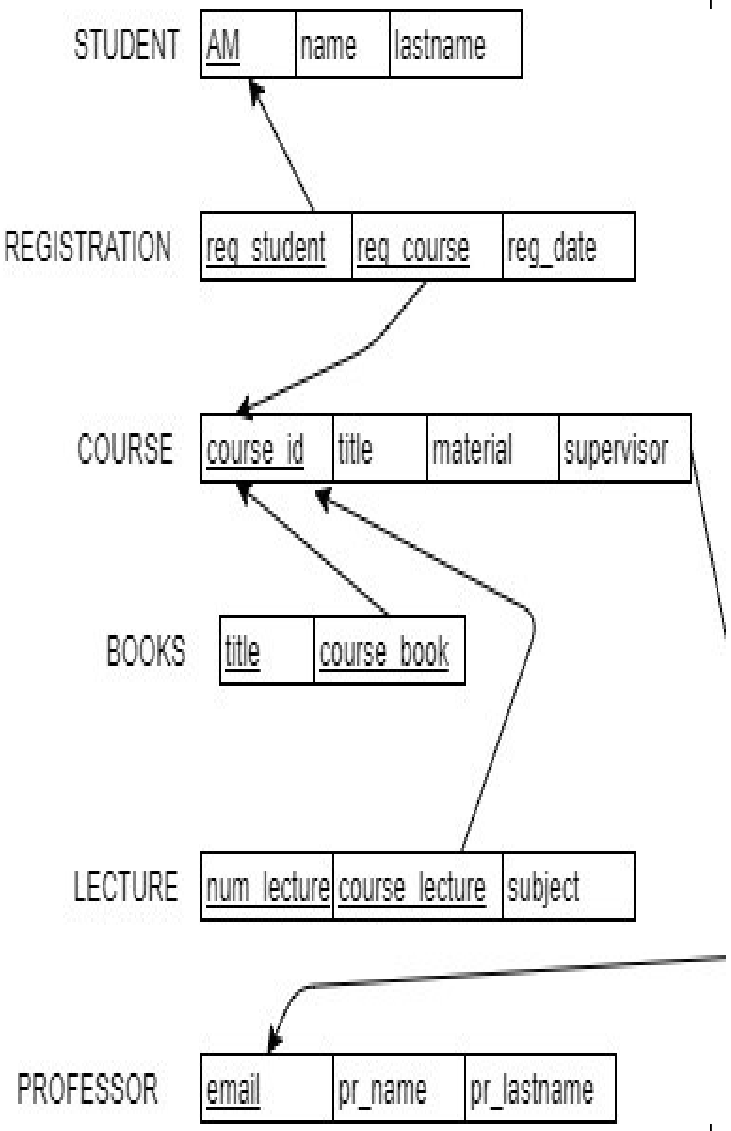


```
+----------------+------------------+
| course_lecture | sum(num_lecture) |
+----------------+------------------+
|              2 |                6 |
|              3 |                3 |
+----------------+------------------+
```

Data from multiple tables: WHEN?

- The required data are stored in more than one table
- These tables are linked by related columns, known as **foreign key** columns
- Tables must be combined based on these columns to get meaningful data.
  - Linking tables without using foreign key columns, the Cartesian product of the tables is returned.

- Example
  - Find the names of students that have registered for courses later than2012
  - Which tables contain this information?



