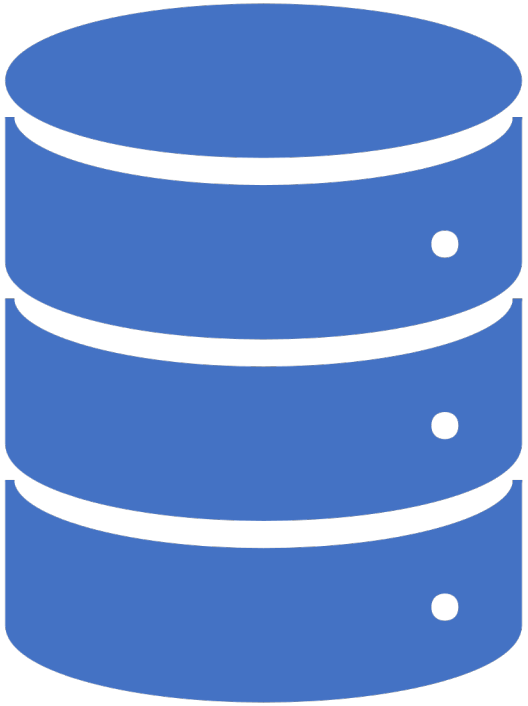


Seminar 2

SQL – Data Manipulation Language (DML).

The SELECT statement

SQL – Data Manipulation Language (DML)

DML – contains commands to insert, update, delete and query the data from the database.

DML instructions:

- ***INSERT*** – insert new records in the tables
- ***UPDATE*** – modify / update the records from the tables
- ***DELETE*** – delete the records from the tables
- ***SELECT*** – extract information from the tables

INSERT command– used to add records in the tables

INSERT INTO table_name(column_name_1, column_name_2, …)]
VALUES (value_1, value_2, …)

OR

INSERT INTO table_name
VALUES (value_1, value_2, …)

SQL – Data Manipulation Language (DML)

INSERT command

- The column_name specification is optional
- By specifying the name of the columns are obtained associations *column_name-value*, so the order in which the values will be added is not necessarily the order from *create table* statement or *alter table* statement, performed after
- If the column_name specification is not used, the order in which the values are added is the order from the *create table* statement or *alter table* statement, performed after
- If there is no value specified for a column_name, SQL Server will check the *default* constraint for that column_name and it will fulfill it; if the *default* constraint was not defined for that column, but it is defined *not null* then the insert could not be fullfilled with success (an error message will be displayed)
- On an *IDENTITY* column_name, no value should be added (it is automatically inserted)

SQL – Data Manipulation Language (DML)

INSERT command- examples

- With column names

```
INSERT INTO Students(Sid, Name, Surname, Dob, Email, Gid)
VALUES (1, 'Mihnea', 'Dan', '03/09/2000', 'dan@uc.ro', 822)
```

- Without column names

```
INSERT INTO Students VALUES (2, 'Mailat', 'Mihaela', '11/08/2001', 'mm@ucs.ro', 822)
```

- Multiple values

```
INSERT INTO Students VALUES
(3, 'Roman', 'Paul', '01/10/2001', 'paul@ucs.ro', 221),
(4, 'Romaniuc', 'Loredana', '05/11/2000', 'lore@us.ro', 221),
(5, 'Cristea', 'Mihai', '11/12/2001', 'mcristea@uc.ro', 221),
(6, 'Pitic', 'Mirela', '07/05/2000', 'pmirela@ucs.ro', 924)
```

SQL – Data Manipulation Language (DML)

UPDATE command – used to change the values of the records from the tables

UPDATE table_name
SET column_name_1=expression_1 , column_name_2=expression_2, …
WHERE condition_column_name(s)

- All the records from the table are modified if the *WHERE* clause is missing
- Examples:

```
UPDATE Students
SET Dob='09/08/2020', Gid=225
WHERE Name='Roman' AND Surname='Paul’

UPDATE Groups
SET NoOfStudents=NoOfStudents+1
```

SQL – Data Manipulation Language (DML)

DELETE command – used to remove the records from the tables

DELETE FROM table_name
WHERE condition_column_name(s)

- All the records from the table are deleted if the *WHERE* clause is missing
- Examples:
  - Delete all the records from the table

