ICT & Business Intelligence & CRM
SQL

Andrea Vandin
Scuola Superiore Sant'Anna
Paolo Ferragina
Scuola Superiore Sant'Anna
Anna Monreale
Università di Pisa

Reference: Chapter 4 of Databases Essentials, Antonio Albano

Calendar Alert

- You Prof. Monreale asked to move her class of
  - Of 26/03/2025 09:00-11:00
  - To 31/03/2025 09:00-11:00

- You should have been notified by a colleague of yours
  - Agreed

Calendar Alert

- You asked to move
  - Of 24/03/2025 09:00-11:00
  - To 19/03/2025 11:00-13:00
    - *By Prof Monreale*
    - *Recall that you already have an ICT class 19/03/2025 9:00-11:00*
- Ok.

Introduction

- Standard language to define and query relational DB, 1973
  - We can refer in general to it as SQL,
  - But it consists of some sub-languages depending on its use
- **SQL (Structured Query Language)**
  - Interaction with DBMS: **to make queries**
- **DDL (Data Definition Language)**
  - Creation of DB Schema: **to create/delete tables**
- **DCL (Data Control Language)**
  - Control of users authorization
- **DML (Data Manipulation Language)**
  - Instance manipulation: **to add/remove rows**

Create and Delete a DB

- **DDL Instruction**
  - Defining the DB
- **Syntax**
  - CREATE DATABASE <name>;
  - DROP DATABASE <name>;

- **Example**
  - `CREATE DATABASE university;`
  - `DROP DATABASE university;`

Create and Delete a DB

- **Semantics**
  - CREATE DATABASE
    - Create a new empty database
    - Need user's authorization
    - The user becomes owner of the DB
  - DROP DATABASE
    - Delete the DB even if it is not empty!
    - The user must have the authorization

Create and Delete a Table

- **DDL Instruction**
  - CREATE TABLE
    - Defines the schema of a table and creates an empty istance
    - One has to specify attributes, domains and constraints
  - DROP TABLE
    - Delete a table
- **Syntax**
  - `CREATE TABLE <name> (<schema>);`
  - `DROP TABLE <name>;`

Example: Table Teachers

```
CREATE TABLE Teachers (
   code CHAR(4),
   name VARCHAR(20) NOT NULL,
   surname VARCHAR(20) NOT NULL,
   role CHAR(15),
   department CHAR(10),
   PRIMARY KEY(code)
   );

DROP TABLE Teachers;
```

Example: Table Exams

```
CREATE TABLE Exams(
  student integer
      REFERENCES Students(studentID)
      ON DELETE cascade
      ON UPDATE cascade,
  course CHAR(3) REFERENCES Courses(cod),
  mark integer,
  laud bool,
  CHECK (mark>=18 and mark<=30),
  CHECK (not laud or mark=30),
  PRIMARY KEY (student, course)
  );
```

Create and delete tables

- **<schema>**
  - One or more attribute definitions
  - Zero or more constraints definition
- **Attribute Definition**
  - `<name attribute> <type> [<constraints on col>]`
  - Constraints on a single attribute
- **Constraint definition**
  - Constraints related to different attributes

Example: Table Exams

```
CREATE TABLE Exams(
   student integer
       REFERENCES Students(studentID)
       ON DELETE cascade
       ON UPDATE cascade,
   course CHAR(3) REFERENCES Courses(cod),
   mark integer,
   laud bool,
   CHECK (mark>=18 and mark<=30),
   CHECK (not laud or mark=30),
   PRIMARY KEY (student, course)
   );
```

Attribute definition

Constraint definition

Example: Table Exams

Attribute name

Attribute type (domain)

```
CREATE TABLE Exams(
    student integer
        REFERENCES Students(StudentID)
        ON DELETE cascade
        ON UPDATE cascade,
    course CHAR(3) REFERENCES Courses(cod),
    mark integer,
    laud bool,
    CHECK (mark>=18 and mark<=30),
    CHECK (not laud or mark=30),
    PRIMARY KEY (student, course)
    );
```