STUDENT: AM, name, lastname

REGISTRATION: req_student, req_course, reg_date

COURSE: course_id, title, material, supervisor

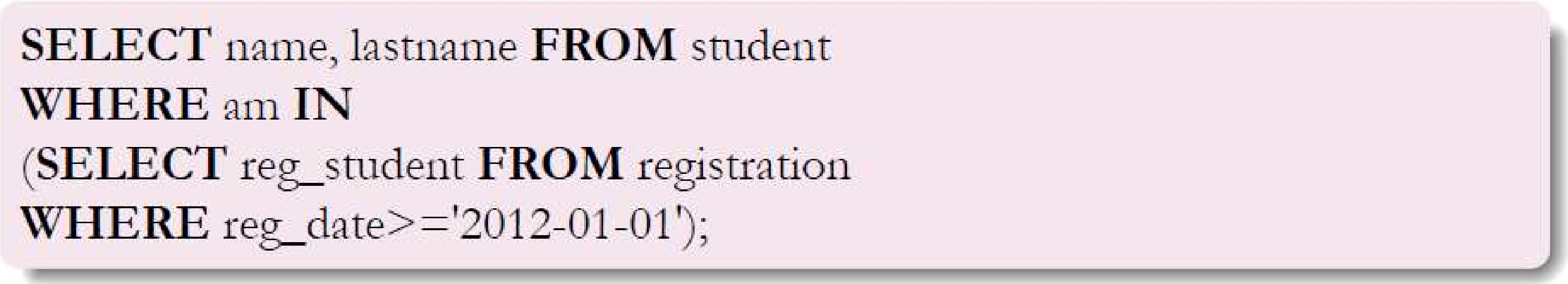
BOOKS: title, course_book

LECTURE: num_lecture, course_lecture, subject

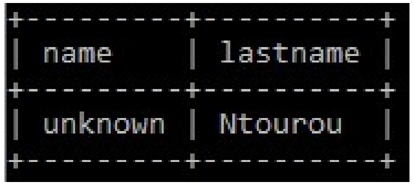
PROFESSOR: email, pr_name, pr_lastname

Data from multiple tables : 1. Subqueries – Nested queries

- A subquery is a query nested within another query
- A subquery is called inner query, while the query containing it is called outer query
- An inner query is evaluated first, and its result is returned to its outer query
- List the names and lastnames of students that registered for courses later than 2012



```
SELECT name, lastname FROM student
WHERE am IN
(SELECT reg_student FROM registration
WHERE reg_date>='2012-01-01');
```

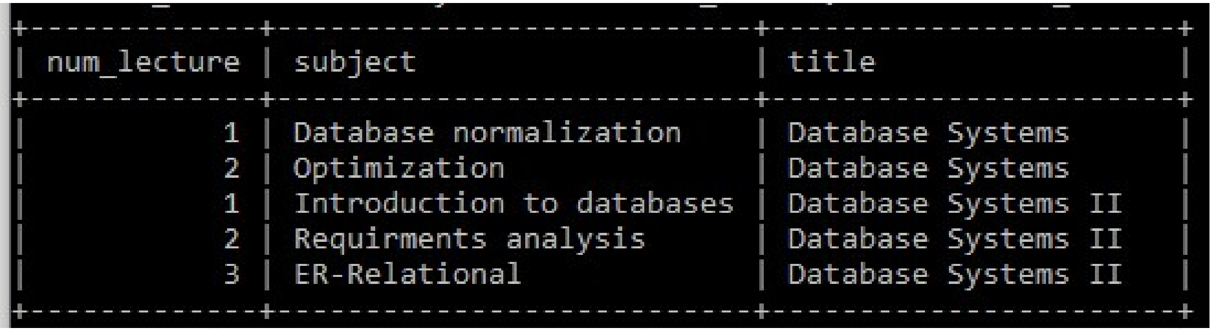


```
+---------+----------+
| name    | lastname |
+---------+----------+
| unknown | Ntourou  |
+---------+----------+
```

Data from multiple tables :
2. JOIN

- JOIN: method of combining data between tables based on values of the common column between the tables.
- Display the numbers of all lectures, the subject and the title of the course in included in, order by course_id and the number of lecture

```
SELECT lecture.num_lecture, lecture.subject, course.title
FROM lecture
INNER JOIN course ON course.course_id=lecture.course_lecture
ORDER BY course.course_id ASC, lecture.num_lecture ASC;
```



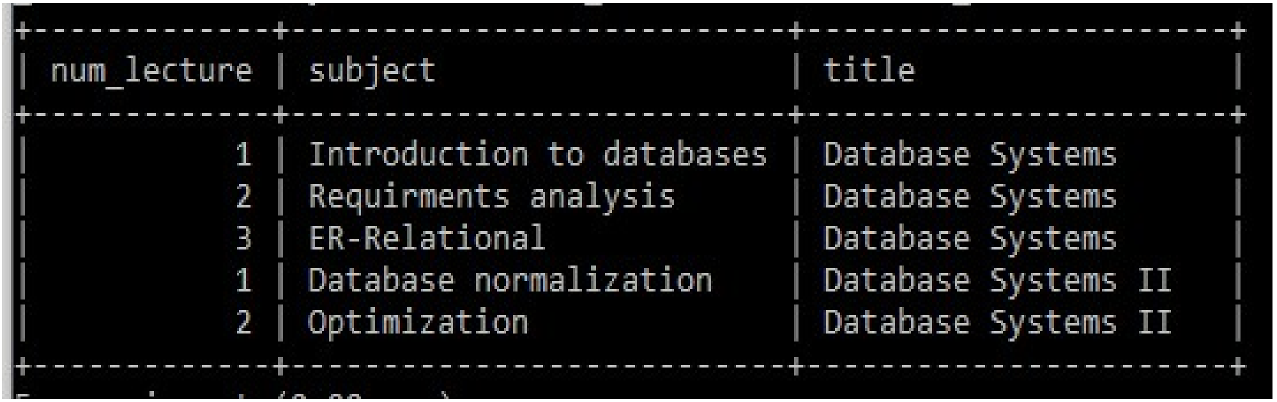
```
+-------------+----------------------------+----------------------+
| num_lecture | subject                    | title                |
+-------------+----------------------------+----------------------+
|           1 | Database normalization     | Database Systems     |
|           2 | Optimization               | Database Systems     |
|           1 | Introduction to databases  | Database Systems II  |
|           2 | Requirments analysis       | Database Systems II  |
|           3 | ER-Relational              | Database Systems II  |
+-------------+----------------------------+----------------------+
```

