

Basi di Dati - VI

Corso di Laurea in Informatica
Anno Accademico 2023/2024

Alessandra Raffaetà
raffaeta@unive.it

Il linguaggio SQL

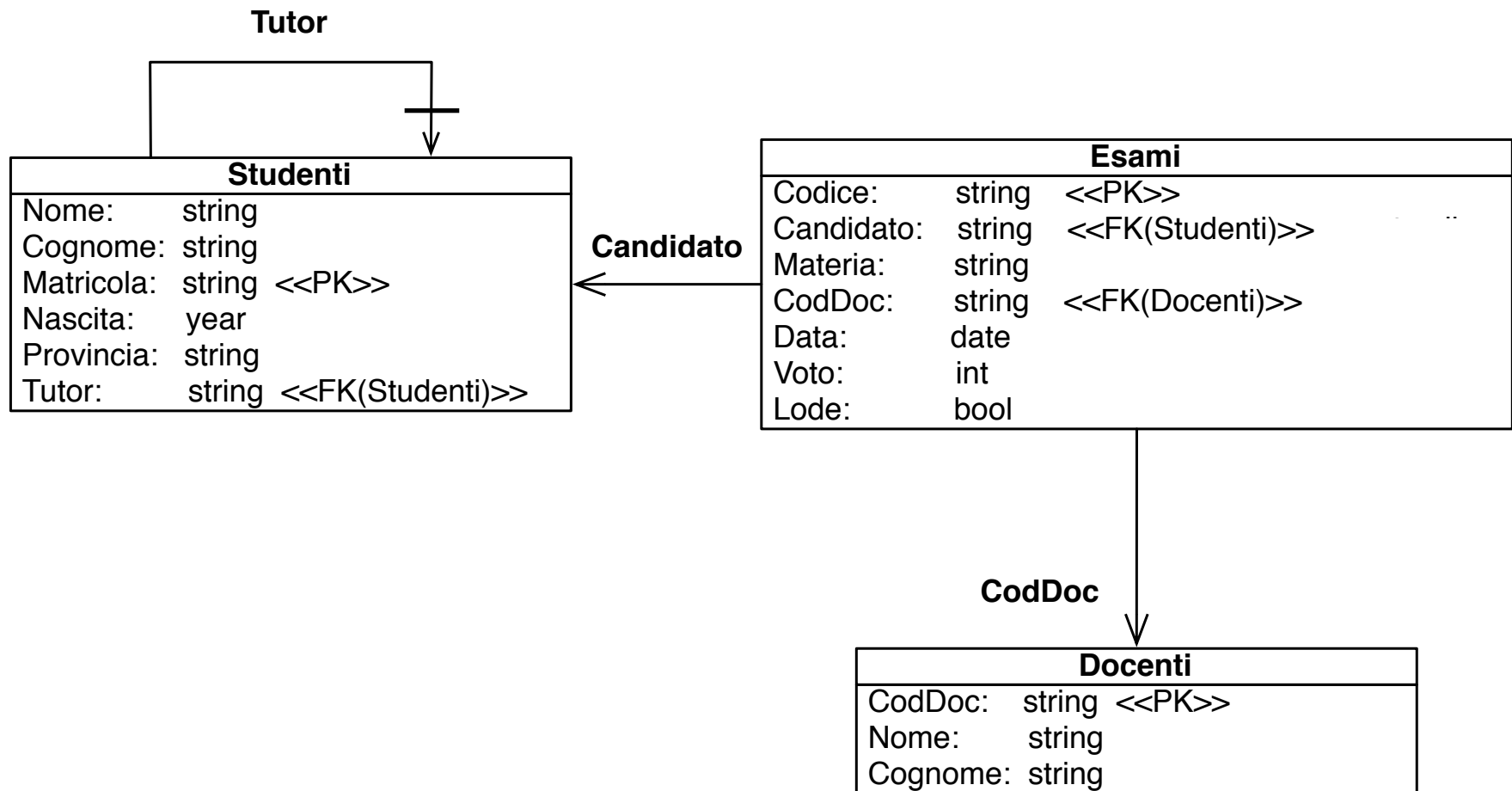
- Linguaggio più diffuso per basi di dati relazionali
 - Nasce nel 1973, all'IBM per il sistema relazionale **System/R**
 - **SEQUEL** (Structured English QUery Language) -> SQL
 - Intorno agli anni '80 inizia un processo di standardizzazione
 - SQL-84, SQL-89, ..., SQL-99, SQL:2003, SQL:2006
- Le implementazioni nei vari DBMS relazionali commerciali
 - includono funzionalità non previste dallo standard
 - non includono funzionalità previste dallo standard
 - implementano funzionalità previste dallo standard ma in modo diverso :-(