Attribute Definition

<attribute name>

- Identifier

<type>

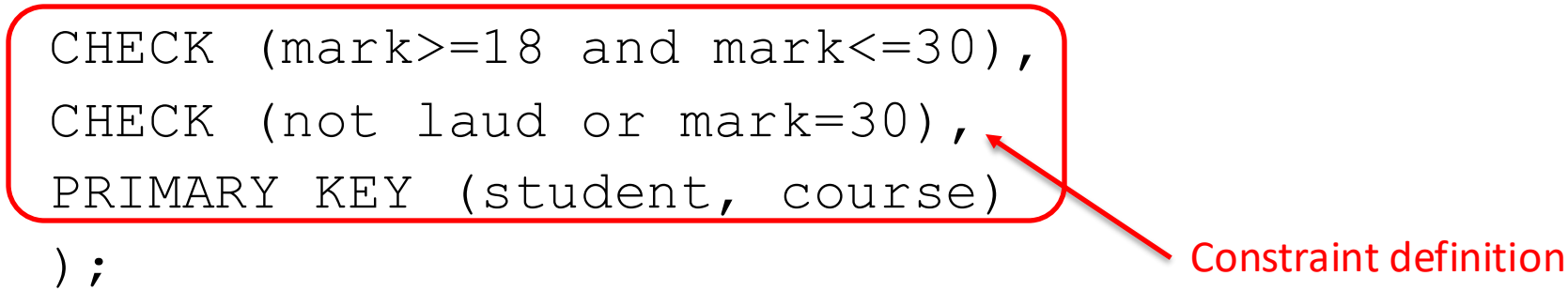
- **Numerical Types**
  - INTEGER,
  - SMALLINT
  - DECIMAL(lung,dec)
  - NUMERIC
  - REAL
  - FLOAT
  - DOUBLE PRECISION

Attribute Definition

- **Single characters or strings**:
  - CHAR(n)
  - VARCHAR(n)
  - LONG VARCHAR, TEXT
- **Time:**
  - DATE
  - TIME
  - TIMESTAMP
- **FLAGs, that specify if an object has a property or not**
  - BINARY(n), BIT(n)
  - VARBINARY(n), VARBIT(n)
  - LONG VARBINARY, BLOB
  - BOOLEAN

Example: Table Exams

```
CREATE TABLE Exams(
  student integer
      REFERENCES Students(StudentID)
      ON DELETE cascade
      ON UPDATE cascade,
  course CHAR(3) REFERENCES Courses(cod),
  mark integer,
  laud bool,
  CHECK (mark>=18 and mark<=30),
  CHECK (not laud or mark=30),
  PRIMARY KEY (student, course)
  );
```



Constraints

- PRIMARY KEY after the type
  - Only one per table
  - Implies NOT NULL
- PRIMARY KEY(<attribute list>) after the end of the attribute definition
  - in case it is composed of more than one attribute
- UNIQUE: same value cannot appear twice in a column
  - Defines a key (not a primary one)
  - After the type
- UNIQUE (<attribute list>)
  - in case it is composed of more than one attribute
- NOT NULL
- FOREIGN KEY (<attribute list>) REFERENCES <external key>
  - [ON update CASCADE]
  - [ON delete CASCADE]
- CHECK (<expression>)

Example: Primary Key

```
CREATE TABLE Teachers (
   code CHAR(4) PRIMARY KEY,
   name VARCHAR(20) NOT NULL,
   surname VARCHAR(20) NOT NULL,
   role CHAR(15),
   department CHAR(10)
   );

CREATE TABLE Exams(
   student integer,
   course CHAR(3),
   mark integer,
   laud bool,
   CHECK (mark>=18 and mark<=30),
   CHECK (not laud or mark=30),
   PRIMARY KEY (student, course),
   FOREIGN KEY (course) REFERENCES Courses(code),
   FOREIGN KEY (student) REFERENCES Students(StudentID)
   );
```

Example: UNIQUE and NOT NULL

```
CREATE TABLE Teachers (
   code CHAR(4) PRIMARY KEY,
   name VARCHAR(20) NOT NULL,
   surname VARCHAR(20) NOT NULL,
   role CHAR(15),
   department CHAR(10),
   UNIQUE (surname,name)
   );

CREATE TABLE Exams(
   student integer,
   course CHAR(3),
   mark integer,
   laud bool,
   CHECK (mark>=18 and mark<=30),
   CHECK (not laud or mark=30),
   PRIMARY KEY (student, course),
   FOREIGN KEY (course) REFERENCES Courses(code),
   FOREIGN KEY (student) REFERENCES
   Students(StudentID)
   );
```

This is not the same thing!

```
name    VARCHAR(20) NOT NULL UNIQUE,
surname VARCHAR(20) NOT NULL UNIQUE,
```