- INNER JOIN: returns all records from both tables that have matching values
- LEFT JOIN: returns all the records from the left table with any matching records from the right
- RIGHT JOIN: returns all the records from the right table with any matching records from the left

Data from multiple tables:
3. List tables

- All required tables are listed in the FROM clause separated by coma
- To avoid a Cartesian product, a join condition must be set in the WHERE clause.
    - Display the numbers of all lectures, the subject and the title of the course in included in, order by course_id and the number of lecture

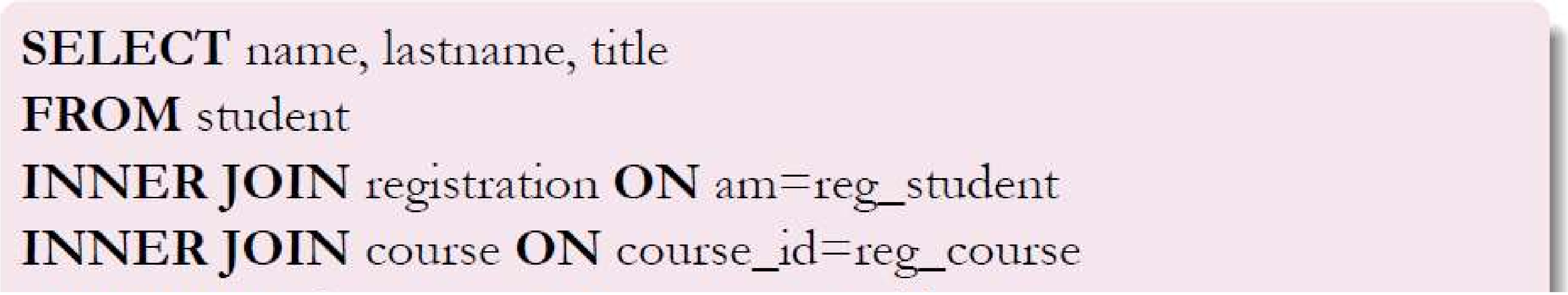
```
SELECT lecture.num_lecture, lecture.subject, course.title
FROM lecture, course
WHERE course.course_id=lecture.course_lecture
ORDER BY course.course_id ASC, lecture.num_lecture ASC;
```



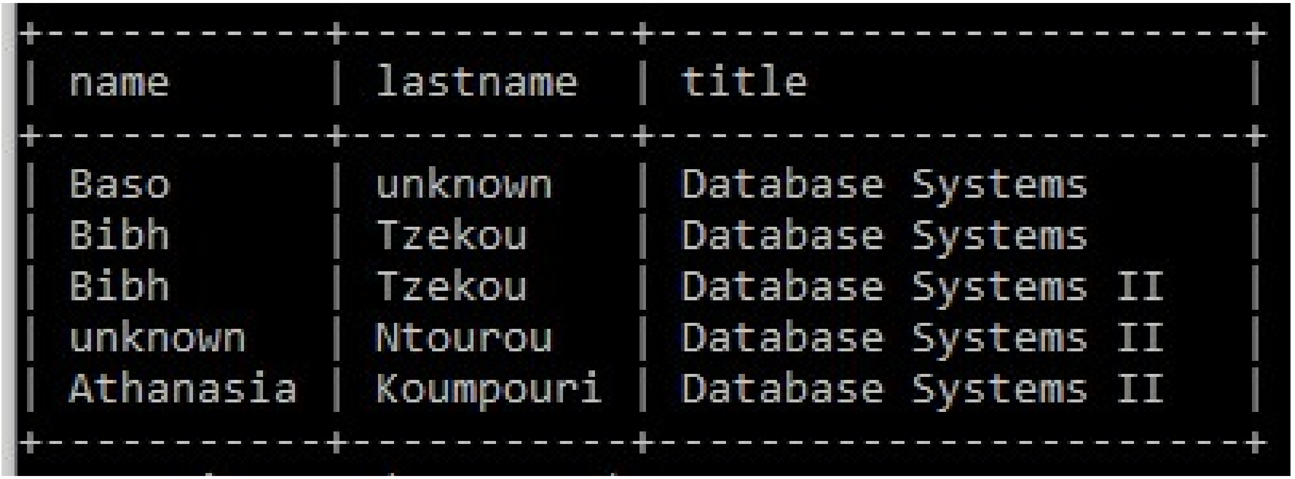
```
+-------------+----------------------------+----------------------+
| num_lecture | subject                    | title                |
+-------------+----------------------------+----------------------+
|           1 | Introduction to databases | Database Systems     |
|           2 | Requirments analysis       | Database Systems     |
|           3 | ER-Relational              | Database Systems     |
|           1 | Database normalization     | Database Systems II  |
|           2 | Optimization               | Database Systems II  |
+-------------+----------------------------+----------------------+
```