    DELETE FROM Students

  - Delete the groups that have the id betweem 221 and 227

    DELETE FROM Groups
    WHERE Gid BETWEEN 221 AND 227

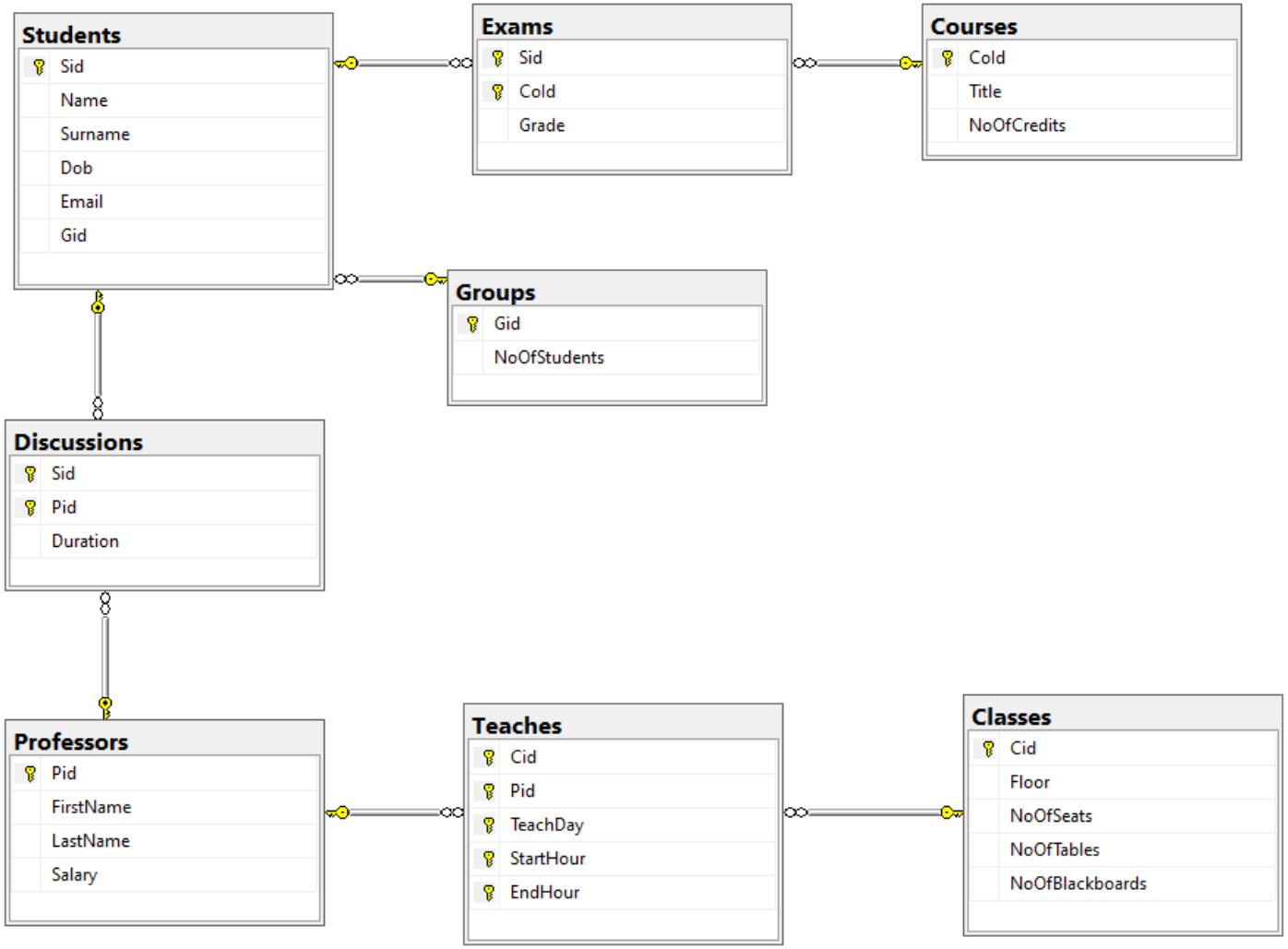
The SELECT statement

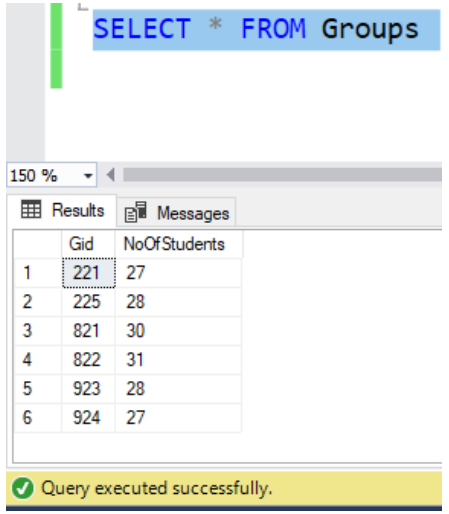
Consider the Faculty database:

```
CREATE TABLE Groups(
Gid INT PRIMARY KEY,
NoOfStudents INT)

CREATE TABLE Students(
Sid INT PRIMARY KEY,
Name VARCHAR(20) NOT NULL,
Surname VARCHAR(30),
Dob DATE DEFAULT '01/01/2000',
Email VARCHAR(50),
Gid INT FOREIGN KEY REFERENCES Groups(Gid))

CREATE TABLE Classes(
Cid INT PRIMARY KEY IDENTITY(1,1),
Floor SMALLINT,
NoOfSeats INT,
NoOfTables INT,
NoOfBlackboards INT DEFAULT 1)

CREATE TABLE Courses(
CoId INT PRIMARY KEY,
Title VARCHAR(40),
NoOfCredits INT)

CREATE TABLE Professors(
Pid INT PRIMARY KEY IDENTITY,
FirstName VARCHAR(20) NOT NULL,
LastName VARCHAR(30),
Salary INT CHECK (Salary BETWEEN 1000 AND 2000))

CREATE TABLE Discussions(
Sid INT FOREIGN KEY REFERENCES Students(Sid),
Pid INT FOREIGN KEY REFERENCES Professors(Pid),
Duration INT
PRIMARY KEY(Sid, Pid))

CREATE TABLE Teaches(
Cid INT FOREIGN KEY REFERENCES Classes(Cid),
Pid INT FOREIGN KEY REFERENCES Professors(Pid),
TeachDay VARCHAR(20) CHECK (TeachDay IN ('Monday', 'Tuesday', 'Wendsday', 'Thursday', 'Friday')),
StartHour TIME,
EndHour TIME
PRIMARY KEY(Cid, Pid, TeachDay, StartHour, EndHour))

CREATE TABLE Exams(
Sid INT FOREIGN KEY REFERENCES Students(Sid),
CoId INT FOREIGN KEY REFERENCES Courses(CoId),
Grade INT
PRIMARY KEY(Sid, CoId))
```