- Linguaggio **dichiarativo** basato su **Calcolo Relazionale** su **Ennuple** e **Algebra Relazionale**
 - relazioni -> **tabelle**
 - ennuple -> **record/righe**
 - attributi -> **campi/colonne**
- Le tabelle possono avere righe duplicate (una tabella è un **multinsieme**), per
 - **efficienza**: eliminare i duplicati costa ($n \log(n)$)
 - **flessibilità**:
 - può essere utile vedere i duplicati
 - possono servire per le funzioni di aggregazione (es. media)

- Il linguaggio comprende
 - DML (Data Manipulation Language)
ricerche e/o modifiche interattive -> interrogazioni o query
 - DDL (Data Definition Language)
definizione (e amministrazione) della base di dati
 - uso di SQL in altri linguaggi di programmazione

II DML di SQL

- Consideriamo lo schema relazionale



```
SELECT  s.Nome, s.Cognome, e.Data
FROM    Studenti s JOIN Esami e ON s.Matricola = e.Candidato
WHERE    e.Materia='BD' AND e.Voto=30
```

```
SELECT  s.Nome AS Nome,
          date_part('year', current_date) - s.Nascita AS Età,
          0 AS NumeroEsami
FROM    Studenti s
WHERE NOT EXISTS (SELECT  *
                   FROM    Esami e
                   WHERE    s.Matricola = e.Candidato)
```


- Il comando base dell'SQL:

```
SELECT [DISTINCT] Attributi  
FROM    Tabelle  
[WHERE Condizione]
```

Tabelle ::= Tabella [Ide] { , Tabella [Ide] }

- Condizione può essere una combinazione booleana (**AND**, **OR**, **NOT**) di (dis)uguaglianze tra attributi (=, <, <=, ...) ... ma anche molto altro.

- Semantica: prodotto + restrizione + proiezione.

- **SELECT** *

```
FROM   R1 JOIN R2 ON C2
        ... JOIN Rn ON Cn
WHERE  C
```

$$\sigma_C(R_1 \bowtie_{C_2} R_2 \cdots \bowtie_{C_n} R_n)$$

- **SELECT** DISTINCT A1, ..., An

```
FROM   R1 JOIN R2 ON C2
        ... JOIN Rn ON Cn
WHERE  C
```

$$\delta(\pi_{A_1, \dots, A_n}^b(\sigma_C(R_1 \bowtie_{C_2} R_2 \cdots \bowtie_{C_n} R_n)))$$

- **SELECT ***
FROM Studenti;
- **SELECT ***
FROM Esami
WHERE Voto > 26;
- **SELECT DISTINCT** Provincia
FROM Studenti;
- **SELECT ***
FROM Studenti **JOIN** Esami **ON** Matricola = Candidato;

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

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

Nome	Matricola	Provincia
Paolo	71523	VE
Anna	76366	PD
Chiara	71347	VE

Studenti

Nome	Cognome	Matricola	Provincia	Nascita	Tutor
Paolo	Verdi	71523	VE	1986	NULL
Chiara	Scuri	71346	VE	1987	71523
Paolo	Poli	71576	PD	1985	76366
Anna	Rossi	76366	PD	1987	NULL
Giorgio	Zeri	71347	VE	1986	76366