Query on **M:N**

- List the name and lastname of students together with the title of all courses they have registered for



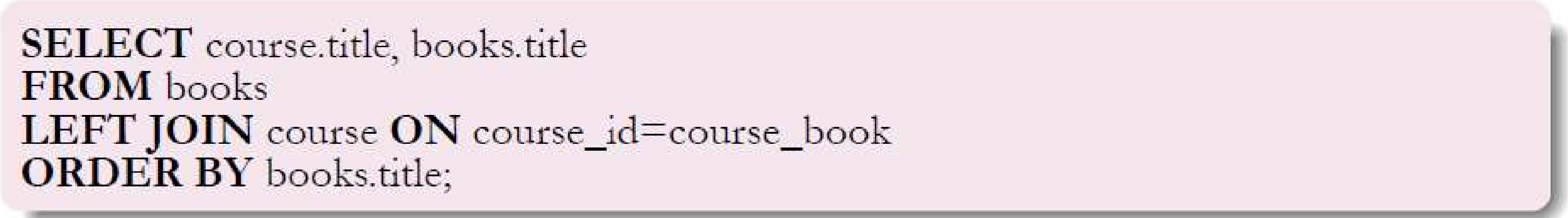
```
SELECT name, lastname, title
FROM student
INNER JOIN registration ON am=reg_student
INNER JOIN course ON course_id=reg_course
```



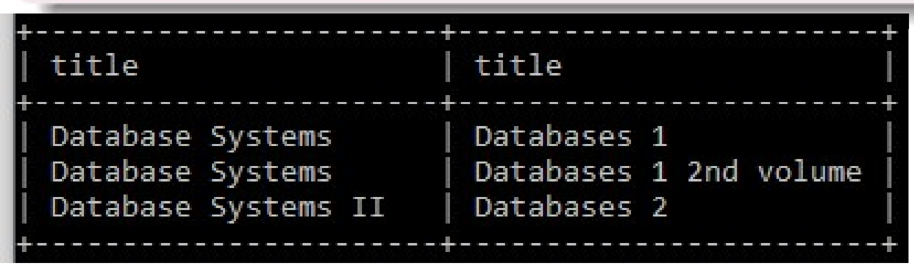
```
+-----------+-----------+----------------------+
| name      | lastname  | title                |
+-----------+-----------+----------------------+
| Baso      | unknown   | Database Systems     |
| Bibh      | Tzekou    | Database Systems     |
| Bibh      | Tzekou    | Database Systems II  |
| unknown   | Ntourou   | Database Systems II  |
| Athanasia | Koumpouri | Database Systems II  |
+-----------+-----------+----------------------+
```

Examples with JOIN

- Display the title of every book and the title of the course it is used in.

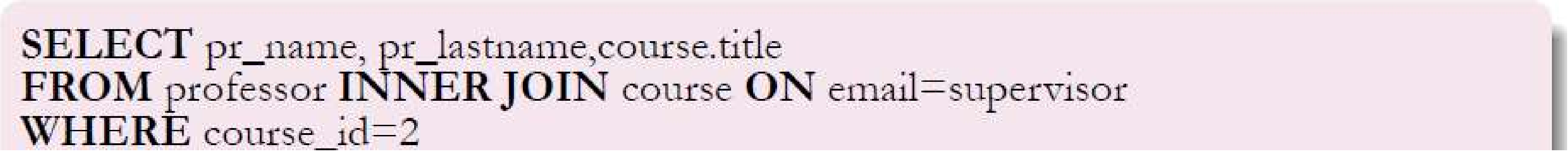


```
SELECT course.title, books.title
FROM books
LEFT JOIN course ON course_id=course_book
ORDER BY books.title;
```



```
+-----------------------+-----------------------+
| title                 | title                 |
+-----------------------+-----------------------+
| Database Systems      | Databases 1           |
| Database Systems      | Databases 1 2nd volume |
| Database Systems II   | Databases 2           |
+-----------------------+-----------------------+
```

- Display the name and the laty name of the professor who supervises the course with id 2 together with its title

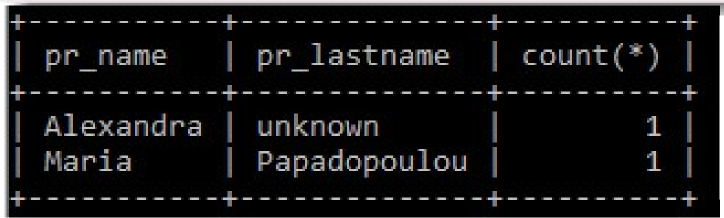


```
SELECT pr_name, pr_lastname,course.title
FROM professor INNER JOIN course ON email=supervisor
WHERE course_id=2
```

