

# Basi di Dati - V

Corso di Laurea in Informatica  
Anno Accademico 2013/2014

Paolo Baldan

[baldan@math.unipd.it](mailto:baldan@math.unipd.it)

<http://www.math.unipd.it/~baldan>

# Il linguaggio SQL

## Il linguaggio SQL

3

- 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 :-)

## Il linguaggio SQL (cont.)

4

- Linguaggio dichiarativo basato su **Calcolo Relazionale su Ennuple e Algebra Relazionale**
  - relazioni -> **tabelle**
  - ennuple -> **record/righe**
  - attributi -> **campi/colonne**

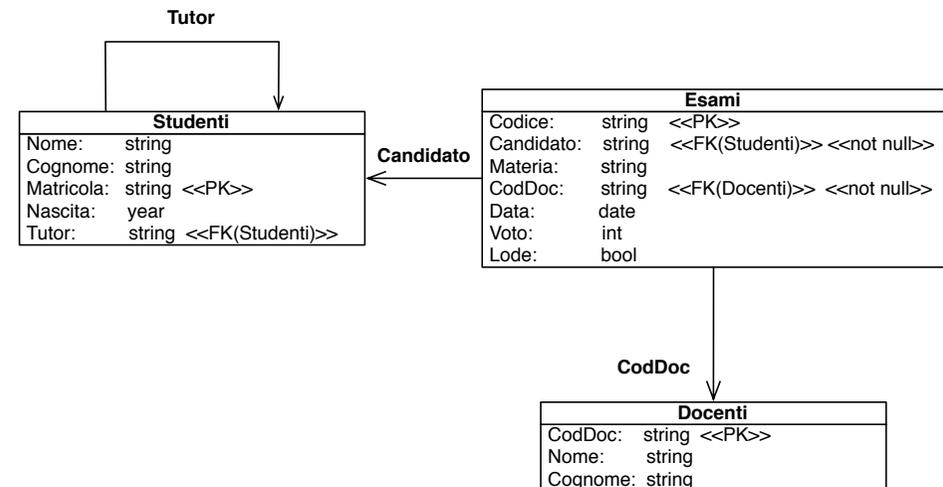
- Linguaggio dichiarativo basato su *Calcolo Relazionale su Ennuple e Algebra Relazionale*
  - relazioni -> *tabelle*
  - ennuple -> *record/righe*
  - attributi -> *campi/colonne*
- Le relazioni possono avere righe duplicate (una tabella è un *multinsieme*), per
  - *efficienza*: eliminare i duplicati 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

## Il DML di SQL

## Un assaggio ...

- 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, YEAR(CURDATE()) - 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.

## Il comando SELECT (cont.)

11

- Semantica: prodotto + restrizione + proiezione.

```

● SELECT *
  FROM   R1, ..., Rn

```

$$R_1 \times \dots \times R_n$$

## Il comando SELECT (cont.)

11

- Semantica: prodotto + restrizione + proiezione.

```

● SELECT *
  FROM   R1, ..., Rn

```

$$R_1 \times \dots \times R_n$$

```

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

```

$$\sigma_C(R_1 \times \dots \times R_n)$$

- Semantica: prodotto + restrizione + proiezione.

- **SELECT** \*  
**FROM** R1, ..., Rn  $R_1 \times \dots \times R_n$

- **SELECT** \*  
**FROM** R1, ..., Rn  $\sigma_C(R_1 \times \dots \times R_n)$   
**WHERE** C

- **SELECT DISTINCT** A1, ..., An  
**FROM** R1, ..., Rn  $\pi_{A_1, \dots, A_n}(\sigma_C(R_1 \times \dots \times R_n))$   
**WHERE** C

- **SELECT** \*  
**FROM** Studenti;

- **SELECT** \*  
**FROM** Studenti;

- **SELECT** \*  
**FROM** Esami  
**WHERE** Voto > 26;

