Basi di dati MOD 1
- SQL -

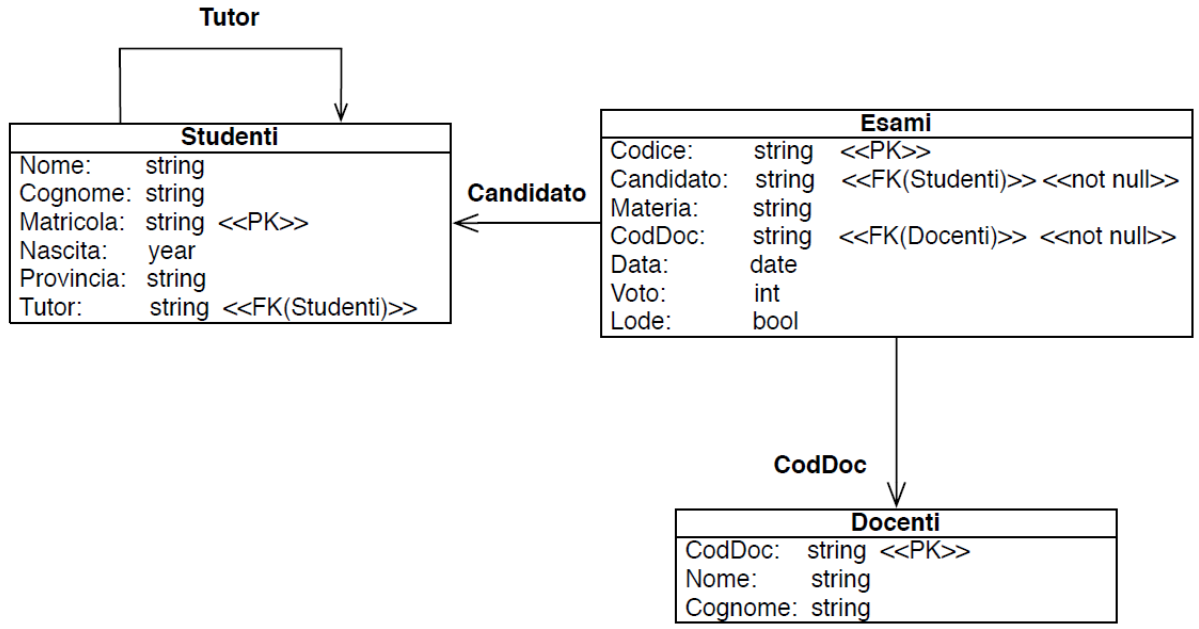
SQL sta per Structured Query Language ed è il più utilizzato nei DBMS.
È un linguaggio dichiarativo che si basa su calcolo relazionale su ennuple e su algebra relazionale, nel quale:

- Le relazioni diventano tabelle
- Le ennuple diventano record/righe
- Gli attributi diventano campi/colonne

Le tabelle possono avere righe duplicate per vari motivi:

- efficienza: eliminarle è costoso (n log(n))
- flessibilità: possono servire i duplicati, ad esempio per calcolare le medie

Con SQL possiamo interrogare il DB ma anche definirlo (creare tabelle, alterare gli schemi, ecc).
D'ora in poi per fare gli esempi ci baseremo su questo schema:



SELECT:
Il comando SELECT è la proiezione
SELECT [**DISTINCT**] Attributi
FROM Tabelle
[**WHERE** Condizione]

La condizione può essere una combinazione booleana (AND, OR, NOT) ma possiamo farci cose ancora più complesse (vedremo più avanti).
Es:
SELECT *
FROM R1, ..., Rn

Il FROM sarebbe: $R_1 \times \ldots \times R_n$

```
SELECT *
FROM R1, ..., Rn
WHERE C
```

il FROM e il WHERE sarebbero: $\sigma_C(R_1 \times \ldots \times R_n)$

```
SELECT DISTINCT A1, ..., An
FROM R1, ..., Rn
WHERE C
```

il DISTINCT toglie i duplicati!