Examples with JOIN

- Display the name and lastname of the professors that supervise at least one course along with the number of the courses they supervise

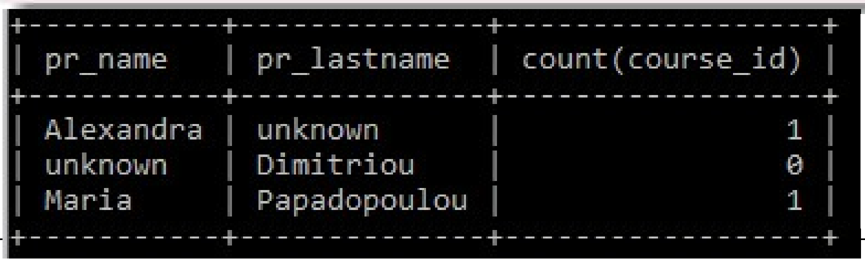
```
SELECT pr_name, pr_lastname, count(*)
FROM professor
INNER JOIN course ON supervisor=email
GROUP BY supervisor;
```



```
+-----------+-------------+----------+
| pr_name   | pr_lastname | count(*) |
+-----------+-------------+----------+
| Alexandra | unknown     |        1 |
| Maria     | Papadopoulou |       1 |
+-----------+-------------+----------+
```

- Display the name and lastname of all the professors and the number of the courses they supervise

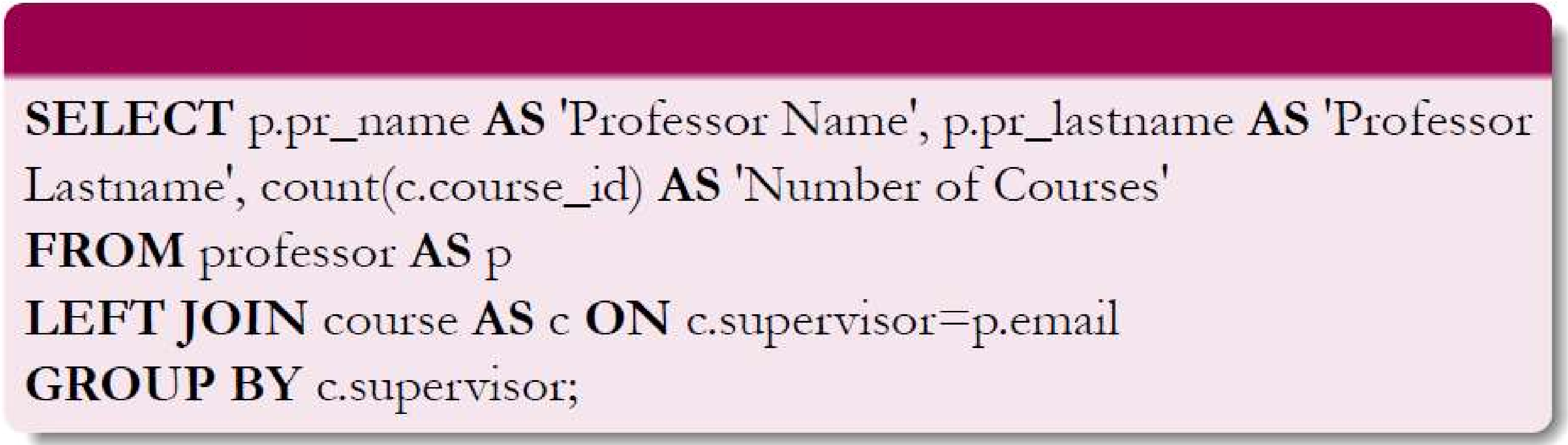
```
SELECT pr_name, pr_lastname, count(course_id)
FROM professor
LEFT JOIN course ON supervisor=email
GROUP BY supervisor;
```



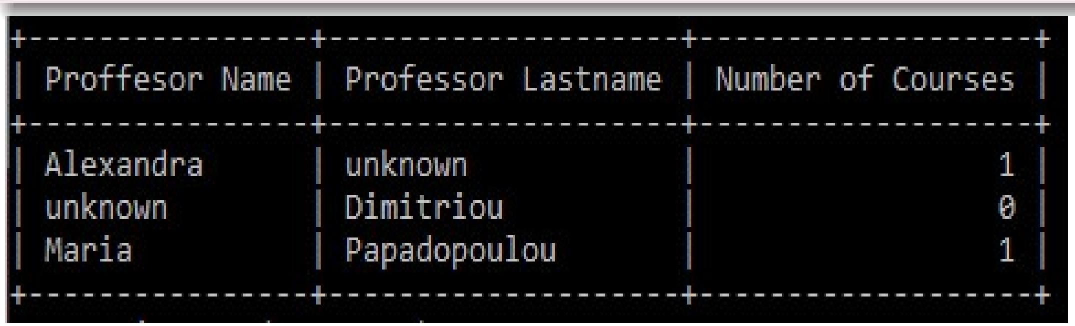
```
+-----------+--------------+------------------+
| pr_name   | pr_lastname  | count(course_id) |
+-----------+--------------+------------------+
| Alexandra | unknown      |                1 |
| unknown   | Dimitriou    |                0 |
| Maria     | Papadopoulou |                1 |
+-----------+--------------+------------------+
```