The SELECT statement

Consider the Faculty database diagram:





	Gid	NoOfStudents
1	221	27
2	225	28
3	821	30
4	822	31
5	923	28
6	924	27

The SELECT statement

Consider the Faculty database diagram:

	Gid	NoOfStudents
1	221	27
2	225	28
3	821	30
4	822	31
5	923	28
6	924	27

	Sid	Name	Surname	Dob	Email	Gid
1	1	Mihnea	Dan	2000-03-09	dan@uc.ro	822
2	2	Mailat	Mihaela	2001-11-08	mm@ucs.ro	822
3	3	Roman	Paul	2001-01-10	paul@ucs.ro	221
4	4	Romaniuc	Loredana	2000-05-11	lore@us.ro	221
5	5	Cristea	Mihai	2001-11-12	mcristea@...	221
6	6	Pitic	Mirela	2000-07-05	pmirela@u...	924

	Cid	Floor	NoOfSeats	NoOfTables	NoOfBlackboards
1	1	3	23	20	1
2	2	3	25	30	2
3	3	2	100	100	2
4	4	1	200	100	3

	Pid	FirstName	LastName	Salary
1	1	Dan	Kiraly	1200
2	2	Popescu	Raluca	1300
3	3	Voina	Claudia	2000
4	4	Maier	Clementi...	1550

	Cid	Pid	TeachDay	StartHour	EndHour
1	1	1	Monday	09:20:00.0000000	11:10:00.0000000
2	1	2	Friday	09:20:00.0000000	11:10:00.0000000
3	1	4	Thursday	18:30:00.0000000	20:20:00.0000000
4	2	1	Monday	07:30:00.0000000	09:20:00.0000000
5	3	3	Wendsday	14:50:00.0000000	16:40:00.0000000

	Sid	Pid	Duration
1	1	1	30
2	1	3	45
3	2	1	35
4	2	4	60
5	3	4	20

	CoId	Title	NoOfCredits
1	11	Databases	6
2	12	Operation Systems	5
3	22	Algebra	4
4	23	Geometry	4

	Sid	CoId	Grade
1	1	11	9
2	1	23	5
3	2	12	10
4	3	11	8
5	4	23	7

The SELECT statement

SELECT command – used to extract data form the database

SELECT column_name(s) FROM table_name

OR

SELECT * FROM table_name

The result of the *SELECT* is stored in a table called *result-set*

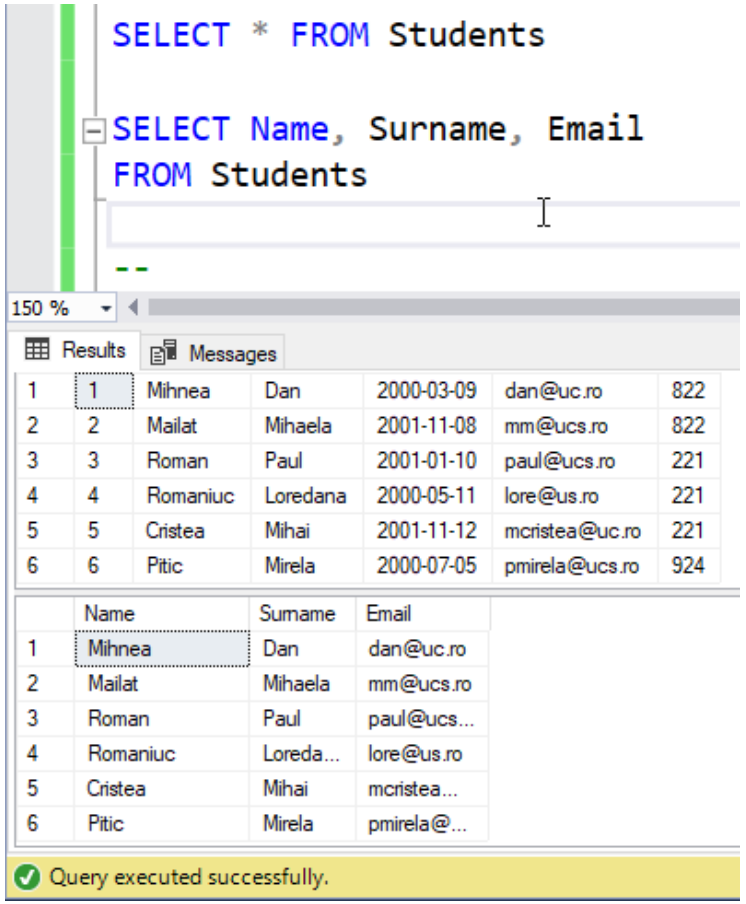
Example:

- Display all the students

  SELECT * FROM Students

- Display the name, the surname and the email of each student

  SELECT Name, Surname, Email
  FROM Students



The SELECT statement

SELECT command – used to extract data form the database

SELECT DISTINCT column_name(s) FROM table_name

To avoid the display of the duplicate values, use *DISTINCT*

Example:

- Display the days in which are teach activities.

```
SELECT TeachDay
FROM Teaches
```

- Display the distinct days in which are teach activities.

```
SELECT DISTINCT TeachDay
FROM Teaches
```

	TeachDay
1	Monday
2	Friday
3	Thursday
4	Monday
5	Wendsday

	TeachDay
1	Friday
2	Monday
3	Thursday
4	Wendsday

The SELECT statement

SELECT command – used to extract data form the database

SELECT column_name(s) FROM table_name
WHERE condition

- *Condition* is of form *column_name operator value*

Only the records that fulfill the *condition* from the *WHERE* clause, are displayed.