Altri esempi…

```
SELECT *
FROM Studenti;
```

```
SELECT *
FROM Esami
WHERE Voto > 26;
```

```
SELECT DISTINCT Provincia
FROM Studenti;
```

```
SELECT *
FROM Studenti, Esami;
```

Trovare il nome, la matricola e la provincia degli studenti:

```
SELECT Nome, Matricola, Provincia
FROM Studenti
```

Trovare tutti i dati degli studenti di Venezia:

```
SELECT *
FROM Studenti
WHERE Provincia = 'VE‘;
```

Trovare nome, matricola e anno di nascita degli studenti di Venezia:

```
SELECT Nome, Matricola, Nascita
FROM Studenti
WHERE Provincia = 'VE';
```

Prodotto e giunzioni:

Il prodotto e le giunzioni “mettono insieme/combinano” varie relazioni.

Es:

Tutte le possibili coppie (Studente, Esame):

```
SELECT *
FROM Studenti, Esami
```

Tutte le possibili coppie (Studente, Esame sostenuto dallo studente):

```
SELECT *
FROM Studenti, Esami
WHERE Matricola = Candidato
```

Nome e data degli esami per studenti che hanno superato l'esame di BD con 30:

```
SELECT Nome, Data
FROM Studenti, Esami
WHERE Matricola = Candidato AND
Materia='BD' AND Voto=30
```

Se alcune tabelle hanno degli attributi col nome uguale bisogna qualificarli, lo si fa con la "dot notation". Es:
Generare una tabella che riporti Codice, Nome, Cognome dei docenti e Codice degli esami corrispondenti:

```
SELECT Docenti.CodDoc, Docenti.Nome, Docenti.Cognome,
Esami.Codice
FROM Esami, Docenti
WHERE Docenti.CodDoc = Esami.CodDoc
```

Possiamo anche dare un alias (darle un altro nome provvisorio) ad una tabella, cosa essenziale nel caso dobbiamo fare una query ricorsiva! Es:
Generare una tabella che contenga cognomi e matricole degli studenti e dei loro tutor:

```
SELECT s.Cognome, s.Matricola, t.Cognome, t.Matricola
FROM Studenti s, Studenti t
WHERE s.Tutor = t.Matricola
```

Ciò rende anche la query più leggibile!

Con la keyword AS possiamo associare ad una colonna (attributo o espressione) il nome che vogliamo! Es:

```
SELECT Nome, Cognome, YEAR(CURDATE())-Nascita AS Età
FROM Studenti
WHERE Provincia='VE'
```

Le espressioni nel SELECT possono contenere:

- operatori aritmetici (o altre funzioni sugli attributi).
- funzioni di aggregazione

Funzioni di aggregazione:
Hanno le keyword: COUNT(*), SUM, AVG, MIN, MAX

Nel SELECT o sono TUTTE funzioni di aggregazione o NESSUNA! Non può esserci un mix tra attributi e funzioni di aggregazione! Le funzioni di aggregazione NON si possono usare nella clausola WHERE!

Es:
Numero di elementi della relazione Studenti:

```
SELECT COUNT(*)
FROM Studenti
```

Anno di nascita minimo, massimo e medio degli studenti:

```
SELECT MIN(Nascita), MAX(Nascita), AVG(Nascita)
FROM Studenti
```

Questa query è DIVERSA da:

```
SELECT MIN(Nascita), MAX(Nascita), AVG(DISTINCT Nascita)
FROM Studenti
```

Perchè se tolgo i doppioni di Nascita otterrò un valore diverso!

Questa NON si può fare per il motivo di prima:

```
SELECT Candidato, AVG(Voto)
FROM Esami
```

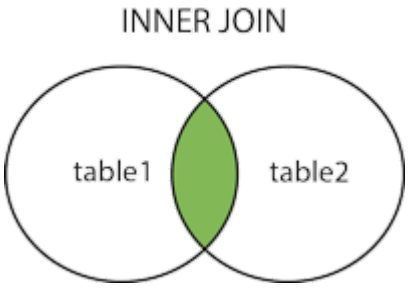
Numero di Studenti che hanno un Tutor:

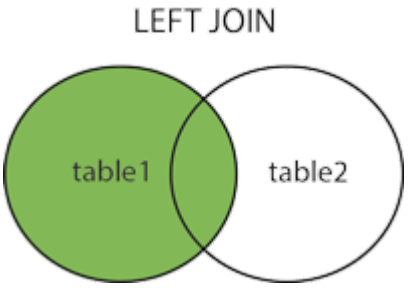
```
SELECT COUNT(Tutor)
FROM Studenti
```