Aliases

- The keyword **as** can be used to give columns or tables an alias, that is a temporary name for the duration of the query.
- Using an alias for a column or table can make queries more readable and easily understood



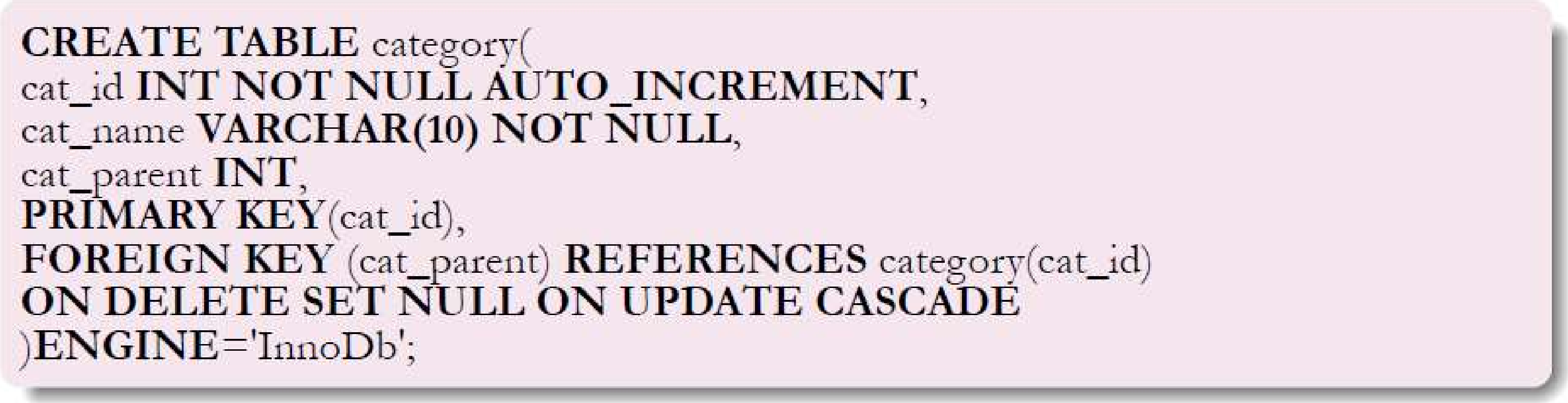
```
SELECT p.pr_name AS 'Professor Name', p.pr_lastname AS 'Professor Lastname', count(c.course_id) AS 'Number of Courses'
FROM professor AS p
LEFT JOIN course AS c ON c.supervisor=p.email
GROUP BY c.supervisor;
```



```
+----------------+-------------------+-------------------+
| Proffesor Name | Professor Lastname | Number of Courses |
+----------------+-------------------+-------------------+
| Alexandra      | unknown           |                 1 |
| unknown        | Dimitriou         |                 0 |
| Maria          | Papadopoulou      |                 1 |
+----------------+-------------------+-------------------+
```

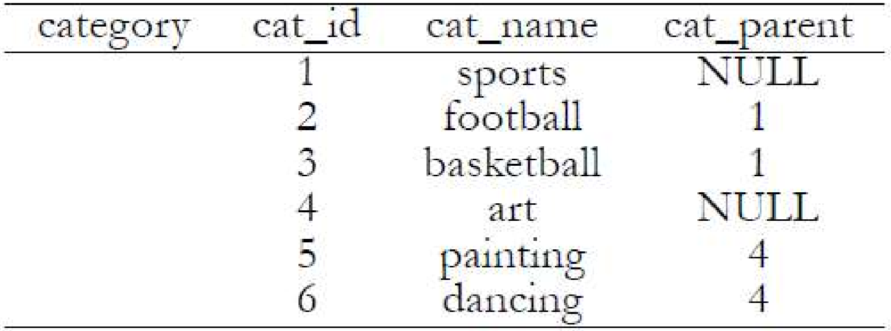
Aliases used in self join(1/3)

- A self join joins a table with itself
- Aliases must be used to achieve a self join
- For example:



```
CREATE TABLE category(
cat_id INT NOT NULL AUTO_INCREMENT,
cat_name VARCHAR(10) NOT NULL,
cat_parent INT,
PRIMARY KEY(cat_id),
FOREIGN KEY (cat_parent) REFERENCES category(cat_id)
ON DELETE SET NULL ON UPDATE CASCADE
)ENGINE='InnoDb';
```