Esami

<u>Codice</u>	Materia	Candidato*	Data	Voto	Lode	CodDoc*
B112	BD	71523	08.07.06	27	N	AM1
F31	FIS	76366	08.07.07	30	N	GL1
B247	BD	76366	18.07.07	30	S	AM1
A143	ALG	71523	28.12.06	25	N	NG2
A213	ALG	71576	19.07.07	21	N	NG2
F45	FIS	71576	29.07.07	22	N	GL1

- Trovare tutti i dati degli studenti di Venezia:

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

Nome	Cognome	...	Provincia	...
Paolo	Verdi	...	VE	...
Chiara	Scuri	...	VE	...

- Trovare nome, matricola e anno di nascita degli studenti di Venezia
(Proiezione+Restrizione):

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

Nome	Matricola	Nascita
Paolo	71523	1989
Chiara	71346	1987

- Tutte le possibili coppie (Studente, Esame):

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

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

```
SELECT  *  
FROM    Studenti JOIN Esami ON Matricola = Candidato
```

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

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

- Se si opera sul prodotto di tabelle con attributi omonimi occorre **qualificarli**, ovvero identificare la tabella alla quale ciascun attributo si riferisce
- **Notazione con il Punto**. Utile se si opera su tabelle **diverse** con attributi aventi lo stesso nome

Tabella.Attributo

- 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 JOIN Docenti ON Docenti.CodDoc = Esami.CodDoc
```


● Alias

- Si associa un identificatore alle relazioni in gioco
- Essenziale se si opera su più copie della stessa relazione (-> associazioni ricorsive!)
- 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 JOIN Studenti t ON s.Tutor = t.Matricola
```

- La qualificazione è sempre possibile e può rendere la query più leggibile

- Gli alias permettono di avere 'ricorsività' a un numero arbitrario di livelli.

Esempio:

```
Persone (Id, Nome, Cognome, IdPadre, Lavoro)  
PK(Id), IdPadre FK(Persone)
```

```
SELECT n.Nome, n.Cognome,  
        f.Nome, f.Cognome  
FROM   Persone n,  
        Persone p,  
        Persone f  
WHERE  f.IdPadre = p.Id AND  
        p.IdPadre = n.Id AND  
        n.Lavoro = f.Lavoro
```

Cognome e nome delle
persone (e dei nonni) che
fanno lo stesso lavoro dei
nonni

- **Attributi ::= ***
| Expr [[**AS**] Nome] {, Expr [[**AS**] Nome] }

Expr AS Nome: dà il nome Nome alla colonna ottenuta come risultato dell'espressione **Expr**

- usato per rinominare attributi o più comunemente per dare un nome ad un attributo calcolato

```
SELECT Nome, Cognome, date_part('year', current_date)-Nascita  
AS Età  
FROM    Studenti  
WHERE   Provincia='VE'
```

- **Nota**: Un attributo A di una tabella "R x" si denota come: A oppure R.A oppure x.A

- Le **espressioni** possono includere operatori aritmetici (o altri operatori e funzioni sui tipi degli attributi) o funzioni di **aggregazione**
 - $\text{Expr} ::= [\text{Ide.}] \text{Attributo} \mid \text{Const}$
 $\mid (\text{Expr}) \mid [-] \text{Expr} [\text{Op Expr}]$
 $\mid \text{COUNT} (*)$
 $\mid \text{AggrFun} ([\text{DISTINCT}] [\text{Ide.}] \text{Attributo})$
 - $\text{AggrFun} ::= \text{SUM} \mid \text{COUNT} \mid \text{AVG} \mid \text{MAX} \mid \text{MIN}$
- NB: si usano tutte funzioni di aggregazione (-> produce un'unica riga) o nessuna.
- Le funzioni di aggregazione **NON** possono essere usate nella clausola WHERE

- 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
```

- è diverso da

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

- Nota: non ha senso ... (vedi GROUP BY)