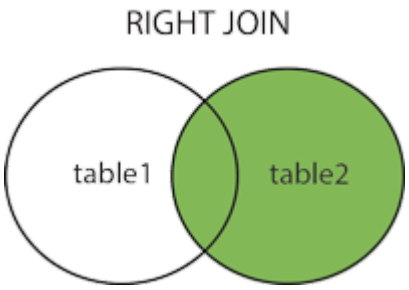
Numero di studenti che fanno i Tutor:

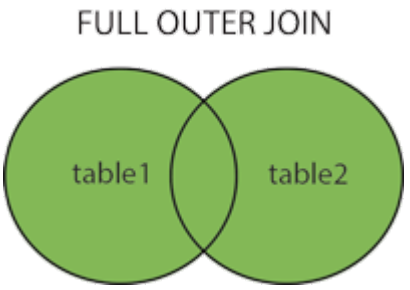
```
SELECT COUNT(DISTINCT Tutor)
FROM Studenti
```

Giunzioni:
Nella clausola FROM oltre al prodotto tra tabelle (T1,T2,...,Tn) possiamo fare anche le giunzioni.
Non fanno altro che combinare le tabelle e le relative tuple basandosi sui parametri della giunzione.
I tipi della giunzione che vengono usate più spesso sono:

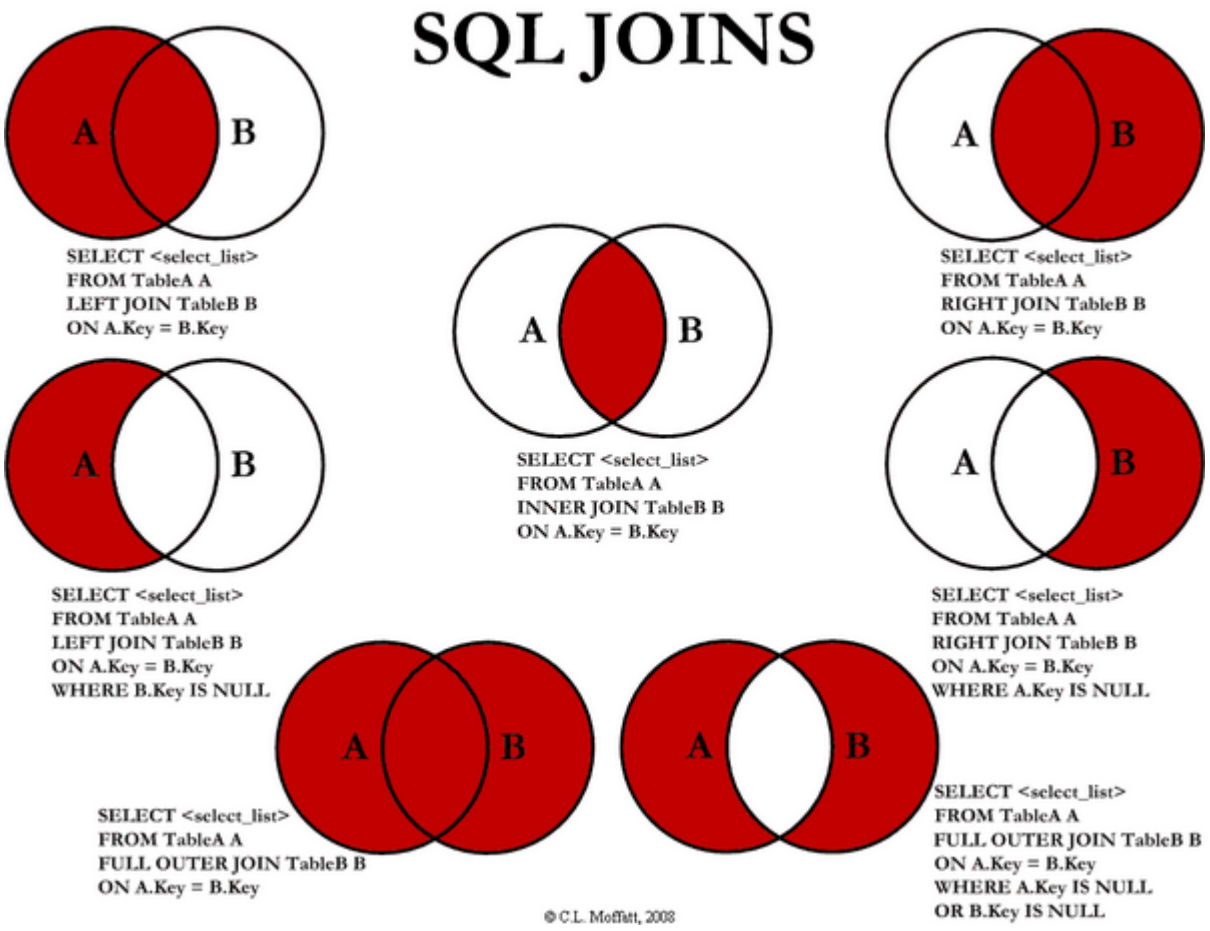








Nel complesso tutte le JOIN sono:



La sintassi per fare le JOIN è:
SELECT Attributi
FROM Tabelle
WHERE Condizione

Con tabelle che ha la seguente sintassi:
Tabella Giunzione Tabella
[USING (Attributi) | ON Condizione]

Giunzione può avere: [CROSS|NATURAL] [LEFT|RIGHT|FULL] JOIN
Come funzionano è scritto sopra nelle immagini.

- CROSS JOIN: realizza il prodotto

- NATURAL JOIN: Realizza la JOIN tra le due tabelle basandosi sulla FK e la PK che collega le due tabelle.
- … JOIN ... ON: Dopo ON c'è la condizione sulla quale viene basata la JOIN
  Es:
  SELECT *
  FROM Studenti s JOIN Studenti t ON s.Tutor = t.Matricola;
- … JOIN … USING Attributi in comune
  Funziona come il NATURAL JOIN, ma solo sugli attributi comuni elencati
  Es:
  SELECT s.Cognome AS CognomeStud,e.Materia,d.Cognome AS CognomeDoc
  FROM Studenti s JOIN Esami e ON s.Matricola = e.Candidato
  JOIN Docenti d USING (CodDoc);

- LEFT, RIGHT, FULL, INNER (JOIN) fanno la rispettiva JOIN esterna, nelle foto prima spiega come funzionano.