Example:

- Display the students from the group 822.

```
SELECT * FROM Students
WHERE Gid=822
```

- Display the distinct names and surnames of the students from the group 221 or the ones that have their name *Mihnea.*

```
SELECT DISTINCT Name, Surname
FROM Students
WHERE Gid=221 OR Name='Mihnea'
```

	Sid	Name	Surname	Dob	Email	Gid
1	1	Mihnea	Dan	2000-03-09	dan@uc.ro	822
2	2	Mailat	Mihaela	2001-11-08	mm@ucs.ro	822

	Name	Surname
1	Cristea	Mihai
2	Mihnea	Dan
3	Roman	Paul
4	Romaniuc	Loredana

The SELECT statement

Where operators

Operator	Description
=	Equal
<>, !=	Diferent
>	Greater
<	Smaller
>=	Greater equal
<=	Smaller equal
!>	No greater than
!<	No smaller than

SELECT column_name(s) FROM table_name
WHERE column_name LIKE pattern

Operator	Description
IN	In a specified set
NOT IN	Outside of a specified set
BETWEEN	In a closed interval
NOT BETWEEN	Outside of a closed interval
LIKE	Start, contain, finish with a given template _ replace 1 character % replace 0-n characters [charlist] any character from the list [^charlist] any character that is not in the list
NOT LIKE	Different than a given template
IS [NOT] NULL	The value is [not] NULL
cond1 AND/OR cond2	AND – both conditions fulfilled OR – at least one condition fulfilled

The SELECT statement

Alias (Range variables)

SELECT column_name(s) FROM table_name AS alias

- A table or a column can have an *alias*
- An *alias* is useful when the name of the column or the name of the table is to long or complex
- An *alias* is necessary when the name of the table appear more than once in the *FROM* clause
- An *alias* for the tables is recommended when using columns with the same name from different tables
  - Groups.Gid and Students.Gid OR G.Gid=S.Gid, where Groups G and Students S
- An *alias* is useful for calculated columns values
- The queries with *alias* become easier to be read
- Solve the ambiguity

Example: Display the students with the grade 9 (student name, course id and grade).

```
SELECT Name, CoId, Grade
FROM Students, Exams
WHERE Students.Sid=Exams.Sid AND Grade=9
                OR
                        SELECT S.Name, E.CoId, E.Grade
                        FROM Students S, Exams E
                        WHERE S.Sid=E.Sid AND E.Grade=9
```

The SELECT statement

Arithmetic expression and the LIKE operator - examples

- Display the students that have the name starting with *R.*
  SELECT Name, Email FROM Students
  WHERE Name LIKE 'R%’

- Display the students that have the name finishing with *R.*
  SELECT Name, Email FROM Students
  WHERE Name LIKE '%R’

- Display the students for which the name contains *R.*
  SELECT Name, Email FROM Students
  WHERE Name LIKE '%R%’

- Display the students that have the name starting with *R* and has at least 2 characters.
  SELECT Name, Email FROM Students
  WHERE Name LIKE 'R_%'

- Display the following calculations
  SELECT Floor, NoOfSeats*2,
  NoOfSeats+NoOfTables AS product,
  Total=NoOfBlackboards+NoOfSeats+NoOfTables
  FROM Classes

	Floor	(No column name)	product	Total
1	3	46	43	44
2	3	50	55	57
3	2	200	200	202
4	1	400	300	303

The SELECT statement

Arithmetic expression and the LIKE operator – other examples

- SELECT Name, Email FROM Students WHERE Name > 'h’
- SELECT Name, Email FROM Students WHERE Name IN (‘Roman’, ‘Romanciuc’)
- SELECT Name, Email FROM Students WHERE Name LIKE ‘%Ro%’
- SELECT Name, Email FROM Students WHERE Name LIKE ‘Ro_’
- SELECT Name, Email FROM Students WHERE Name LIKE ‘[abc]%’
- SELECT Name, Email FROM Students WHERE Name LIKE ‘^[abc]%’

- SELECT Floor, NoOfSeats FROM Classes WHERE NoOfSeats BETWEEN 10 AND 20
- SELECT Floor, NoOfSeats FROM Classes WHERE NoOfSeats NOT IN (15, 25)
- SELECT Floor, NoOfSeats FROM Classes WHERE NoOfSeats NOT BETWEEN 17 AND 19
- SELECT Floor, NoOfSeats FROM Classes WHERE NoOfSeats IS NOT NULL
- SELECT Floor, NoOfSeats FROM Classes WHERE NoOfSeats=12 OR NoOfBlackboards IS NULL

The SELECT statement

UNION, INTERSECT, EXCEPT - duplicate rows are eliminated

- UNION – Combine the results of 2 of more queries in a single result-set

  SELECT column_name_1, column_name_2, column_name_3 … FROM table_name_1 …
  UNION [ALL]
  SELECT column_name_11, column_name_22, column_name_33 … FROM table_name_2 …

- Each query must contain the same number of column and the data types of the columns has to be compatibles
- ***UNION ALL*** – include also the duplicates

Examples: Display the first name of the professors that have the salary*>1600* OR the last name starting with *K.*

Display the names of the students with the name starting with *R* union with the first names of the professors that have the salary <1580.