Example: FOREIGN KEY

```
CREATE TABLE Teachers (
   code CHAR(4) PRIMARY KEY,
   name VARCHAR(20) NOT NULL,
   surname VARCHAR(20) NOT NULL,
   role CHAR(15),
   department CHAR(10),
   UNIQUE (Surname,Name)
   );

CREATE TABLE Exams(
   student integer,
   course CHAR(3),
   mark integer,
   laud bool,
   CHECK (mark>=18 and mark<=30),
   CHECK (not laud or mark=30),
   PRIMARY KEY (student, course),
   FOREIGN KEY (course) REFERENCES Courses(code)
               ON DELETE cascade
               ON UPDATE cascade,
   FOREIGN KEY (student) REFERENCES Students(studentID)
               ON DELETE cascade
               ON UPDATE cascade);
```

It means: this attribute refers to a primary key of another table

- This constraint prevents to add invalid data: it has to be a value present in the other table
- What happens if a referred row gets
  - modified or
  - deleted?

Cascade

We can decide how to handle changes in referred keys

- **`ON UPDATE CASCADE:`**
  - If a referred key gets changed in the original table, the change applies to all values referring it
- **`ON DELETE CASCADE:`**
  - if a referred row gets deleted, all rows referring that it get deleted
- **`DEFAULT:`**
  - no update and cancellation is allowed if there are rows in other tables referring that value
- **Example**: update the **studentID** in table **Students**.

ON UPDATE CASCADE

STUDENTS

StudentID	Surname	Name	DoB
888888	Rossi	Maria	25/11/1991
485745	Neri	Anna	23/04/1992
200768	Verdi	Fabio	12/02/1992
587614	Rossi	Luca	10/10/1991
937653	Bruni	Mario	01/12/1991

EXAMS

Student	Mark	Laud	Course
888888	28	0	01
888888	27	0	04
937653	25	0	01
200768	30	1	04

ON DELETE CASCADE

students

StudentID	Surname	Name	DoB
~~276545~~	~~Rossi~~	~~Maria~~	~~25/11/1991~~
485745	Neri	Anna	23/04/1992
200768	Verdi	Fabio	12/02/1992
587614	Rossi	Luca	10/10/1991
937653	Bruni	Mario	01/12/1991

EXAMS

Student	Mark	Laud	Course
~~276545~~	~~28~~	~~0~~	~~01~~
~~276545~~	~~27~~	~~0~~	~~04~~
937653	25	0	01
200768	30	1	04

INSERT

- **DML Instruction**
  - INSERT

- **Syntax**
  - `INSERT INTO <table> (<attributes>) VALUES (<values>);`

- **Semantics**
  - Insert a tuple (a record, a row) into a table

Examples of INSERT

```
INSERT INTO Teachers (code, surname, name, role, department)
VALUES ('FT', 'Pedreschi', 'Dino', 'full professor', 'Computer Science');

INSERT INTO Students(StudentID, surname, name, Program, year, supervisor)
VALUES (111, 'Rossi', 'Mario', 'bachelor', 3, NULL);

INSERT INTO Students(StudentID, surname, name, Program, year)
VALUES (111, 'Rossi', 'Mario', 'bachelor', 3);

INSERT INTO Courses(code, title, Program, teacher)
VALUES ('PR1', 'Programmazione1', 'bachelor', 'FT');
```

Queries

- **SQL Instruction**
  - One or more sub-queries correlated by operators on SETS

- **Sub-query**
  - Selection of some records
  - Duplicates eliminations (DISTINCT)
  - Rename attributes
  - Order of the final results (ORDER BY)
  - Projection (with aggregate functions)

Quering a DB

- **Core of the SELECT Instruction**
  - SELECT: projection, rename, distinct
  - FROM: cartesian product or join, alias
  - [WHERE]: selection
- **Other components**
  - [ORDER BY]

Quering a DB

```
SELECT attribute list
FROM tables list
[WHERE conditions]
```

- Select
  - from the tables listed in `FROM`
  - only the rows satisfying the conditions expressed in `WHERE`
  - extracting only the attributes listed in `SELECT`

A quick recap on Boolean expressions

A	NOT A
True	
False	

A	B	A AND B
True	True	
True	False	
False	True	
False	False	

A	B	A OR B
True	True	
True	False	
False	True	
False	False	

SQL- NULL Values

- SQL uses a logic **with three values** for evaluating the truth value of a condition

  True (T), False (F), Unknown (NULL)

- A simple predicate evaluated on an attribute with value NULL is evaluated as NULL
- **A tuple with truth value NULL is not returned by the query**
- If a predicate of a constraint has NULL value the constraint is not violated