  Es:
  SELECT Nome, Cognome, Matricola, Data, Materia
  FROM Studenti s LEFT JOIN Esami e
  ON s.Matricola=e.Candidato;
  Darà come risultato:

Nome	Cognome	Matricola	Data	Materia
Chiara	Scuri	71346	NULL	NULL
Giorgio	Zeri	71347	NULL	NULL
Paolo	Verdi	71523	2006-07-08	BD
Paolo	Verdi	71523	2006-12-28	ALG
Paolo	Poli	71576	2007-07-19	ALG
Paolo	Poli	71576	2007-07-29	FIS
Anna	Rossi	76366	2007-07-18	BD
Anna	Rossi	76366	2007-07-08	FIS

  Compaiono anche studenti con valori NULL perchè non hanno fatto esami!

- ORDER BY attributo:
  La clausola ORDER BY ordina l'elenco di record in output alla query in base all'attributo (o attributi) elencati in ordine crescente (ASC) o decrescente (DESC). Se non si specifica nulla viene impostato di default ad ASC!
  Es:
  SELECT Nome,Cognome
  FROM Studenti
  WHERE Provincia='VE'
  ORDER BY Cognome DESC, Nome DESC

- OPERATORI INSIEMISTICI:
  Servono a combinare i risultati di più query che hanno come risultato colonne di egual nome e tipo. Sono UNION, INTERSECT ed EXCEPT.
  Es:

  Nome, cognome e matricola degli studenti di Venezia e di quelli che hanno preso più di 28 in qualche esame:

```
SELECT Nome, Cognome, Matricola
FROM Studenti
WHERE Provincia='VE'
UNION
SELECT Nome, Cognome, Matricola
FROM Studenti JOIN Esami ON Matricola=Candidato
WHERE Voto>28;
```

  Le matricole degli studenti che non sono tutor:

```
SELECT Matricola
FROM Studenti
EXCEPT
SELECT Tutor AS Matricola
FROM Studenti
```