SELECT FirstName FROM Professors WHERE Salary>1600 UNION SELECT FirstName FROM Professors WHERE LastName LIKE 'K%'	SELECT Name FROM Students WHERE Name LIKE 'R_%' UNION ALL SELECT FirstName FROM Professors WHERE Salary<1580

The SELECT statement

UNION, INTERSECT, EXCEPT

- INTERSECT – perform the intersection between the 2 queries and return a single result-set that contains the records that fulfill the first *SELECT* AND also the second *SELECT* (if there are multiples queries, are performed the first 2, and then the result-set is INTERSECT with the following one, …)

  SELECT column_name_1, column_name_2, column_name_3 … FROM table_name_1 …
  INTERSECT
  SELECT column_name_11, column_name_22, column_name_33 … FROM table_name_2 …

Examples: Display the first name of the professors that have the salary*>1600 AND* the last name starting with *K.*

Display the first names of the professors that cannot be found in the names of the students and in the titles of the courses.

SELECT FirstName FROM Professors WHERE Salary>1600 INTERSECT SELECT FirstName FROM Professors WHERE LastName LIKE 'K%'	SELECT FirstName FROM Professors INTERSECT SELECT Name FROM Students INTERSECT SELECT Title FROM Courses

The SELECT statement

UNION, INTERSECT, EXCEPT

- EXCEPT – perform the diference between the 2 (or more) queries and return a single result-set that contains the records that fulfill the first *SELECT* AND NOT fulfill the second *SELECT*

  SELECT column_name_1, column_name_2, column_name_3 … FROM table_name_1 …
  EXCEPT
  SELECT column_name_11, column_name_22, column_name_33 … FROM table_name_2 …

Examples: Display the first name of the professors that have the salary*>1600 BUT DON'T HAVE* the last name starting with *K.*
Display the courses with exams that have 5 credits but not 6 credits.

SELECT FirstName FROM Professors WHERE Salary>1600 EXCEPT SELECT FirstName FROM Professors WHERE LastName LIKE 'K%'	SELECT C.CoId FROM Courses C, Exams E WHERE C.CoId=E.CoId AND NoOfCredits=5 EXCEPT SELECT C.CoId FROM Courses C, Exams E WHERE C.CoId=E.CoId AND NoOfCredits=6

The SELECT statement

Nestes queries

- A query can contain another query inside of it (a subquery)
- In the *WHERE, FROM, HAVING* clauses
- The subquery is evaluated when testing the condition in the *WHERE* clause in the main query
- **IN** operator – tests whether a value belongs to a set of elements (explicity specified or generated by a query)
- **EXISTS** operator – tests whether a set is non-empty

- **ANY** operator – evaluated true if the condition is true for at least one item in the subquery results
- **ALL** operator – evaluated true if the condition is true for all the items in the subquery results
  - Compare a scalar value with a set of values provided from one column
  - **scalar_expression** [***= / <> / != / > / >= / !> / < / <= / !<***] [**SOME | ANY | ALL**] (subquery that has a result set of one column)

The SELECT statement

Nestes queries – examples

- Display the name of the students that are graded in course 11.

```
SELECT S.Name FROM Students S, Exams E
WHERE S.Sid=E.Sid AND CoId=11
```

```
SELECT S.Name
FROM Students S
WHERE S.Sid IN (SELECT E.Sid FROM Exams E WHERE CoId=11)
```

```
SELECT S.Name
FROM Students S
WHERE EXISTS (SELECT E.Sid FROM Exams E WHERE S.Sid=E.Sid AND CoId=11)
```

```
SELECT R.Name
FROM (SELECT S.Name FROM Students S, Exams E
WHERE S.Sid=E.Sid AND CoId=11) R
```

The SELECT statement

Nestes queries – examples

- Display the name of the students that **have not** discussed with the professor *Dan Kiraly.*

```
SELECT S.Name, S.Surname FROM Students S
WHERE S.Sid NOT IN (SELECT D.Sid FROM Discussions D, Professors P
WHERE D.Pid=P.Pid AND P.FirstName='Dan' AND P.LastName='Kiraly')

SELECT S.Name, S.Surname FROM Students S
WHERE NOT EXISTS (SELECT * FROM Discussions D, Professors P
WHERE D.Pid=P.Pid AND S.Sid=D.Sid AND P.FirstName='Dan' AND P.LastName='Kiraly’)
```

- Display the name of the students that **have** discussed with the professor *Dan Kiraly*.

```
SELECT R.Name, R.Surname FROM
(SELECT S.Sid, S.Name, S.Surname FROM Students S, Discussions D, Professors P
WHERE D.Pid=P.Pid AND S.Sid=D.Sid AND P.FirstName='Dan' AND P.LastName='Kiraly') R
```

The SELECT statement

Nestes queries – examples

ANY

- Display the courses that have the same number of credits as a course called *Databases.*

```
SELECT * FROM Courses C WHERE C.NoOfCredits = ANY (SELECT C1.NoOfCredits
FROM Courses C1 WHERE C1.Title='Databases')
```

- Display the courses that have a smaller number of credits than a course called *Databases.*

```
SELECT * FROM Courses C WHERE C.NoOfCredits < ANY (SELECT C1.NoOfCredits
FROM Courses C1 WHERE C1.Title='Databases’)
```

ALL

- Display the courses that have a smaller number of credits related to all the courses called *Databases.*

```
SELECT * FROM Courses C WHERE C.NoOfCredits < ALL (SELECT C1.NoOfCredits
FROM Courses C1 WHERE C1.Title='Databases')
```

The SELECT statement

JOIN operations

- Allows to extract data from multiple tables connected by relations that are established between columns from different tables.
- A single result-set is returned.
- The join between 2 tables involves:
  - The columns from each table (primary key from a table and the associated foreign key from the other table)
  - The logical operator (= or <>) used to compare the values from the columns