```
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
```

- Le tabelle si possono combinare usando:

- “,” (prodotto): `FROM T1,T2`
- **Giunzioni** di vario genere

● `Tabelle ::= Tabella [Ide] { , Tabella [Ide] } |`

`Tabella Giunzione Tabella`
`[USING (Attributi) | ON Condizione]`

`Giunzione ::= [CROSS | NATURAL] [LEFT | RIGHT | FULL] JOIN`

- **CROSS JOIN**

realizza il prodotto

```
SELECT *  
FROM    Esami CROSS JOIN Docenti
```

- **NATURAL JOIN**

è il join naturale

```
SELECT *  
FROM    Esami NATURAL JOIN Docenti;
```


● **JOIN ... ON Condizione**

effettua il join su di una condizione (es. indica quali valori devono essere uguali)

```
SELECT *
```

```
FROM    Studenti s JOIN Studenti t ON s.Tutor = t.Matricola;
```

● **JOIN ... USING** *Alcuni attributi comuni*

come il natural join, ma solo sugli attributi comuni elencati

```
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**

se precedono **JOIN**, effettuano la corrispondente **giunzione esterna**

- **Esempio:** Esami di tutti gli studenti, con nome e cognome relativo, elencando anche gli studenti che non hanno fatto esami

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

● 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

Nota: compaiono anche ennuple corrispondenti a studenti che non hanno fatto esami, [completate con valori nulli](#).

- Inserendo la clausola

ORDER BY *Attributo* [**DESC**|**ASC**] {, *Attributo* [**DESC**|**ASC**] }

si può far sì che la tabella risultante sia ordinata, secondo gli attributi indicati (ordine lessicografico) in modo crescente (**ASC**) [default] o decrescente (**DESC**):

e.g.

```
SELECT Nome, Cognome
FROM    Studenti
WHERE   Provincia='VE'
ORDER BY Cognome DESC, Nome DESC
```

- SQL comprende operatori **insiemistici** (UNION, INTERSECT ed EXCEPT) per combinare i risultati di tabelle con colonne di ugual nome e ugual tipo
- 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;
```

- Se le tabelle sulle quali operare hanno attributi con lo stesso tipo, ma con nome diverso, si possono rinominare con AS
- Esempio: Le matricole degli studenti che non sono tutor

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

- Effettuano la rimozione dei duplicati, a meno che non sia esplicitamente richiesto il contrario 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 di un campo di un'ennupla può mancare per varie ragioni
 - attributo non applicabile
 - attributo non disponibile
 - ...
- SQL fornisce il valore speciale **NULL** per tali situazioni.
- La presenza di NULL introduce dei problemi:
 - la condizione "**Voto=28**" è vera o falsa quando il Voto è NULL?
è vero **NULL=NULL**?
Cosa succede degli operatori **AND**, **OR** e **NOT**?

- Dato che NULL può avere diversi significati
 - **NULL=0** non è né vero, né falso, ma **unknown**
 - anche **NULL=NULL** è **unknown**
- Occorre una logica a 3 valori (vero, falso e unknown).

p	$\neg p$
T	F
F	T
U	U

p	q	$p \wedge q$	$p \vee q$
T	T	T	T
T	F	F	T
T	U	U	T
F	T	F	T
F	F	F	F
F	U	F	U
U	T	U	T
U	F	F	U
U	U	U	U

- Va definita opportunamente la semantica dei costrutti. Ad es.

SELECT ... FROM ...

WHERE *COND*

restituisce solo le ennuple che rendono **vera** la condizione *COND*.

- Necessario un predicato per il **test di nullità**

Expr **IS** [**NOT**] NULL

è vero se *Expr* (non) è NULL

- Nota che NULL=NULL vale UNKNOWN!!
- Nuovi operatori sono utili (es. giunzioni esterne)

- Gli studenti che non hanno Tutor

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

Nome	Cognome	Matricola	Nascita	Provincia	Tutor
Anna	Rossi	76366	1987	PD	NULL
Paolo	Verdi	71523	1986	VE	NULL

- Cosa ritorna?

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

- **Expr₁ IS [NOT] DISTINCT FROM Expr₂**

è vero se i due valori sono diversi, o uno solo dei due è NULL;

è falso quando i due valori sono uguali anche nel caso in cui sono entrambi uguali a NULL.

- **COALESCE(Expr₁, ..., Expr_n)**

Viene usato per trasformare un valore NULL in un valore non nullo. Valuta le espressioni in sequenza, da sinistra verso destra. Viene restituito il primo valore trovato diverso da NULL. L'operatore ritorna NULL se tutte le espressioni hanno valore NULL.

- Su valori numerici
 - **WHERE** *Expr* **BETWEEN** *Expr* **AND** *Expr*

- **SELECT** *
FROM Studenti
WHERE Matricola **BETWEEN** 71000 **AND** 72000;

Nome	Cognome	Matricola	Nascita	Provincia	Tutor
Chiara	Scuri	71346	1987	VE	71523
Giorgio	Zeri	71347	1986	VE	76366
Paolo	Verdi	71523	1986	VE	NULL
Paolo	Poli	71576	1985	PD	76366

- Sulle stringhe
 - **WHERE** *Expr* **LIKE** *pattern*
- Il pattern può contenere caratteri e i simboli speciali
 - % sequenza di 0 o più caratteri qualsiasi
 - _ un carattere qualsiasi
- 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'
```