Logic with three values

A	NOT
True	False
False	True
NULL	NULL

A	B	A AND B
True	True	True
True	False	False
True	NULL	NULL
False	True	False
False	False	False
False	NULL	False
NULL	True	NULL
NULL	False	False
NULL	NULL	NULL

A	B	A OR B
True	True	True
True	False	True
True	NULL	True
False	True	True
False	False	False
False	NULL	NULL
NULL	True	True
NULL	False	NULL
NULL	NULL	NULL

Table Students

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

Example Query

– List of name and surname of students of the bachelor degree

```
SELECT S.Surname, S.Name
FROM Students S
WHERE S.program='bachelor'
```

Surname	Name
Rossi	Maria
Neri	Anna
Verdi	Fabio

FROM

- Lists the table or set of tables on which performing the query

Simple case: only one table

- `FROM Students`

or

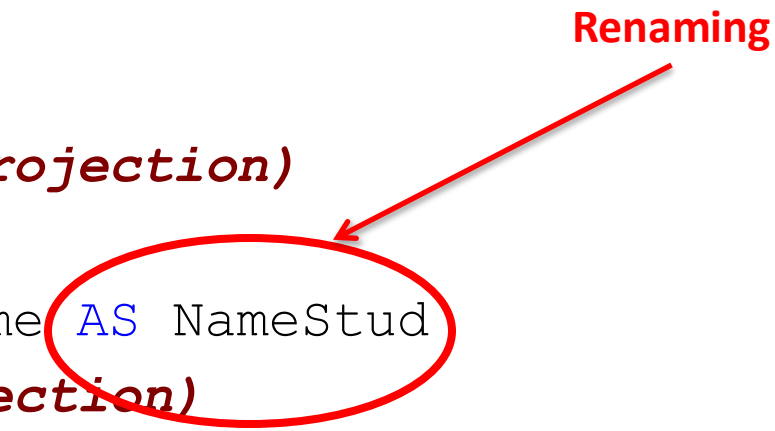
- `FROM Students S`

Projection: SELECT

- **Extract some columns from the table (Projection)**
- **SELECT [DISTINCT] <attributes> | ***
  - <attributes>
    - List of names of attributes
    - Use AS for renaming

Example

```
    SELECT *      (without projection)
OR
    SELECT S.Surname, S.name AS NameStud
                  (with projection)
```



Projection: SELECT

- **Schema of the result**
  - attributes of the original schema
- **Instance of the result**
  - Restriction of tuples to the specified attributes
- **Attention**
  - If the results do not contain keys, then you might get duplicates

Example of Projection

- List the surname degli students

```
SELECT Surname
FROM Students
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

Surname
Rossi
Neri
Verdi
Rossi
Bruni

Selection: `WHERE`

- **Selection of the tuples satisfying a specific condition**
- **`WHERE <condition>`**
- **<condition>**
  - Condition of selection, Boolean connectors

Example

```
WHERE Surname='Rossi' AND Year>1
```

Example of Selection

- Extract information about students with surname Rossi

```
SELECT *
FROM Students
WHERE Surname='Rossi'
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

StudentID	Surname	Name	Program	Year	Supervisor
587614	Rossi	Luca	master	2	FT
276545	Rossi	Maria	bachelor	1	null

Example of Selection

- Extract surname of students with surname Rossi and Year > 1
-

```
SELECT *
FROM Students
WHERE Surname='Rossi' AND Year>1
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

StudentID	Surname	Name	Program	Year	Supervisor
587614	Rossi	Luca	master	2	FT

Complex Condition

- Extract information about students of the bachelor degree at the first or third year

```
SELECT *
FROM Students
WHERE Program = 'bachelor' AND (Year = 1 OR Year = 3)
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
200768	Verdi	Fabio	bachelor	3	FT

Condition“LIKE”

List people having a name starting with ‘M’ and containing ‘r' as third character

```
SELECT *
FROM Students
WHERE name LIKE 'M_r%'
```

1 2 3*rest*

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
937653	Bruni	Mario	master	1	CV

Manage NULL Values

– Students without a Supervisor

```
SELECT *
FROM Students
WHERE Supervisor is NULL
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null

DISTINCT

- The result may contain duplicates
- **Example**
  - List the “Program” in the table students

```
SELECT Program
FROM Students
```

Program
bachelor
bachelor
bachelor
master
master

```
SELECT DISTINCT Program
FROM Students
```

Program
bachelor
master

ORDER BY

- Sort tuples in the result

- **ORDER BY <attributes>**
  - **<attributes>**
    - List of attributes
    - <attribute> {ASC | DESC}

Example