  Gli operatori insiemistici effettuano la rimozione dei duplicati in automatico, a meno che non sia richiesto che ci siano con l'opzione ALL.
  Es:
  Nome e cognome degli studenti che hanno preso in un esame 18 e in un altro esame 30:

```
SELECT Nome, Cognome, Matricola
FROM Studenti JOIN Esami ON Matricola=Candidato
WHERE Voto = 18
INTERSECT ALL
SELECT Nome, Cognome, Matricola
FROM Studenti JOIN Esami ON Matricola=Candidato
WHERE Voto = 30;
```

- Il valore NULL:
  Il valore NULL è particolare, viene inserito quando il valore ad un attributo non è applicabile, quando l'attributo non è disponibile, ecc.
  Per verificare che un attributo sia NULL usiamo la sintassi:

```
SELECT …
FROM …
WHERE Cond
```

  Cond = Expr IS [NOT] NULL

Solo così si può verificare se un attributo è NULL, non si può usare attributo == NULL!

Es:
Gli studenti che non hanno Tutor

```
SELECT *
FROM Studenti
WHERE Tutor IS NULL
```

- BETWEEN:
  È una condizione che viene posta su un valore numerico che deve essere in un range. Es:

```
SELECT *
FROM Studenti
WHERE Matricola BETWEEN 71000 AND 72000
```

- Pattern matching:
  Viene effettuato sulle stringhe. Vediamo la sintassi per capire subito:

```
WHERE Expr LIKE pattern
```

pattern può contenere caratteri e simboli speciali:
- % sequenza di 0 o più caratteri speciali
- _ un carattere qualsiasi

Es:
Studenti con il nome di almeno due caratteri che inizia per A:

```
SELECT *
FROM Studenti
WHERE Nome LIKE 'A_%'
```

Studenti con il nome che inizia per ‘A’ e termina per ‘a’ oppure ‘i’:

```
SELECT *
FROM Studenti
WHERE Nome LIKE 'A%a' OR Nome LIKE ‘A%i’
```

- WHERE:
  La clausola WHERE è più complessa di come l'abbiamo vista fino ad ora. Può essere una combinazione booleana (AND, OR, NOT) di predicati tra cui:
  - Expr Comp Expr
  - Expr Comp (Sottoselect che torna esattamente un valore)
  - Expr [NOT] IN (Sottoselect) (oppure IN (v1,..,vn))
  - [NOT] EXISTS (Sottoselect)
  - Expr Comp (ANY | ALL) (Sottoselect)

Comp: <, =, >, <>, <=, >= (e altri).

- SELECT annidate:
  Vengono inserite nel campo WHERE assieme ad altre operazioni di confronto, servono in quei casi dove bisogna estrarre dati e confrontarli. Es:
  Studenti che vivono nella stessa provincia dello studente con matricola 71346, escluso lo studente stesso.

```
SELECT   *
FROM     Studenti
WHERE    (Matricola <> '71346') AND
         Provincia = (SELECT Provincia
                      FROM   Studenti
                      WHERE  Matricola='71346')
```

  Anche se la sottoselect non era indispensabile… (vedi pag. 43 slide SQL).
- Quantificazione:
  Ci sono due tipi di quantificazione: Universale ed Esistenziale.
  - Universale: esistenziale negata.Es:
    Non tutti i voti sono =30 (universale) = esiste un voto ≠30 (esistenziale).
  - Esistenziale: universale negata. Es:
    Non esiste un voto diverso da 30 (esistenziale) = Tutti i voti sono uguali a 30 (universale).
- Quantificatore esistenziale EXISTS:
  Es:
  Gli studenti con almeno un voto > 27.
  Usiamo una sottoselect:

```
SELECT ...

FROM   ...

WHERE [NOT] EXISTS (Sottoselect)
```

  Per ogni tupla (o combinazione di tuple) t della select esterna
  - calcola la sottoselect
  - verifica se ritorna una tabella [non] vuota e in questo caso seleziona t

  Che diventerà:

```
SELECT *
FROM   Studenti s
WHERE  EXISTS (SELECT *
               FROM  Esami e
               WHERE e.Candidato = s.Matricola
                     AND e.Voto > 27)
```

Posso ottenere lo stesso risultato anche attraverso una giunzione:

```
SELECT DISTINCT s.*
FROM   Studenti s JOIN Esami e ON e.Candidato = s.Matricola
WHERE  e.Voto > 27
```