- stessa query usando le [espressioni regolari](#) (sintassi Oracle)

```
SELECT *  
FROM    Studenti  
WHERE    REGEXP_LIKE (Nome, '^A.*(a|i)$')
```

- La clausola WHERE è più complicata di come l'abbiamo vista finora.
- 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)

- Alcune interrogazioni richiedono di estrarre dati dalla BD e usarli in operazioni di confronto
- E' possibile specificare select **annidate**, inserendo nel campo **WHERE** una condizione che usa una select (che a sua volta può contenere sottoselect ...)
- Si può
 - eseguire **confronti con l'insieme** di valori ritornati dalla sottoselect (sia quando questo è un singoletto, sia quando contiene più elementi)
 - verificare la **presenza/assenza** di valori dati nell'insieme ritornato dalla sottoselect
 - verificare se l'insieme di valori ritornato dalla sottoselect è o meno **vuoto**

- Nel campo WHERE
 - ***Expr Comp(Sottoselect che torna esattamente un valore)***
- 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' )
```

- E' indispensabile la sottoselect?

```
SELECT  altri.*
FROM    Studenti altri, Studenti s
WHERE    altri.Matricola <> '71346' AND
           s.Matricola = '71346' AND altri.Provincia = s.Provincia
```

- ... è un join

```
SELECT  altri.*  
FROM    Studenti altri JOIN Studenti s USING (Provincia)  
WHERE   altri.Matricola <> '71346' AND  
         s.Matricola = '71346';
```

- Le interrogazioni su di una associazione multivalore vanno quantificate



- Non: gli studenti che hanno preso 30 (ambiguo!)
ma:
 - Gli studenti che hanno preso sempre (solo, tutti) 30: universale
 - Gli studenti che hanno preso qualche (almeno un) 30: esistenziale
 - Gli studenti che non hanno preso mai 30 (senza alcun 30): universale
 - Gli studenti che non hanno preso sempre 30: esistenziale

- Universale negata = esistenziale:

- Non tutti i voti sono =30 (universale) = esiste un voto ≠30 (esistenziale)
- Più formalmente

$$\neg \forall x. P(x) \equiv \exists x. \neg P(x)$$

- Esistenziale negata = universale:

- Non esiste un voto diverso da 30 (esistenziale) = Tutti i voti sono uguali a 30 (universale)
- Più formalmente

$$\neg \exists x. \neg P(x) \equiv \forall x. P(x)$$

- Come condizione nel **WHERE** possiamo usare

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

- La query studenti con almeno un voto > 27

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


- Query con EXISTS:

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

- La stessa query, ovvero gli studenti con almeno un voto > 27, tramite giunzione:

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

- Un altro costrutto che permette una quantificazione esistenziale

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

- La solita query
“Studenti che hanno preso almeno un voto > 27”
si può esprimere anche tramite ANY ...

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

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

- ANY non fa nulla in più di EXISTS