```
ORDER BY Surname ASC, Name DESC
ORDER BY Surname DESC, Name DESC
ORDER BY Surname, Name
```

(default ASC)

- **Semantics**
  - Sorting tuples

Example: ORDER BY

- Extract information about students with surname Rossi

```
SELECT *
FROM Students
WHERE S.Surname='Rossi'
ORDER BY Surname, Name DESC
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
587614	Rossi	Luca	master	2	FT

FROM with JOIN

- In the relational data model, data are split in different tables
- During the queries we need **to correlate data from different tables**
- It is possible to compute the **cartesian product**
- To express JOIN operations on the result of cartesian product, we apply the conditions in WHERE indicating the link between tables

Example of JOIN

STUDENTS

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

TEACHERS

Code	Surname	Name	Role	Department
FT	Monreale	Anna	Reseacher	Computer Science
CV	Tesconi	Maurizio	Reseacher	Engineering

StudentID	Surname	Name	Program	Year	Supervisor	Code	Surname	Name	Role	Departmentd
276545	Rossi	Maria	bachelor	1	null	FT	Monreale	Anna	Engineering	Computer Science
276545	Rossi	Maria	bachelor	1	null	CV	Tesconi	Maurizio	Reseacher	Engineering
485745	Neri	Anna	bachelor	2	null	FT	Monreale	Anna	Reseacher	Computer Science
485745	Neri	Anna	bachelor	2	null	CV	Tesconi	Maurizio	Reseacher	Engineering
200768	**Verdi**	**Fabio**	**bachelor**	**3**	**FT**	**FT**	**Monreale**	**Anna**	**Reseacher**	**Computer Science**
200768	Verdi	Fabio	bachelor	3	FT	CV	Tesconi	Maurizio	Reseacher	Engineering
587614	**Rossi**	**Luca**	**master**	**2**	**FT**	**FT**	**Monreale**	**Anna**	**Reseacher**	**Computer Science**
587614	Rossi	Luca	master	2	FT	CV	Tesconi	Maurizio	Reseacher	Engineering
937653	Bruni	Mario	master	1	CV	FT	Monreale	Anna	Reseacher	Computer Science
937653	**Bruni**	**Mario**	**master**	**1**	**CV**	**CV**	**Tesconi**	**Maurizio**	**Reseacher**	**Engineering**

Example of JOIN

STUDENTS

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

TEACHERS

Code	Surname	Name	Role	Department
FT	Monreale	Anna	Reseacher	Computer Science
CV	Tesconi	Maurizio	Reseacher	Engineering

StudentID	Surname	Name	Program	Year	Supervisor	Code	Surname	Name	Role	Department
276545	Rossi	Maria	bachelor	1	null	FT	Monreale	Anna	Reseacher	Computer Science
276545	Rossi	Maria	bachelor	1	null	CV	Tesconi	Maurizio	Reseacher	Engineering
485745	Neri	Anna	bachelor	2	null	FT	Monreale	Anna	Reseacher	Computer Science
485745	Neri	Anna	bachelor	2	null	CV	Tesconi	Maurizio	Reseacher	Engineering
200768	**Verdi**	**Fabio**	**bachelor**	**3**	**FT**	**FT**	**Monreale**	**Anna**	**Reseacher**	**Computer Science**
200768	Verdi	Fabio	bachelor	3	FT	CV	Tesconi	Maurizio	Reseacher	Engineering
587614	**Rossi**	**Luca**	**master**	**2**	**FT**	**FT**	**Monreale**	**Anna**	**Reseacher**	**Computer Science**
587614	Rossi	Luca	master	2	FT	CV	Tesconi	Maurizio	Reseacher	Engineering
937653	Bruni	Mario	master	1	CV	FT	Monreale	Anna	Reseacher	Computer Science
937653	**Bruni**	**Mario**	**master**	**1**	**CV**	**CV**	**Tesconi**	**Maurizio**	**Reseacher**	**Engineering**

Example of JOIN

STUDENTS

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	2	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

TEACHERS

Code	Surname	Name	Role	Department
FT	Monreale	Anna	Reseacher	Computer Science
CV	Tesconi	Maurizio	Reseacher	Engineering