- **INNER JOIN** – extract the records that are connected in both tables
- **LEFT [OUTER] JOIN** – extract all the records from the left side table (even if exist or not matchings in the right side table) and the records that are connected in both tables
- **RIGHT [OUTER] JOIN** – extract all the records from the right side table (even if exist or not matchings in the left side table) and the records that are connected in both tables
- **FULL [OUTER] JOIN** – a combination between LEFT OUTER JOIN + RIGHT OUTER JOIN (all from the left side table and all from the right side table)

The SELECT statement

INNER JOIN

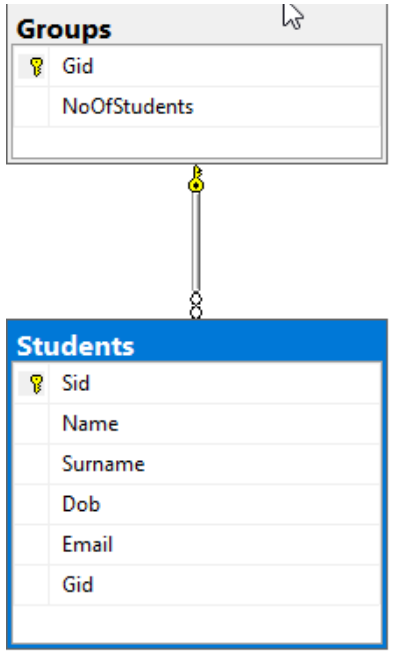
SELECT column_name(s)
FROM table_name_1 INNER JOIN table_name_2
ON table_name_1.column_name(pk) = table_name_2.column_name(fk)

Examples: 1:n relationship: Groups – Students

SELECT * FROM Groups
SELECT * FROM Students

- Display the students from each group.

```
SELECT *
FROM Groups G, Students S
WHERE G.Gid=S.Gid

-- equivalent

SELECT *
FROM Groups G INNER JOIN Students S ON G.Gid=S.Gid
```



	Gid	NoOfStudents
1	221	27
2	225	28
3	821	30
4	822	31
5	923	28
6	924	27

	Sid	Name	Surname	Dob	Email	Gid
1	1	Mihnea	Dan	2000-03-09	dan@uc.ro	822
2	2	Mailat	Mihaela	2001-11-08	mm@ucs.ro	822
3	3	Roman	Paul	2001-01-10	paul@ucs.ro	221
4	4	Romaniuc	Loredana	2000-05-11	lore@us.ro	221
5	5	Cristea	Mihai	2001-11-12	mcristea@uc.ro	221
6	6	Pitic	Mirela	2000-07-05	pmirela@ucs.ro	924

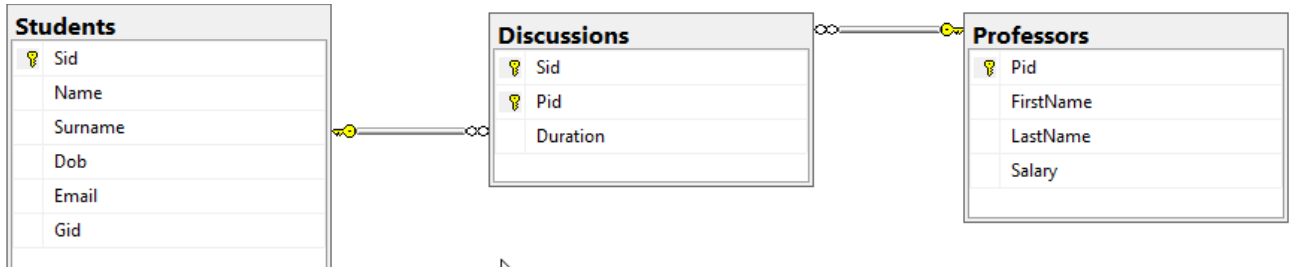
	Gid	NoOfStudents	Sid	Name	Surname	Dob	Email	Gid
1	822	31	1	Mihnea	Dan	2000-03-09	dan@uc.ro	822
2	822	31	2	Mailat	Mihaela	2001-11-08	mm@ucs.ro	822
3	221	27	3	Roman	Paul	2001-01-10	paul@ucs.ro	221
4	221	27	4	Rom...	Loredana	2000-05-11	lore@us.ro	221
5	221	27	5	Cristea	Mihai	2001-11-12	mcristea@uc.ro	221
6	924	27	6	Pitic	Mirela	2000-07-05	pmirela@ucs.ro	924

	Gid	NoOfStudents	Sid	Name	Surname	Dob	Email	Gid
1	822	31	1	Mihnea	Dan	2000-03-09	dan@uc.ro	822
2	822	31	2	Mailat	Mihaela	2001-11-08	mm@ucs.ro	822
3	221	27	3	Roman	Paul	2001-01-10	paul@ucs.ro	221
4	221	27	4	Rom...	Loredana	2000-05-11	lore@us.ro	221
5	221	27	5	Cristea	Mihai	2001-11-12	mcristea@uc.ro	221
6	924	27	6	Pitic	Mirela	2000-07-05	pmirela@ucs.ro	924

The SELECT statement

Example: m:n relationship: Students - (Discussions) - Professors

SELECT * FROM Students
SELECT * FROM Discussions
SELECT * FROM Professors




```
-- m:n Students-Discussions-Professors
SELECT * FROM Students
SELECT * FROM Discussions
SELECT * FROM Professors
```