```
SELECT *  
FROM    Tab1  
WHERE   attr1 op ANY (SELECT attr2  
                                FROM    Tab2  
                                WHERE   C);
```

- diventa

```
SELECT *  
FROM    Tab1  
WHERE   EXISTS (SELECT *  
                                FROM    Tab2  
                                WHERE   C AND attr1 op attr2);
```

- Forma ancora più blanda di quantificazione esistenziale:

```
SELECT ...
```

```
FROM ...
```

```
WHERE Expr IN (sottoselect)
```

- **Nota:** abbreviazione di =ANY
- La solita query si può esprimere anche tramite IN:

```
SELECT *
```

```
FROM Studenti s
```

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

- Può essere utilizzato con ennuple di valori

`Expr IN (val1, val2, ..., valn)`

- Gli studenti di Padova, Venezia e Belluno

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

-
- La quantificazione esistenziale si fa con:
 - EXISTS
 - Giunzione
 - =ANY, >ANY, <ANY, ...
 - IN
 - Il problema vero è: non confondere esistenziale con universale!

- Gli studenti che hanno preso solo 30
- Errore comune (e grave):

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

- In stile OQL: **FORALL** e **IN** s.HaSostenuto: e.Voto=30

```
SELECT *  
FROM   Studenti s  
WHERE  FORALL e IN Esami WHERE e.Candidato = s.Matricola:  
                                     e.Voto = 30
```


- In SQL non c'è un operatore generale esplicito **FOR ALL**. Si usa l'equivalenza logica

$$\forall e \in E. P \equiv \neg(\exists e \in E. \neg P)$$

- Quindi da:

```
SELECT *  
FROM    Studenti s  
WHERE FORALL e IN Esami WHERE e.Candidato = s.Matricola:  
                                e.Voto = 30
```

- ... si può passare a ...

- In SQL diventa:

```
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) è diventato e.Voto <> 30

- E' disponibile un operatore duale rispetto a ANY, che e' ALL:

```
WHERE Expr Comp ALL (Sottoselect)
```

- Sostituendo EXISTS con =ANY, la solita query (studenti con tutti 30):

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

- Diventa:

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



...

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



Ovvero:

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

- Supponiamo che la BD sia tale che la query
- **SELECT** s.Nome, s.Cognome, e.Materia, e.Voto
FROM Studenti s **LEFT JOIN** Esami e **ON** s.Matricola=e.Candidato;
- ritorni

Nome	Cognome	Materia	Voto
Chiara	Scuri	NULL	NULL
Giorgio	Zeri	NULL	NULL
Paolo	Verdi	BD	27
Paolo	Verdi	ALG	25
Paolo	Poli	ALG	21
Paolo	Poli	FIS	22
Anna	Rossi	BD	30
Anna	Rossi	FIS	30

- Qual e' l'output della query 'studenti che hanno preso solo trenta'?

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

```
+-----+
|  Cognome  |
+-----+
|  Scuri    |
|  Zeri     |
|  Rossi    |
+-----+
```

- Se voglio gli studenti che hanno preso solo trenta, e hanno superato qualche esame:

```
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;
```


- Per ogni materia, trovare nome della materia e voto medio degli esami in quella materia [selezionando solo le materie per le quali sono stati sostenuti più di tre esami]:
 - Per ogni materia vogliamo
 - Il nome, che e' un attributo di Esami
 - Una funzione aggregata sugli esami della materia
- Soluzione:

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

- Costrutto:

SELECT ... FROM ... WHERE ...