- **SELECT** \*  
**FROM** Studenti;

- **SELECT** \*  
**FROM** Esami  
**WHERE** Voto > 26;

- **SELECT DISTINCT** Provincia  
**FROM** Studenti;

- **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
```

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

- Trovare tutti i dati degli studenti di Padova:

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

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

- Trovare tutti i dati degli studenti di Padova:

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

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

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

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

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, 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 Materia='BD' AND Voto=30
AND Matricola = Candidato
```

- 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, Docenti
WHERE 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, Studenti t
WHERE 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 Nonno.Nome, Figlio.Nome
FROM Persone AS Figlio,
     Persone AS Padre,
     Persone AS Nonno
WHERE Figlio.IdPadre = Padre.Id AND
      Padre.IdPadre = Nonno.Id AND
      Nonno.Lavoro = Figlio.Lavoro
```

**Nomi delle persone 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, YEAR(CURDATE())-Nascita AS Età
FROM Studenti
WHERE Provincia='PD'
```

- **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**

- Expr ::= [Ide.]Attributo | Const  
| ( Expr ) | [-] Expr [Op Expr]  
| **COUNT**(\*)  
| AggrFun ( [**DISTINCT**] [Ide.]Attributo)

- AggrFun ::= **SUM** | **COUNT** | **AVG** | **MAX** | **MIN**

- NB: si usano tutte funzioni di aggregazione (-> produce un'unica riga) o nessuna.

## ESEMPI: funzioni di aggregazione

- 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
```

## ESEMPI: funzioni di aggregazione

- Numero di Studenti che hanno un Tutor

- Numero di Studenti che hanno un Tutor
- Numero di studenti che fanno i Tutor

- 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** (oppure **INNER JOIN**)

realizza il prodotto

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

- **NATURAL JOIN**

è il join naturale

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

- **JOIN ... USING** *Alcuni attributi comuni*  
come il natural join, ma solo sugli attributi comuni elencati

- **JOIN ... ON** **Condizione**

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

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

- **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 del **JOIN** semplice

Nome	Cognome	Matricola	Data	Materia
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

- Risultato del **LEFT JOIN**

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='PD'
ORDER BY Cognome (DESC), Nome (DESC)
```

- Nota: In MySQL ASC e DESC senza parentesi

- SQL comprende operatori insiemistici (UNION, INTERSECTION ed EXCEPT) per combinare i risultati di tabelle con colonne di ugual nome e ugual tipo
- Es: Nome e cognome degli studenti di Padova e di quelli che hanno preso più di 28 in qualche esame

```

SELECT Nome, Cognome
FROM Studenti
WHERE Provincia='PD'
UNION
SELECT Nome, Cognome
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

```
- Nota: EXCEPT e INTERSECTION non presenti in MySQL

- Effettuano la rimozione dei duplicati, a meno che non sia esplicitamente richiesto il contrario con l'opzione ALL
- Es: Nome e cognome degli studenti di Padova oppure che hanno preso più di 28 in qualche esame

```

SELECT Nome, Cognome
FROM Studenti
WHERE Provincia='PD'
UNION ALL
SELECT Nome, Cognome
FROM Studenti JOIN Esami ON (Matricola=Candidato)
WHERE Voto>28;

```

- 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 "**Matricola=9**" è vera o falsa quando la Matricola è NULL?
    - è vero **NULL=NULL?**
    - Cosa succede degli operatori **AND**, **OR** e **NOT?**

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

p	¬p
T	F
F	T
U	U

p	q	p∧q	p∨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 **NULL!!**
- Nuovi operatori sono utili (es. giunzioni esterne)

## Il valore NULL: Esempio

- Gli studenti che non hanno Tutor

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

Nome	Cognome	Matricola	Nascita	Provincia	Tutor
Giorgio	Zeri	71347	1987	PD	NULL
Paolo	Verdi	71523	1986	PD	NULL

- Cosa ritorna?