- Quantificatore esistenziale ANY:

```
SELECT ...

FROM   ...

WHERE  Expr Comp ANY (Sottoselect)
```

Per ogni tupla (o combinazione di tuple) t della select esterna
- calcola la sottoselect
- verifica se Expr è in relazione Comp con almeno uno degli elementi ritornati dalla select

Es: la query di prima.

```
SELECT *
FROM   Studenti s
WHERE  s.Matricola =ANY (SELECT e.Candidato
                         FROM Esami e
                         WHERE e.Voto >27)
```

ANY non fa nulla in più di EXISTS! Semplicemente lo scrivi in un altra maniera…

- Quantificatore esistenziale IN:

È un'abbreviazione di ANY...

```
SELECT ...

FROM   ...

WHERE Expr IN (sottoselect)
```

Es:
Ancora la query di prima…

```
SELECT *
FROM   Studenti s
WHERE  s.Matricola IN (SELECT e.Candidato
                       FROM   Esami e
                       WHERE  e.Voto >27)
```

Posso anche cercare un valore in un'ennupla con vari valori:

```
SELECT   *
FROM     Studenti
WHERE    Provincia IN ('PD','VE','BL');
```

Quindi la quantificazione esistenziale si fa con: EXISTS, JOIN, ANY, IN.
Ora però non confondiamola con la quantificazione universale!

- Quantificazione universale:
  Es: gli studenti che hanno preso solo 30.

```
SELECT s.*
FROM   Studenti s, Esami e
WHERE  e.Candidato = s.Matricola AND e.Voto = 30
```

Questa query è sbagliata! E se fate questo errore è gravissimo!
La query giusta è:

```
SELECT *  FROM Studenti s
WHERE NOT EXISTS (SELECT *
                  FROM   Esami e
                  WHERE  e.Candidato = s.Matricola
                         AND e.Voto <> 30)
```

dove NOT(e.Voto = 30) e` diventato e.Voto <> 30

Vediamo ora il quantificatore universale ALL:
Stessa query di prima

```
SELECT * FROM Studenti s
WHERE s.Matricola <>ALL (SELECT e.Candidato
                         FROM Esami e
                         WHERE e.Voto <> 30)
```

Vediamo un altro esempio di quantificazione universale:
Prendiamo come esempio questa query:

```
SELECT s.Nome, s.Cognome, e.Materia, e.Voto
FROM   Studenti s LEFT JOIN Esami e ON s.Matricola=e.Candidato;
```

Che da in output la seguente tabella:

Nome	Cognome	Materia	Voto
Chiara	Scuri	NULL	NULL
Giorgio	Zeri	NULL	NULL
Paolo	Verdi	BD	27
Paolo	Verdi	ALG	30
Paolo	Poli	ALG	22
Paolo	Poli	FIS	22
Anna	Rossi	BD	30
Anna	Rossi	FIS	30

La query “studenti che hanno preso solo 30” è:

```
SELECT s.Cognome
FROM   Studenti s
WHERE NOT EXISTS (SELECT *
                  FROM  Esami e
                  WHERE e.Candidato = s.Matricola
                        AND e.Voto <> 30)
```

Con output:

Cognome
Scuri
Zeri
Rossi

La query “gli studenti che hanno preso solo trenta, e hanno superato qualche esame” è:

```
SELECT *
FROM Studenti s
WHERE NOT EXISTS (SELECT *
                  FROM  Esami e
                  WHERE e.Candidato = s.Matricola
                        AND e.Voto <> 30)
      AND EXISTS (SELECT *
                  FROM  Esami e
                  WHERE e.Candidato = s.Matricola)
```

Oppure:

```
SELECT s.Matricola, s.Cognome
FROM   Studenti s JOIN Esami e ON s.Matricola = e.Candidato
GROUP BY s.Matricola, s.Cognome
HAVING MIN(e.Voto) = 30;
```

- Raggruppamento:
  GROUP BY attributo
  Raggruppa i risultati della tabella per i parametri specificati.

```
SELECT   e.Materia, AVG(e.Voto)
FROM     Esami e
GROUP BY e.Materia
```

Esempio di esecuzione del GROUP BY:

```
SELECT Candidato, COUNT(*) AS NEsami,
       MIN(Voto), MAX(Voto), AVG(Voto)

FROM   Esami

GROUP BY Candidato

HAVING AVG(Voto) > 23;
```