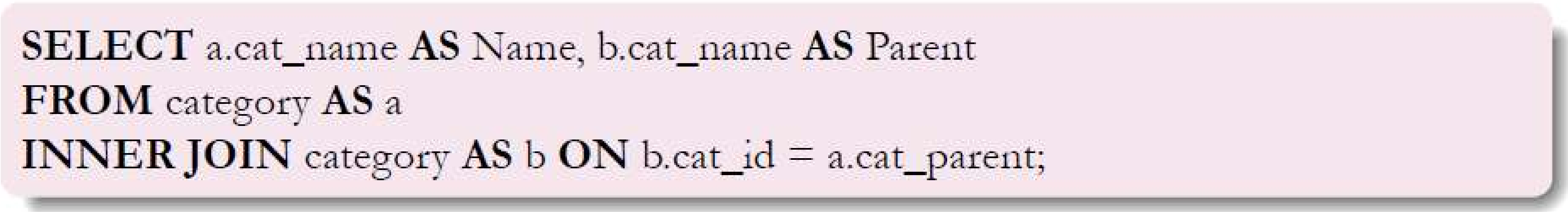
  - with data:



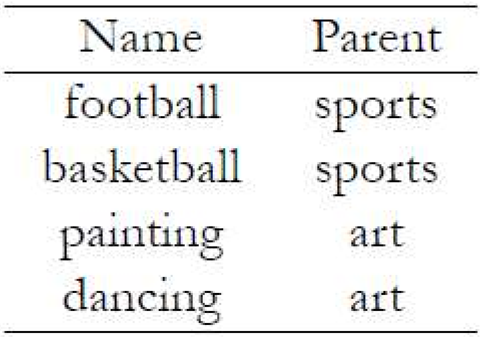
category	cat_id	cat_name	cat_parent
	1	sports	NULL
	2	football	1
	3	basketball	1
	4	art	NULL
	5	painting	4
	6	dancing	4

Aliases used in self join(2/3)

- List the names of categories that have parental categories along with the names of the parental categories



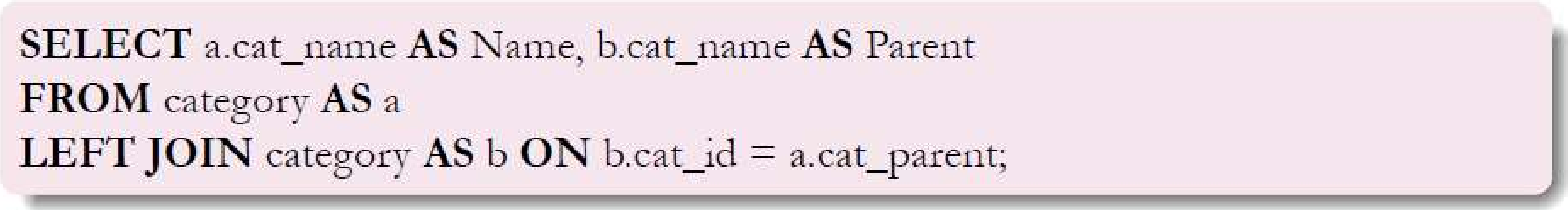
```
SELECT a.cat_name AS Name, b.cat_name AS Parent
FROM category AS a
INNER JOIN category AS b ON b.cat_id = a.cat_parent;
```



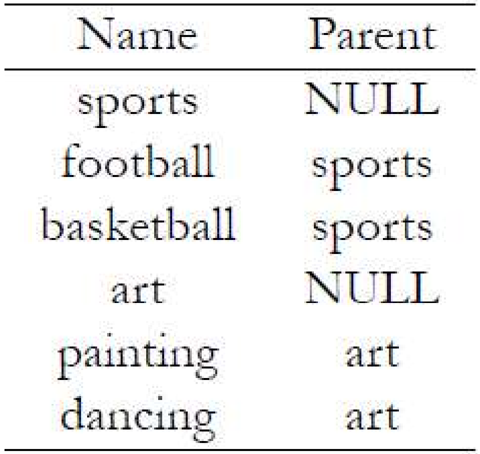
Name	Parent
football	sports
basketball	sports
painting	art
dancing	art

Aliases used in self join(**3/3**)

- List the names of **all** categories and the names of their parental categories