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

## Altre condizioni: between

- 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     | 1985    | PD        | 71347 |
| Giorgio | Zeri    | 71347     | 1987    | PD        | NULL  |
| Paolo   | Verdi   | 71523     | 1986    | PD        | NULL  |
| Paolo   | Poli    | 71576     | 1988    | VE        | 71523 |

- 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](#)

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

- MySQL ha anche il costrutto 'Exp **SOUNDS LIKE** Exp'

- 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)*

- 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

- 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?

- 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';

## Quantificazione

45

- Le interrogazioni su di una associazione multivalevole 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

## Quantificazione

46

- Universale negata = esistenziale:
  - Non tutti i voti sono =30 (universale) = esiste un voto ≠30 (esistenziale)
  - Più formalmente  $\neg\forall x.P(x) \equiv \exists x.\neg P(x)$

- 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 ≠ 30 (esistenziale) = Tutti i voti sono uguali a 30 (universale)

- Più formalmente

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

- Gli studenti con almeno un voto > 27; servirebbe un quantificatore

**EXISTS** e **IN** s.HaSostenuto: e.Voto > 27 (stile **OQL**)

- Gli studenti con almeno un voto > 27; servirebbe un quantificatore

**EXISTS** e **IN** s.HaSostenuto: e.Voto > 27 (stile **OQL**)

- Sarebbe bello poterlo tradurre come ...

- Gli studenti con almeno un voto > 27; servirebbe un quantificatore

**EXISTS** e **IN** s.HaSostenuto: e.Voto > 27 (stile **OQL**)

- Sarebbe bello poterlo tradurre come ...

```
SELECT *
FROM Studenti s
WHERE EXISTS e IN Esami WHERE e.Candidato = s.Matricola:
e.Voto > 27
```

- Gli studenti con almeno un voto > 27; servirebbe un quantificatore

**EXISTS** e **IN** s.HaSostenuto: e.Voto > 27 (stile **OQL**)

- Sarebbe bello poterlo tradurre come ...

```
SELECT *
FROM Studenti s
WHERE EXISTS e IN Esami WHERE e.Candidato = s.Matricola:
      e.Voto > 27
```

- Altra query esistenziale
  - gli studenti che non hanno preso 30 in tutti gli esami, ovvero
  - gli studenti per i quali qualche esame ha voto diverso da 30:

- Altra query esistenziale
  - gli studenti che non hanno preso 30 in tutti gli esami, ovvero
  - gli studenti per i quali qualche esame ha voto diverso da 30:
- diverrebbe:

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

- Altra query esistenziale
  - gli studenti che non hanno preso 30 in tutti gli esami, ovvero
  - gli studenti per i quali qualche esame ha voto diverso da 30:
- diverrebbe:

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

- Purtroppo un costruito così non c'è in SQL ma ...

- Come condizione nel **WHERE** possiamo usare

```
SELECT ...
FROM ...
WHERE [NOT] EXISTS (Sottoselect)
```

- 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  $t$

## Quantificazione esistenziale: EXISTS

50

- La query studenti con almeno un voto > 27

```
SELECT *
FROM Studenti s
WHERE EXIST e IN Esami WHERE e.Candidato = s.Matricola:
e.Voto > 27
```

## Quantificazione esistenziale: EXISTS

50

- La query studenti con almeno un voto > 27

```
SELECT *
FROM Studenti s
WHERE EXIST e IN Esami WHERE e.Candidato = s.Matricola:
e.Voto > 27
```

- in SQL diventa:

```
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)
```

- 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 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)
```

- 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)
```

- 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)
```

- 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);
```

- 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)
```

- Forma ancora più blanda di quantificazione esistenziale:

```
SELECT ...
FROM ...
WHERE Expr IN (sottoselect)
```

- Nota: abbreviazione di =ANY

- 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)
```