StudentID	Surname	Name	Program	Year	Supervisor	Code	Surname	Name	Role	Department
200768	Verdi	Fabio	bachelor	3	FT	FT	Monreale	Anna	Reseacher	Computer Science
587614	Rossi	Luca	master	2	FT	FT	Monreale	Anna	Reseacher	Computer Science
937653	Bruni	Mario	master	1	CV	CV	Tesconi	Maurizio	Reseacher	Engineering

FROM

You can do it in two ways:

- **Strategy a**

```
FROM R,S
WHERE S.A=R.B
```

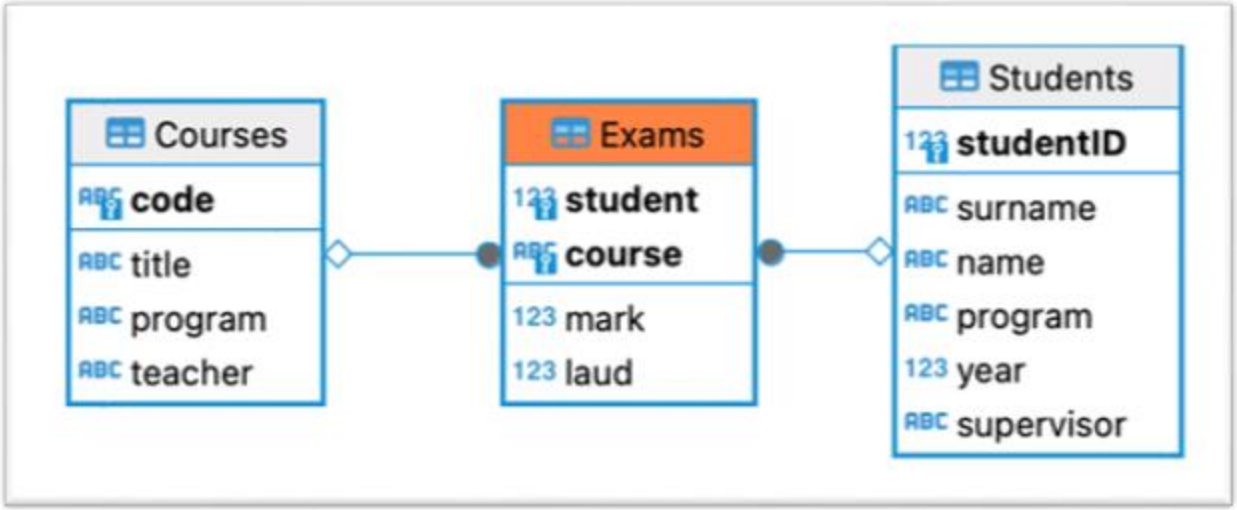
- **Strategy b**

```
FROM S JOIN R ON S.A=R.B
```

Example: JOIN

- List students with a mark greater than 26 in the exam for databases

```
SELECT S.StudentID, S.surname, S.name
FROM Students S JOIN Exams E ON S.StudentID=E.student
          JOIN Courses C ON C.code = E.course
WHERE E.mark > 26 AND C.title = 'databases'
```



Set Operators

- Binary Operators

- `Union`: R U S
- `Intersection`: R ∩ S
- `EXCEPT/MINUS`: R – S

Set Operators

- Tables R and S must have the same number of attributes
- **Positional Association**
  - The list of attributes in the SELECT must have the same type
- **Result Schema**
  - Attribute names of the first table
- **Beware**
  - Elimination of duplicates
  - Otherwise use: UNION ALL, INTERSECT ALL, EXCEPT ALL

Union

Graduated

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialist

StudentID	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Graduated U Specialist

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33

```
SELECT *
FROM Graduated

UNION

SELECT *
FROM Specialist
```

Union

Graduated

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialist

StudentID	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Graduated U Specialist

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33

```
SELECT StudentID, Name, Surname
FROM Graduated

UNION

SELECT StudentID, Surname,Name
FROM Specialist
```

Intersection

Graduated

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialist

StudentID	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Graduated ∩ Specialist

StudentID	Name	Age
7432	Neri	54
9824	Verdi	45

```
SELECT *
FROM Graduated

INTERSECT

SELECT *
FROM Specialist
```

Intersection

Graduated

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialist

StudentID	Name	Age
9297	Neri	trenta
7432	Neri	venti
9824	Verdi	quaranta

```
SELECT *
FROM Graduated
```

Graduated ∩ Specialist

StudentID	Name
7432	Neri
9824	Verdi

```
FROM Specialist
```

```
SELECT StudentID, name
FROM Graduated

INTERSECT

SELECT StudentID, name
FROM Specialist
```