GROUP BY A_1, \dots, A_n

[**HAVING** *condizione*]

- Semantica:

- Esegue le clausole FROM - WHERE
- Partiziona la tabella risultante rispetto all'uguaglianza su tutti i campi A_1, \dots, A_n (in questo caso, si assume $\text{NULL} = \text{NULL}$)
- Elimina i gruppi che non rispettano la clausola HAVING
- Da ogni gruppo estrae una riga usando la clausola SELECT

```
SELECT Candidato, COUNT(*) AS NEsami,  
        MIN(Voto), MAX(Voto), AVG(Voto)  
  
FROM    Esami  
  
GROUP BY Candidato  
  
HAVING AVG(Voto) > 23;
```

Codice	Materia	Candidato	Data	Voto	Lode	CodDoc
B112	BD	71523	2006-07-08	27	N	AM1
A143	ALG	71523	2006-12-28	25	N	NG2
B247	BD	76366	2007-07-18	30	L	AM1
A213	ALG	71576	2007-07-19	21	N	NG2
F31	FIS	76366	2007-07-08	30	N	GL1
F45	FIS	71576	2007-07-29	22	N	GL1





Codice	Materia	Candidato	Data	Voto	Lode	CodDoc
B112	BD	71523	2006-07-08	27	N	AM1
A143	ALG	71523	2006-12-28	25	N	NG2
A213	ALG	71576	2007-07-19	21	N	NG2
F45	FIS	71576	2007-07-29	22	N	GL1
B247	BD	76366	2007-07-18	30	L	AM1
F31	FIS	76366	2007-07-08	30	N	GL1

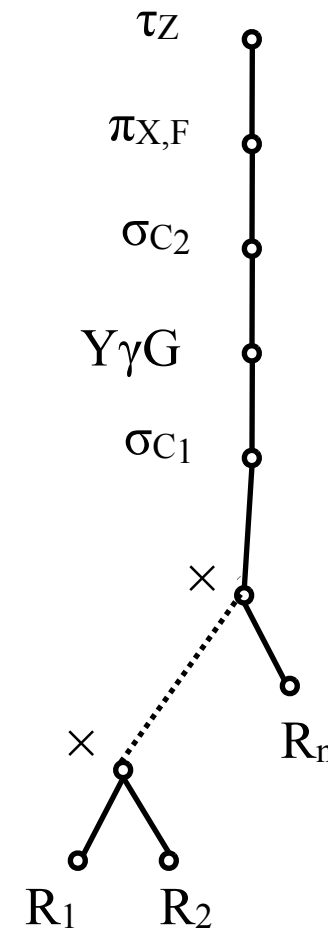


Candidato	NEsami	min(Voto)	max(Voto)	avg(Voto)
71523	2	25	27	26.000
76366	2	30	30	30.000

```

SELECT DISTINCT X, F
FROM R1, ..., Rn
WHERE C1
GROUP BY Y
HAVING C2
ORDER BY Z
    
```

- X, Y, Z sono insiemi di attributi
- F, G sono insiemi di espressioni aggregate, tipo count(*) o sum(A)
- $X, Z \subseteq Y$, $F \subseteq G$, C₂ nomina solo attributi in Y o espressioni in G



- Per ogni studente, cognome e voto medio:

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

- È necessario scrivere:

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

- Gli attributi espressi non aggregati nella select (s.Cognome) e in **HAVING** se presenti (s.Matricola) devono essere inclusi tra quelli citati nella **GROUP BY**
- Gli attributi aggregati (AVG(e.Voto)) vanno scelti tra quelli non raggruppati

- Anche la clausola HAVING cita solo:
 - espressioni su attributi di raggruppamento;
 - funzioni di aggregazione applicate ad attributi non di raggruppamento.
- Non va ...

```
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;
```

- Nel raggruppamento si assume (è uno dei pochi casi) **NULL = NULL**
- Es: Matricole dei tutor e relativo numero di studenti di cui sono tutor

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

Tutor	NStud
NULL	2
71347	2
71523	1

- **CASE** è un'espressione condizionale che può essere usata in qualsiasi contesto dove un'espressione è valida.

CASE WHEN condizione **THEN** risultato

[**WHEN** ...]