- La query ci darà in output la tabella:

Codice	Materia	Candidato	Data	Voto	Lode	CodDoc
B112	BD	71523	2006-07-08	27	N	AM1
B247	ALG	71523	2006-12-28	30	S	NG2
B248	BD	76366	2007-07-18	29	N	AM1
B249	ALG	71576	2007-07-19	22	N	NG2
F313	FIS	76366	2007-07-08	26	N	GL1
F314	FIS	71576	2007-07-29	22	N	GL1

Che verrà divisa nella seguente tabella dal GROUP BY:

Codice	Materia	Candidato	Data	Voto	Lode	CodDoc
B112	**BD**	**71523**	**2006-07-08**	**27**	**N**	**AM1**
B247	**ALG**	**71523**	**2006-12-28**	**30**	**S**	**NG2**
B249	ALG	71576	2007-07-19	22	N	NG2
F314	FIS	71576	2007-07-29	22	N	GL1
B248	**BD**	**76366**	**2007-07-18**	**29**	**N**	**AM1**
F313	**FIS**	**76366**	**2007-07-08**	**26**	**N**	**GL1**

E che verrà “scremata” nella seguente tabella dal HAVING:

Candidato	NEsami	min(Voto)	max(Voto)	avg(Voto)
71523	2	27	30	28.5000
76366	2	26	29	27.5000

Dopo il GROUP BY possiamo usare la clausola HAVING, ma SOLO dopo il GROUP BY!

Es: Per ogni materia, trovare nome della materia e voto medio degli esami in quella materia [selezionare solo le materie per le quali sono stati sostenuti più di 3 esami].

```
SELECT    e.Materia, AVG(e.Voto)
FROM      Esami e
GROUP BY  e.Materia
[HAVING    COUNT(*)>3]
```

La sintassi è:

```
SELECT … FROM … WHERE …
GROUP BY A1,..,An
[HAVING condizione]
```

La semantica è:

- Esegue le clausole FROM - WHERE
- Partiziona la tabella risultante rispetto all'uguaglianza su tutti i campi A1, …, An (in questo caso, si assume NULL = NULL)
- Scarta i gruppi che non rispettano la clausola HAVING
- Da ogni gruppo estrae una riga usando la clausola SELECT

Nelle query con funzioni di aggregazione, nel caso ci siano attributi "normali" nella SELECT ma anche nel HAVING, bisogna includerli dentro al GROUP BY.

Es:

Nella query:

```
SELECT   s.Cognome, AVG(e.Voto)
FROM     Studenti s, Esami e
WHERE    s.Matricola = e.Candidato
GROUP BY s.Matricola
```

Bisognerà scrivere:

```
GROUP BY s.Matricola, s.Cognome
```

La clausola HAVING cita solo:

- espressioni su attributi di raggruppamento
- funzioni di aggregazione applicate ad attributi normali (non di raggruppamento)

Quindi, ad esempio, la seguente query è sbagliata:

```
SELECT  s.Cognome, AVG(e.Voto)
FROM    Studenti s JOIN Esami e ON s.Matricola = e.Candidato
GROUP BY s.Matricola, s.Cognome
HAVING  YEAR(Data) > 2006;
```

Perchè HAVING non ha come argomento una funzione di aggregazione!

- Raggruppamento e NULL:

Nel raggruppamento, stranamente, si assume che NULL = NULL.

Es:

Matricole dei tutor e relativo numero degli studenti di cui sono tutor.

```
SELECT Tutor, COUNT(*) AS NStud
FROM   Studenti
GROUP BY Tutor;
```

Che ci darà come output:

```
+-------+-------+
| Tutor | NStud |
+-------+-------+
| NULL  |     2 |
| 71347 |     2 |
| 71523 |     1 |
+-------+-------+
```

- Sintassi SELECT e sottoselect:
  Vediamole in maniera completa:

```
SELECT [DISTINCT] Attributi
FROM Tabelle
[WHERE Condizione]
[GROUP BY A1,..,An [HAVING Condizione]]
Sottoselect
{ (UNION [ALL] | INTERSECT [ALL] | EXCEPT [ALL])
  Sottoselect }
[ ORDER BY Attributo [DESC] {, Attributo [DESC]} ]
```