Intersection: IN

Graduated

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialist

StudentID	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Graduated IN Specialist

StudentID	Name	Age
7432	Neri	54
9824	Verdi	45

```
SELECT *
FROM Graduated
WHERE StudentID IN (

SELECT StudentID
FROM Specialist
)
```

Difference

Graduated

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialist

StudentID	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Graduated – Specialist

StudentID	Name	Age
7274	Rossi	42

```
SELECT *
FROM Graduated

EXCEPT

SELECT *
FROM Specialist
```

Difference: NOT IN

Graduated

StudentID	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialist

StudentID	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Graduated– Specialist

StudentID	Name	Age
7274	Rossi	42

```
SELECT *
FROM Graduated
WHERE StudentID NOT IN(
        SELECT StudentID
        FROM Specialist
)
```

Delete ROWS

- **DML Instruction**
  - DELETE
- **Syntax**
  - `DELETE FROM <table> [<WHERE>];`
  - `<WHERE>:` the same `WHERE` as in a Query
- **Semantics**
  - Delete from the table all tuples satisfying the condition in WHERE

Examples: DELETE

```
DELETE FROM Students
   WHERE StudentID=111;

DELETE FROM Students
WHERE Program='bachelor' AND Supervisor= 'FT';
```

UPDATE

- **DML Instruction**
  - UPDATE
- **Syntax**
  - `UPDATE <table> SET <attribute>=<espressione> [<WHERE>]`
- **Semantics**
  - Update the value of an attribute in all tuples satisfying the condition in WHERE

Examples: UPDATE

```
UPDATE Students SET Year=Year+1;

UPDATE Students SET StudentID=11111
   WHERE StudentID=111;

UPDATE Courses SET Teacher='VC'
   WHERE Program='bachelor' AND Teacher='FT';
```

GROUP BY

Andrea Vandin
Scuola Superiore Sant'Anna
Anna Monreale
Università di Pisa

Querying a DB

- **Core of the SELECT Instruction**
  - SELECT: projection, rename, distinct
  - FROM: cartesian product or join, alias
  - [WHERE]: selection
- **Other components**
  - [ORDER BY]
  - [GROUP BY]
  - [HAVING]

Aggregate Queries

- Average mark and total number of exams

```
SELECT AVG(mark), COUNT(*)
FROM Exams
```

- Average mark and total number of exams of student with StudentID = 1000

```
SELECT AVG(mark), COUNT(*)
FROM Exams
WHERE student = 1000
```

Aggregate Function

- Expressions computing functions starting from a set of tuples
  - Count the number of tuples: `COUNT(*)`
    - Count non-null values in a column: `COUNT(attribute)`
  - Minimum value of an attribute: `MIN(attribute)`
  - Maximum value of an attribute: `MAX(attribute)`
  - Average value of an attribute: `AVG(attribute)`
  - Sum of values of an attribute: `SUM(attribute)`

- Syntax:
  - Function([DISTINCT] *)
  - Function([DISTINCT] Attribute)

What about returning avarage mark and number of exams student by student?

```
SELECT student, AVG(mark), COUNT(*)
FROM Exams
```

UNLEGAL!

Which student?

What we need …

EXAMS

Student	Mark	Laud	Course	
276545	28	0	01	27.5
276545	27	0	04	
937653	25	0	01	25
200768	30	1	04	30
587614	28	0	03	29
587614	30	1	04	

1. Create different groups
2. For each group compute the function

GROUP BY

- **GROUP BY**
  - Grouping operator
- **Syntax**
  - `GROUP BY <Attribute List>`
- **Semantics**
  - Create groups of tuples
  - Each group has the same value for the grouping attributes
    - Groups are those records having same values on the grouping attributes

Example GROUP BY

- Grouping students by Program

```
SELECT Program
FROM Students
GROUP BY Program
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	1	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

Example GROUP BY

- Grouping Students per Program and Year

```
SELECT Program, Year, count(*)
FROM Students
GROUP BY Program, Year
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	1	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

Example GROUP BY

- Grouping Students per StudentID

```
SELECT StudentID
FROM Students
GROUP BY StudentID
```

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	1	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

A useless grouping!
- Each students has a different studentID :D

…. so our query is ….