- 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');
```

## Riassumendo ...

57

- La quantificazione esistenziale si fa con:
  - EXISTS (il più "espressivo")
  - Giunzione
  - =ANY, >ANY, <ANY, ...
  - IN
- Però: =ANY, >ANY, <ANY, IN,... non aggiungono nulla, EXISTS basta e avanza
- Il problema vero è: **non confondere esistenziale con universale!**

## Quantificazione Universale

58

- Gli studenti che hanno preso solo 30

- 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
```

- 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

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

- 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
```

- 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 ...

- ... si può passare a

```
SELECT *
FROM Studenti
WHERE NOT EXIST e IN Esami WHERE e.Candidato = s.Matricola:
      e.Voto <> 30
```

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

- ... si può passare a

```
SELECT *
FROM Studenti
WHERE NOT EXIST e IN Esami WHERE e.Candidato = s.Matricola:
      e.Voto <> 30
```

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

- In SQL diventa:

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

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

```
WHERE Espr Comp ALL (Sottoselect)
```

## Quantificazione universale: ALL

61

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

```
WHERE Espr 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)
```

## Quantificazione universale: ALL

61

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

```
WHERE Espr 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))
```

## Quantificazione universale: ALL

62

- ...

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

## Quantificazione universale: ALL

62

- ...

```
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)
```

```

...
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)
    
```

E naturalmente, <>ALL è lo stesso di NOT IN ...

```

SELECT * FROM Studenti s
WHERE s.Matricola NOT IN (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     | 30   |
| Paolo   | Poli    | ALG     | 22   |
| Paolo   | Poli    | FIS     | 22   |
| Anna    | Rossi   | BD      | 30   |
| Anna    | Rossi   | FIS     | 30   |

Qual e' l'ouput della query 'studenti che hanno preso tutti trenta'?

```

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

Qual e' l'ouput della query 'studenti che hanno preso tutti 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   |

- Qual è 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)
```

- Cosa cambia se invece di **NOT EXISTS** uso **<>ALL** oppure **NOT IN**?

- 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:

- Per ogni materia, trovare nome della materia e voto medio degli esami in quella materia:
- Per ogni materia vogliamo
  - Il nome, che e` un attributo di Esami
  - Una funzione aggregata sugli esami della materia

- Per ogni materia, trovare nome della materia e voto medio degli esami in quella materia:
- 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
```

- 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

- 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:

● Costrutto:

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

● Costrutto:

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

● Semantica:

- 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)
- 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    |
| 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    |



| 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    |

| 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    |

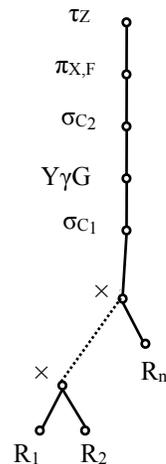
| Candidato | NEsami | min(Voto) | max(Voto) | avg(Voto) |
|-----------|--------|-----------|-----------|-----------|
| 71523     | 2      | 27        | 30        | 28.5000   |
| 76366     | 2      | 26        | 29        | 27.5000   |

## SQL → ALGEBRA

```

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 ⊆ Y, F ⊆ G, C<sub>2</sub> nomina solo attributi in Y o espressioni in G



## Raggruppamento

- Per ogni studente, cognome e voto medio:

```

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

- Per ogni studente, cognome e voto medio:

```
SELECT s.Matricola, 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
```

- Per ogni studente, cognome e voto medio:

```
SELECT s.Matricola, 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 presente (s.Matricola) devono essere inclusi tra quelli citati nella **GROUP BY**

- Per ogni studente, cognome e voto medio:

```
SELECT s.Matricola, 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 presente (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.

- 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 (COUNT(*) > 3) AND (YEAR(Data) > 2006);
```

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

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

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

- Sottoselect:

```
SELECT [DISTINCT] Attributi
FROM Tabelle
[WHERE Condizione]
[GROUP BY A1,..,An [HAVING Condizione]]
```

- Select:

```
Sottoselect
{ (UNION | INTERSECT | EXCEPT)
  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','1986','BL'),
       ('Giorgio','Conte','71577','1941','AT'),
       ('Paolo','Conte','71577','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 Studente (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;
```

- MySQL: senza AS

- 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)

- MySQL: La select non può riguardare la tabella stessa.