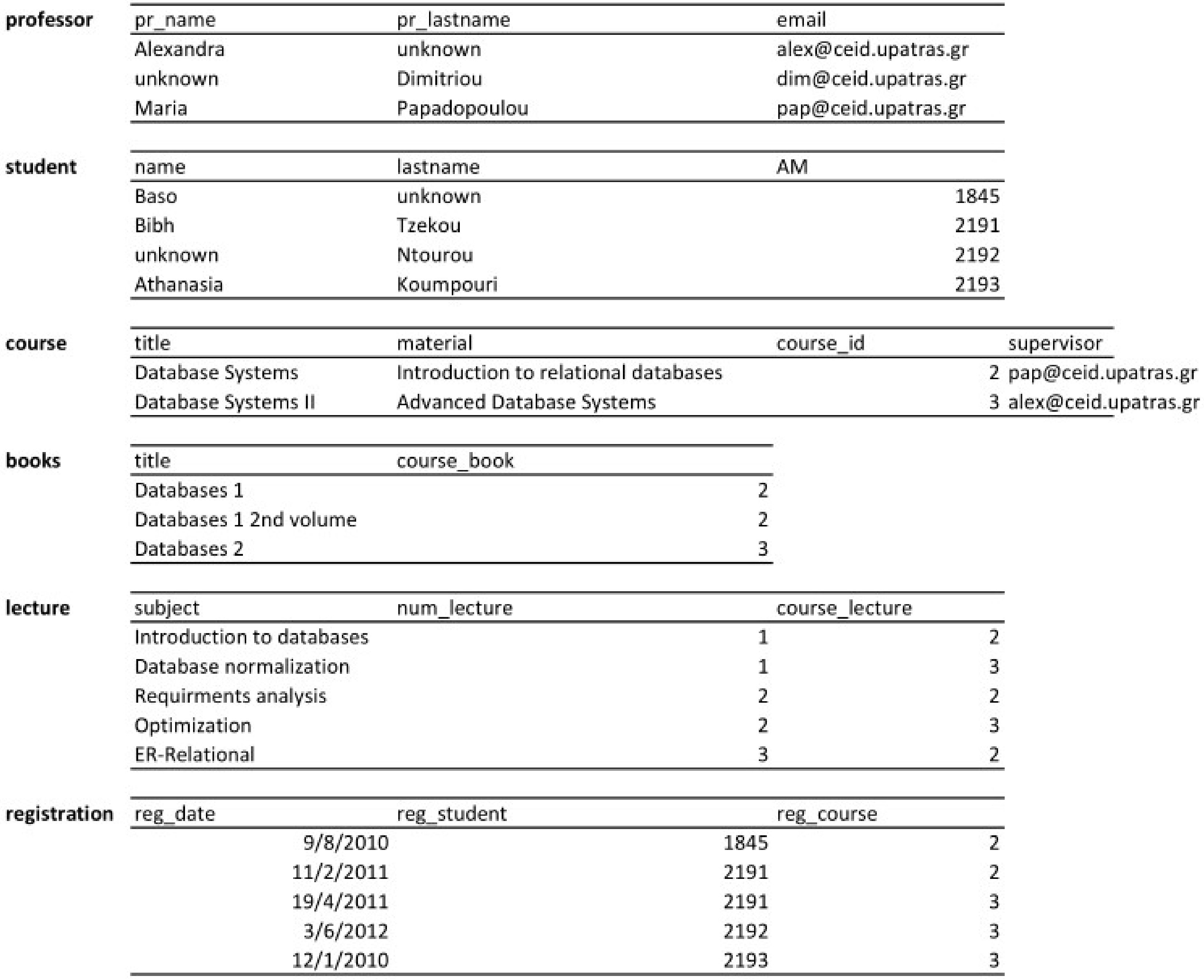


```
SELECT a.cat_name AS Name, b.cat_name AS Parent
FROM category AS a
LEFT JOIN category AS b ON b.cat_id = a.cat_parent;
```



Name	Parent
sports	NULL
football	sports
basketball	sports
art	NULL
painting	art
dancing	art

Data for example database



professor

pr_name	pr_lastname	email
Alexandra	unknown	alex@ceid.upatras.gr
unknown	Dimitriou	dim@ceid.upatras.gr
Maria	Papadopoulou	pap@ceid.upatras.gr

student

name	lastname	AM
Baso	unknown	1845
Bibh	Tzekou	2191
unknown	Ntourou	2192
Athanasia	Koumpouri	2193

course

title	material	course_id	supervisor
Database Systems	Introduction to relational databases	2	pap@ceid.upatras.gr
Database Systems II	Advanced Database Systems	3	alex@ceid.upatras.gr

books

title	course_book
Databases 1	2
Databases 1 2nd volume	2
Databases 2	3

lecture

subject	num_lecture	course_lecture
Introduction to databases	1	2
Database normalization	1	3
Requirments analysis	2	2
Optimization	2	3
ER-Relational	3	2

registration

reg_date	reg_student	reg_course
9/8/2010	1845	2
11/2/2011	2191	2
19/4/2011	2191	3
3/6/2012	2192	3
12/1/2010	2193	3