150 %

Results | Messages

	Sid	Name	Surname	Dob	Email	Gid
1	1	Mihnea	Dan	2000-03-09	dan@uc.ro	822
2	2	Mailat	Mihaela	2001-11-08	mm@ucs.ro	822
3	3	Roman	Paul	2001-01-10	paul@ucs.ro	221
4	4	Romaniuc	Loredana	2000-05-11	lore@us.ro	221
5	5	Cristea	Mihai	2001-11-12	mcristea@uc.ro	221
6	6	Pitic	Mirela	2000-07-05	pmirela@ucs.ro	924

	Sid	Pid	Duration
1	1	1	30
2	1	3	45
3	2	1	35
4	2	4	60
5	3	4	20

	Pid	FirstName	LastName	Salary
1	1	Dan	Kiraly	1200
2	2	Popescu	Raluca	1300
3	3	Voina	Claudia	2000
4	4	Maier	Clementi...	1550

Try by yourself ☺
m:n relationship: Students – (Exams) – Courses
(like in the next slide)

The SELECT statement

Example: m:n relationship: Students - (Discussions) - Professors

JOIN operator	Query	Result
INNER JOIN Display correspondingly to each student, the professors to which have interact.	SELECT S.Name, S.Surname, S.Email, P.FirstName, P.LastName FROM Students S INNER JOIN Discussions D ON S.Sid=D.Sid INNER JOIN Professors P ON P.Pid=D.Pid	
LEFT OUTER JOIN (INNER JOIN) + the students that haven't interact with the professors	SELECT S.Name, S.Surname, S.Email, P.FirstName, P.LastName FROM Students S LEFT JOIN Discussions D ON S.Sid=D.Sid LEFT JOIN Professors P ON P.Pid=D.Pid	
RIGHT OUTER JOIN (INNER JOIN) + the professors that haven't interact with the students	SELECT S.Name, S.Surname, S.Email, P.FirstName, P.LastName FROM Students S RIGHT JOIN Discussions D ON S.Sid=D.Sid RIGHT JOIN Professors P ON P.Pid=D.Pid	
FULL OUTER JOIN (LEFT OUTER JOIN + RIGHT OUTER JOIN) (INNER JOIN +the students that haven't interact with the professors and the professors that haven't interact with the students	SELECT S.Name, S.Surname, S.Email, P.FirstName, P.LastName FROM Students S FULL JOIN Discussions D ON S.Sid=D.Sid FULL JOIN Professors P ON P.Pid=D.Pid	

	Name	Surname	Email	FirstName	LastName
1	Mihnea	Dan	dan@uc.ro	Dan	Kiraly
2	Mihnea	Dan	dan@uc.ro	Voina	Claudia
3	Mailat	Mihaela	mm@ucs.ro	Dan	Kiraly
4	Mailat	Mihaela	mm@ucs.ro	Maier	Clementina
5	Roman	Paul	paul@ucs.ro	Maier	Clementina

	Name	Surname	Email	FirstName	LastName
1	Mihnea	Dan	dan@uc.ro	Dan	Kiraly
2	Mihnea	Dan	dan@uc.ro	Voina	Claudia
3	Mailat	Mihaela	mm@ucs.ro	Dan	Kiraly
4	Mailat	Mihaela	mm@ucs.ro	Maier	Clementina
5	Roman	Paul	paul@ucs.ro	Maier	Clementina
6	Romaniuc	Loredana	lore@us.ro	NULL	NULL
7	Cristea	Mihai	mcristea@uc.ro	NULL	NULL
8	Pitic	Mirela	pmirela@ucs.ro	NULL	NULL

	Name	Surname	Email	FirstName	LastName
1	Mihnea	Dan	dan@uc.ro	Dan	Kiraly
2	Mailat	Mihaela	mm@ucs.ro	Dan	Kiraly
3	NULL	NULL	NULL	Popescu	Raluca
4	Mihnea	Dan	dan@uc.ro	Voina	Claudia
5	Mailat	Mihaela	mm@ucs.ro	Maier	Clementina
6	Roman	Paul	paul@ucs.ro	Maier	Clementina

	Name	Surname	Email	FirstName	LastName
1	Mihnea	Dan	dan@uc.ro	Dan	Kiraly
2	Mihnea	Dan	dan@uc.ro	Voina	Claudia
3	Mailat	Mihaela	mm@ucs.ro	Dan	Kiraly
4	Mailat	Mihaela	mm@ucs.ro	Maier	Clementina
5	Roman	Paul	paul@ucs.ro	Maier	Clementina
6	Romaniuc	Loredana	lore@us.ro	NULL	NULL
7	Cristea	Mihai	mcristea@uc.ro	NULL	NULL
8	Pitic	Mirela	pmirela@ucs.ro	NULL	NULL
9	NULL	NULL	NULL	Popescu	Raluca

The SELECT statement

Aggregation operators

- Perform a calculus on a set of values and return only one value
  - COUNT(*)
  - COUNT([DISTINCT] A)
  - SUM([DISTINCT] A)
  - AVG([DISTINCT] A)
  - MAX(A)
  - MIN(A)
  - where A is an attribute name in the table
- Are evaluated on a set of values and correspond to a group of records
- Besides *COUNT*, all the aggregation functions ignore the value NULL
- Usually are used with the clauses *GROUP BY* and *HAVING*

The SELECT statement

Aggregation operators - examples

- Display the number of students.

```
SELECT COUNT(*) FROM Students
```

- Display the minimum and the maximum number of seats from the classes at the floor 1 and 3.

```
SELECT Min_Seats=MIN(NoOfSeats), Max_Seats=MAX(NoOfSeats)
FROM Classes WHERE Floor IN (1, 3)
```