- 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

- MySQL: La select non può riguardare la tabella stessa.

- Dato il seguente schema relazionale

Attori(CodiceAtt, Nome, AnnoNascita)

AttoriFilm(CodiceAtt\*, CodiceFilm\*)

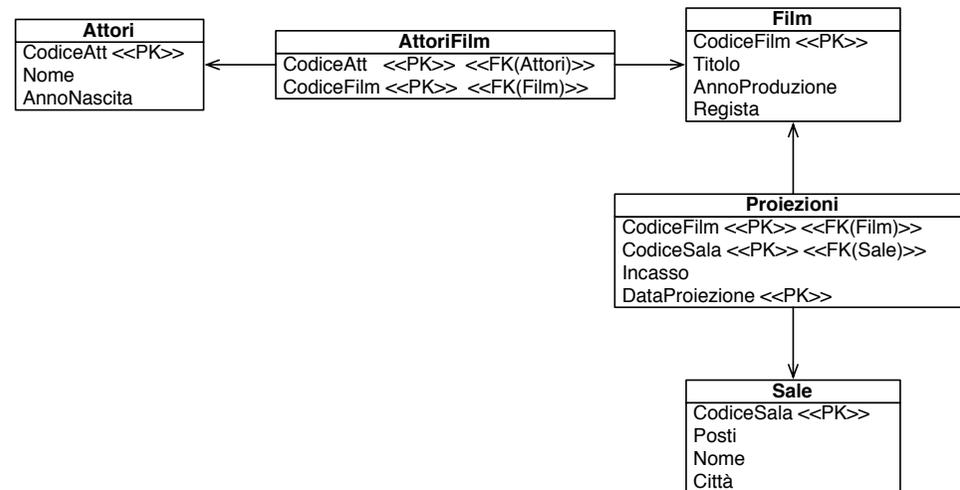
Film(CodiceFilm, Titolo, AnnoProduzione, Regista)

Proiezioni(CodiceFilm\*, CodiceSala\*, Incasso, DataProiezione)

Sale(CodiceSala, Posti, Nome, Città)

- dove

- chiavi primarie: attributi sottolineati
- chiavi esterne: marcate con \* e la relazione che riferiscono e` individuata dal nome dell'attributo



- Query per

1. Trovare il titolo di tutti i film prodotti prima del 1980.
2. Restituire, senza duplicati, Titolo (e Codice) di tutti i film dei quali c'è stata almeno una proiezione
3. Restituire, senza duplicati, il CodiceFilm di tutti i film per i quali c'è stata una proiezione con incasso maggiore di 2000 euro in una sala che avesse meno di 100 posti oppure che si trovasse a Venezia.
4. Per ogni film in cui appare un attore nato prima del 1970 restituire il titolo del film e il nome dell'attore.
5. Per ogni film in cui appaiono due attori diversi nati lo stesso anno trovare il codice del film ed i nomi dei due attori.

6. Per ogni città restituire il nome della città ed il numero di sale con più di 100 posti.
7. Per ogni città il numero di sale, se il numero di posti complessivi è maggiore di 100
8. Per ogni regista, restituire il nome e l'incasso totale di tutte le proiezioni dei suoi film.
9. Per ogni film di Nanni Moretti restituire il titolo, il numero totale di proiezioni a Venezia e l'incasso totale (sempre a Venezia).
10. Nome degli attori che hanno recitato in un solo film
11. Eliminare i film che non sono mai stati proiettati

12. Trasformare euro in lire negli incassi

13. Crea l'archivio delle proiezioni effettuate prima del 2000 e cancellale da Proiezioni