[**ELSE** risultatoDefault]

END

condizione è un'espressione che restituisce un valore booleano.

Se la condizione è vera, restituisce risultato e le altre condizioni non sono valutate. Se la condizione non è vera, le successive clausole **WHEN** sono esaminate in ordine. Se nessuna di queste è vera, allora restituisce il risultatoDefault dell'**ELSE**. Se l'**ELSE** non è presente, il risultato è **NULL**.

- Sottoselect:

```
SELECT [DISTINCT] Attributi  
FROM Tabelle  
[WHERE Condizione]  
[GROUP BY  $A_1, \dots, A_n$  [HAVING Condizione]]
```

- Select:

```
Sottoselect  
{ (UNION [ALL] | INTERSECT [ALL] | EXCEPT [ALL])  
  Sottoselect }  
[ ORDER BY Attributo [DESC] {, Attributo [DESC]} ]
```

- **INSERT INTO** Tabella [(A1,...,An)]
(**VALUES** (V1,...,Vn) | **AS** Select)
- **UPDATE** Tabella
SET Attributo = Expr, ..., Attributo = Expr
WHERE Condizione
- **DELETE FROM** Tabella
WHERE Condizione

- La forma base del comando INSERT è la seguente:

```
INSERT INTO Tabella  
VALUES      (valoreA1,...,valoreAn),  
              (valoreB1,...,valoreBn),  
              ...
```

- dove (valoreX1,...,valoreXn) sono righe del tipo corrente di tabella (con gli attributi nella sequenza corretta!) e.g.

```
INSERT INTO Studenti  
VALUES ( 'Paolo','Poli','71576', 'BL', '1986', '76366'),  
        ( 'Giorgio','Conte','71577', 'AT', '1941', NULL);
```

- Alternativamente si può usare la forma:

```
INSERT INTO Tabella(colonna1,...,colonnam)
VALUES      (valoreA1,...,valoreAm),
              (valoreB1,...,valoreBm),
              ...
```

- m può essere < del numero di attributi n (le restanti colonne o prendono il valore di default o NULL)
- le colonne possono apparire in ordine diverso da quello in cui appaiono nella definizione di Tabella

- **Esempio**

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

- Tutti i valori dichiarati **NOT NULL** e senza un valore di default dichiarato devono essere specificati

- E' possibile aggiungere le righe prodotte da una select ...

```
INSERT INTO Tabella AS Select
```

- **Esempio:** se `StNomeCognome (Nome, Cognome)` è una tabella con due campi di tipo adeguato ...

```
INSERT INTO StNomeCognome AS  
SELECT Nome, Cognome FROM Studenti;
```

- La forma base del comando DELETE è la seguente:

```
DELETE FROM Tabella  
WHERE condizione
```

- Cancella da Tabella le righe che soddisfano la condizione in WHERE: e.g.

```
DELETE FROM Esami  
WHERE Voto<18;
```

- Senza la clausola WHERE

```
DELETE FROM Esami;
```

cancella tutte le righe (ma non la tabella)

- La selezione delle righe da cancellare può essere basata anche su di una select.
Es. Cancella gli studenti che non hanno sostenuto esami

```
DELETE FROM Studenti
```

```
WHERE Matricola NOT IN (SELECT Candidato FROM Esami);
```

- Strutturalmente simile alla **SELECT** (ma cancella intere righe)

- La forma base del comando UPDATE è:

UPDATE Tabella

SET attr1=exp1, ...,
attrn=expn

WHERE condizione

dove attri ed expi devono avere il medesimo tipo; e.g.

- Esempio:

UPDATE Studenti

SET Tutor='71523'

WHERE Matricola='76366' **OR** Matricola='76367'

- Aumenta di 1 punto il voto a tutti gli esami con voto > 23

UPDATE Esami

SET Voto=Voto+1

WHERE Voto>23 **AND** Voto<30;

- Anche in questo caso si possono usare condizioni che coinvolgono **SELECT**