- Display the classes and their floors that have the highest number of seats.

```
SELECT C.Cid, C.Floor
FROM Classes C
WHERE C.NoOfSeats=ANY(SELECT MAX(NoOfSeats) FROM Classes C1)
```

- Display the number of groups that have at least one students called Roman. (without DISTINCT - if in a group there are 2 students called Roman, both of them are going to be counted)

```
SELECT COUNT(DISTINCT S.Gid)
FROM Students S
WHERE S.Name='Roman'
```

The SELECT statement

GROUP BY, HAVING

- ***GROUP BY***
  - Used to group the data from one or multiple columns
  - Usually the aggregation functions are using the *GROUP BY* clause
  - Each group is represented in the final result set by only one row
  - If a query has a *GROUP BY* clause, the entire query will be executed on groups (including *SELECT, HAVING, ORDER BY)*
  - The columns that **don't appear** in the **GROUP BY** clause, **cannot appear** in the command **SELECT**, if it is not in an aggregation function (e.g. MIN, MAX, AVG, SUM, COUNT).

SELECT column_name_1, aggregate_function(column_name_2) …
FROM table_name
WHERE condition
GROUP BY column_name_1

The SELECT statement

Students table

	Sid	Name	Surname	Dob	Email	Gid
1	1	Mihnea	Dan	2000-03-09	dan@uc.ro	822
2	2	Mailat	Mihaela	2001-11-08	mm@ucs.ro	822
3	3	Roman	Paul	2001-01-10	paul@ucs.ro	221
4	4	Romaniuc	Loredana	2000-05-11	lore@us.ro	221
5	5	Cristea	Mihai	2001-11-12	mcristea@uc.ro	221
6	6	Pitic	Mirela	2000-07-05	pmirela@ucs.ro	924

GROUP BY examples:

- Display the number of enrolled students from each group.

```
SELECT S.Gid, COUNT(*) as NoOfStudents
FROM Students S
GROUP BY S.Gid
```

- Display the number of grades and their average for each *4* credits course.

```
SELECT C.CoId, COUNT(*) as [No of grades], AVG(Grade) as average_grade
FROM Courses C, Exams E
WHERE C.CoId=E.CoId AND C.NoOfCredits=4
GROUP BY C.CoId
```

- Display for each student the sum and the average duration of the discussions with the professors that have the first name containing *a*.

```
SELECT S.Sid, S.Name, SUM(Duration) as TotalDuration, AVG(Duration) as AverageDuration
FROM Students S INNER JOIN Discussions D ON S.Sid=D.Sid
INNER JOIN Professors P ON P.Pid=D.Pid
WHERE P.FirstName LIKE '%a%'
GROUP BY S.Sid, S.Name
```

The SELECT statement

GROUP BY, HAVING

- ***HAVING***
  - Used to filter / sort the result set groups after executing the **GROUP BY** clause
  - Only the groups, for which the specified expression from the **HAVING** clause is *TRUE,* are returned
  - The **HAVING** clause is performed only after the records have been grouped and can be used with aggregation functions

SELECT column_name_1, aggregate_function(column_name_2) …
FROM table_name
WHERE condition
GROUP BY column_name_1
HAVING [condition1] aggregate_function(column_name_2) operator value

The SELECT statement

GROUP BY, HAVING examples

- Display the number of enrolled students from each group, if there are at least 3 students per groups.

```
SELECT S.Gid, COUNT(*) as NoOfStudents FROM Students S
GROUP BY S.Gid
HAVING COUNT(*)>=3
```

- Display the average grades greater than 7 for each 5 credits course.

```
SELECT C.CoId, AVG(Grade) as average_grade
FROM Courses C, Exams E
WHERE C.CoId=E.CoId AND C.NoOfCredits=5
GROUP BY C.CoId
HAVING AVG(Grade)>7
```

- Display for each student the sum and the average duration of the discussions with the professors that have the salary greater than 1200; the sum should be between 40 and 60 and the average less than 55.

```
SELECT S.Sid, S.Name, SUM(Duration) as TotalDuration, AVG(Duration) as AverageDuration
FROM Students S INNER JOIN Discussions D ON S.Sid=D.Sid
INNER JOIN Professors P ON P.Pid=D.Pid
WHERE P.Salary>1200
GROUP BY S.Sid, S.Name
HAVING SUM(Duration) BETWEEN 40 AND 60 AND AVG(Duration)<55
```

The SELECT statement

ORDER BY

- Used to filter / sort the result set of the query
- **ASC** – the default (automatically performed) – ascending / alphabetically
- **DESC** – has to be specified – descending / non alphabetically
- Can be used with any of the clauses: *WHERE, GROUP BY / HAVING*

SELECT column_name(s)
FROM table_name
WHERE condition
ORDER BY column_name1 [ASC | DESC], …

The SELECT statement

ORDER BY examples:

- Display the students that have in their surnames *a*, ordered by name.

```
SELECT * FROM Students
WHERE Surname LIKE '%a%'
ORDER BY Name
```

- Display the students that have in their surname *a*, ordered by group id descending and correspondingly, ascending by name.

```
SELECT Gid, Name, Surname FROM Students
WHERE Surname LIKE '%a%'
ORDER BY Gid DESC, Name
```

TOP n – display the first *n* records

- Display the first 2 students from the group 221.

```
SELECT TOP 2 * FROM Students
WHERE Gid=221
```