EXAMS

Student	Mark	Laud	Course
276545	28	0	01
276545	27	0	04
937653	25	0	01
200768	30	1	04
587614	28	0	03

1. Create different groups
2. For each group compute the function

Student	AVG_mark	NExams
276545	27.5	2
200768	30	1
587614	28	1
937653	25	1

```
SELECT Student, AVG(Mark) AS AVG_Mark, COUNT(*) AS NExams
FROM Exams
GROUP BY Student
```

Queries with GROUP BY

- **Grouping Attributes**
  - GROUP BY
- **Projection of grouping attributes**
  - SELECT
- **Aggregate Functions applied to each group**
  - SELECT
- **Conditions on the groups involving aggregate functions**
  - HAVING

Example

- Average mark per course

```
SELECT Course, AVG(Mark) AS AVG_MARK
FROM Exams
GROUP BY Course
```

Course	AVG_Mark
01	26.5
04	28.5
03	28

EXAMS

Student	Mark	Laud	Course
276545	28	0	01
276545	27	0	04
937653	25	0	01
200768	30	1	04
587614	28	0	03

Example COUNT

- Number of Students per Program

```
SELECT Program, COUNT(*)
FROM Students
GROUP BY Program;
```

Program	Count(*)
bachelor	3
master	2

```
SELECT Program, COUNT(*) AS NumStudents
FROM Students
GROUP BY Program;
```

Program	NumStudents
bachelor	3
master	2

Sematics:

- Evaluation of FROM
- Creation of groups with GROUP BY
- Evaluation of SELECT **for each group**
- Each group becomes ONLY ONE tuple in the result

GROUP BY & Constraints in SELECT

- If there is `GROUP BY`, only grouping attributes may appear in `SELECT`

```
SELECT Program, COUNT(*)
FROM Students
GROUP BY Program;
```

CORRECT!

```
SELECT COUNT(*)
FROM Students
GROUP BY Program;
```

CORRECT!

```
SELECT Year, COUNT(*)
FROM Students
GROUP BY Program;
```

WRONG!

Example COUNT

- Distribution per Year of bachelor students

```
SELECT Year, COUNT(*) AS NumStud
FROM Students
WHERE Program='bachelor'
GROUP BY Year;
```

Year	NumStud
1	2
3	1

StudentID	Surname	Name	Program	Year	Supervisor
276545	Rossi	Maria	bachelor	1	null
485745	Neri	Anna	bachelor	1	null
200768	Verdi	Fabio	bachelor	3	FT
587614	Rossi	Luca	master	2	FT
937653	Bruni	Mario	master	1	CV

HAVING

- Expresses conditions on groups
- It is not possible to use `WHERE` on groups
- **Example:**
  - **Avearge Mark for each course with at least 2 exams**

EXAMS

Student	Mark	Laud	Course
276545	28	0	01
276545	27	0	04
937653	25	0	01
200768	30	1	04
587614	28	0	03

```
SELECT Course, AVG(Mark) AS AVG_Mark
FROM Exams
GROUP BY Course
HAVING COUNT(*)>1
```

Course	AVG_Mark
01	26.5
04	28.5

QUERY

```
SELECT [DISTINCT] <result>
FROM <join or cartesian product>
[WHERE <condition on tuples>]
[GROUP BY <grouping attributes>]
[HAVING <conditions on groups>]
[ORDER BY <ordering attributes>]
```

```
CREATE TABLE Teachers (
    code char(4) PRIMARY KEY,
    surnamevarchar(20) NOT NULL,
    name varchar(20) NOT NULL,
    role char(15),
    department char(10)
);

CREATE TABLE Students (
    studentID integer PRIMARY KEY,
    surname varchar(20) NOT NULL,
    name varchar(20) NOT NULL,
    program char(20),
    year integer,
    supervisor char(4)
      REFERENCES Teachers(code)
);


CREATE TABLE Courses (
    code char(3) PRIMARY KEY,
    title varchar(20) NOT NULL,
    program char(20),
    teacher char(4)
      REFERENCES Teachers(code)
);
```

```
CREATE TABLE Tutoring (
    student integer
        REFERENCES
Students(studentID),
    tutor integer
        REFERENCES
Students(studentID),
    PRIMARY KEY (student,tutor)
);

CREATE TABLE Exams (
    student integer
        REFERENCES Students(studentID)
        ON DELETE cascade
        ON UPDATE cascade,
    course char(3)
        REFERENCES Courses(code),
    mark integer,
    laud bool,
    CHECK (mark>=18 and mark<=30),
    CHECK (not laud or mark=30),
    PRIMARY KEY (student, corso)
);

CREATE TABLE Phones(
    teacher char(4)
        REFERENCES Teachers(code),
    phonenumber char(9),
    PRIMARY KEY (teacher,number)
);
```