- Manipolazione dei dati (DML):
  Inserimento di dati:
  Tutti i dati con vincoli NOT NULL vanno specificati sennò ci da errore!

```
INSERT INTO Tabella  [(A1,..,An)]
( VALUES  (V1,..,Vn) | AS Select )
```

Modifica di dati già esistenti:

```
UPDATE Tabella
SET    Attributo = Expr, …,  Attributo = Expr
WHERE  Condizione
```

Cancellazione di dati:

```
DELETE FROM Tabella

WHERE  Condizione
```

Vediamo alcuni esempi:

```
INSERT INTO Studenti (Matricola, Nome, Cognome)
VALUES (74324,'Gino','Bartali')


DELETE FROM Studenti
WHERE Matricola NOT IN (SELECT Candidato FROM Esami);


UPDATE Studenti
SET    Tutor='71523'
WHERE  Matricola='76366' OR Matricola='76367'
```

- Creazione della struttura del DB (DDL):

Creazione dello schema:

```
CREATE SCHEMA Università AUTHORIZATION rossi
```

Cancellazione di uno schema:

```
DROP SCHEMA Università CASCADE
```

CASCADE cancella tutte le tabelle che contiene "a cascata".

In uno schema ci sono due tipi di tabella:
- Tabelle normali: i dati sono fisicamente memorizzati dentro ad essa.
- Viste: i dati non sono fisicamente memorizzati dentro ad essa ma prodotti a runtime quando viene usata.

Es creazione tabella normale:

```
CREATE TABLE Esami (
        Codice      CHAR(4) PRIMARY KEY,
        Materia     CHAR(3),
        Candidato   CHAR(6) NOT NULL,
        Data        DATE,
        Voto        INTEGER CHECK(Voto >= 18 AND Voto <= 30),
        Lode        CHAR(1),
        CodDoc      CHAR(3) NOT NULL,
        UNIQUE (Materia,Candidato),
        FOREIGN KEY (Candidato) REFERENCES Studenti(Matricola)
                                ON UPDATE CASCADE,

        FOREIGN KEY (CodDoc)    REFERENCES Docenti(CodDoc)
                                ON UPDATE CASCADE
);
```

Es creazione vista:

```
CREATE VIEW VotiMedi(Matricola, Media) AS
        SELECT e.Candidato,AVG(Voto)
        FROM   Esami e
        GROUP BY e.Candidato;
```

Per modificare una tabella usiamo il comando ALTER TABLE:
Aggiungere attributi:

```
ALTER TABLE  Studenti
      ADD COLUMN Nazionalita VARCHAR(10) DEFAULT 'Italiana';
```

Cancellare attributi:

```
ALTER TABLE  Studenti
        DROP COLUMN Provincia;
```

Modificare il tipo di una colonna:

```
ALTER TABLE  Studenti
      ALTER COLUMN Nazionalita TYPE VARCHAR(15);
```

Per eliminare una tabella:

```
DROP TABLE Studenti
```

Se alla fine aggiungo CASCADE elimina tutte le viste che la utilizzano, se aggiungo RESTRICT non cancella la tabella (o la vista) se viene utilizzata in una vista.

Possiamo anche inizializzare una tabella al momento della creazione:

```
CREATE TABLE TutorVE AS
        SELECT  t.Matricola, t.Nome, t.Cognome
        FROM    Studenti t
        WHERE   t.Matricola IN (SELECT s.Tutor
                                FROM   Studenti s
                                WHERE  s.Provincia='VE');
```

Es vista:
Trovare la media dei voti massimi ottenuti nelle varie province.

```
  CREATE VIEW ProvMax(Provincia, Max) AS
      SELECT s.Provincia, MAX(e.Voto)
      FROM Studenti s JOIN Esami e ON s.Matricola = e.Candidato
      GROUP BY s.Provincia;

SELECT AVG(Max) FROM ProvMax;
```

Prima mi ricavo il MAX e poi faccio la media, non posso fare subito la AVG(MAX())!

- CASE:
  Sono molto utili. Sintassi:

```
CASE WHEN condizione THEN risultato
        [WHEN ...]
        [ELSE risultato]
END
```

Es:

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

Es:

```sql
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
      WHEN City IS NULL THEN Country
      ELSE City
END);
```