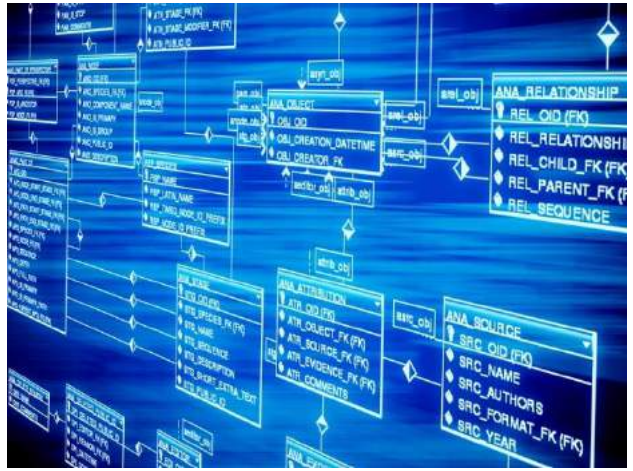
Politecnico di Milano - AA 2018-2019

Prof.ssa Sara Comai

# SQL – Query Language



# SQL

Il nome sta per *Structured Query Language*

Si compone di:

- DDL: definizione di domini, tabelle, indici, autorizzazioni, viste, vincoli, trigger, …
- DML: linguaggio di **query**, linguaggio di modifica, comandi transazionali

Storia:

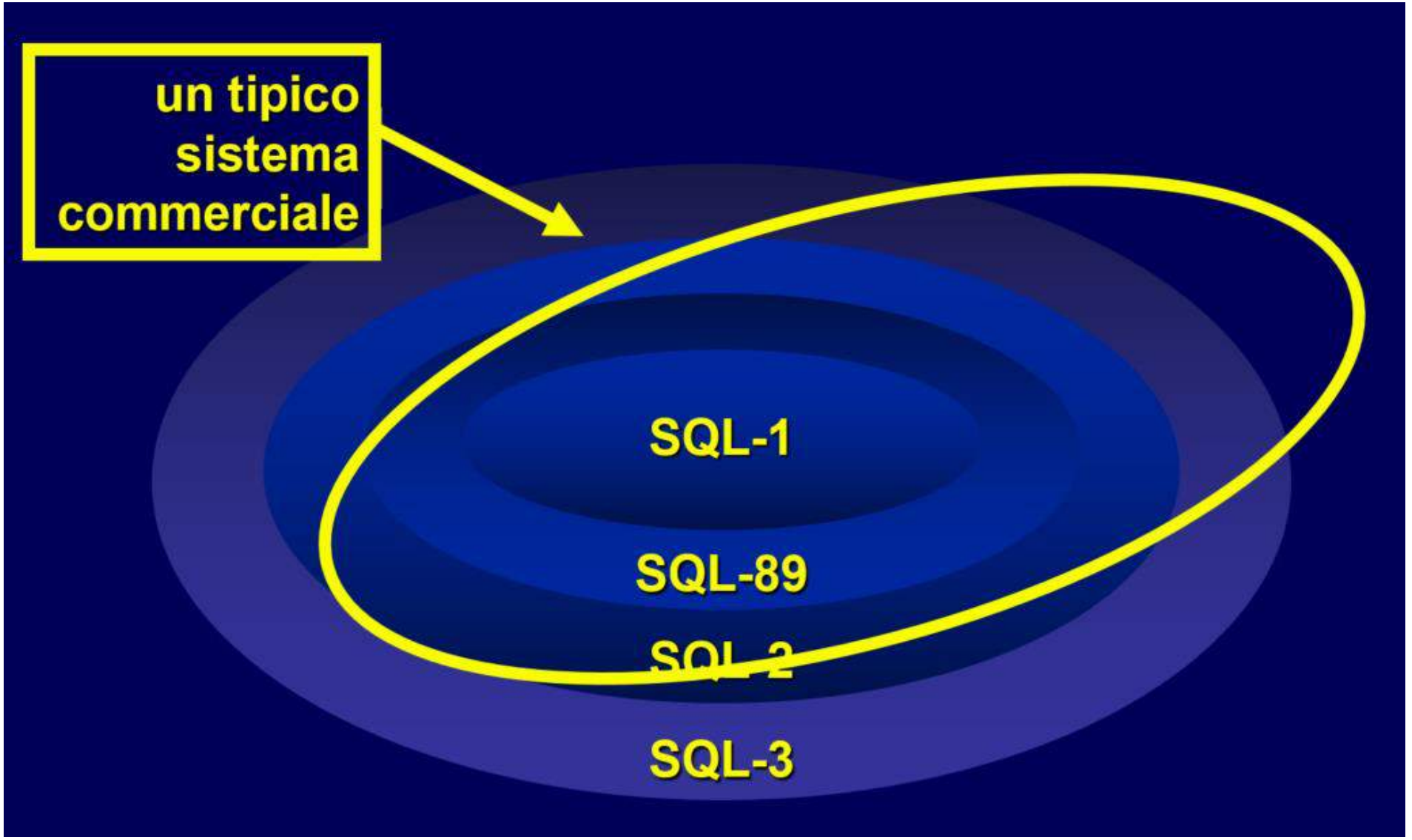
- Prima proposta: SEQUEL (IBM Research, 1974)
- Prima implementazione commerciale in SQL/DS (IBM, 1981 – DB2 1983)
- Standardizzazione (1986-2011)

# Standardizzazione di SQL

La standardizzazione è stata cruciale per la diffusione di SQL (nell'ambito di ANSI e ISO)

- Dal 1983, standard de facto
- SQL-1: SQL-86 (costrutti di base), SQL-89 (integrità referenziale)
- SQL-2: SQL-92 – versione maggiormente adottata tuttora
- SQL-3: SQL:1999 e SQL:2003 – versione più completa, con trigger, oggetti, funzioni esterne, estensioni per Java e XML
- SQL 2006: integrazione con XQUERY
- SQL 2011: integrazione con DB temporali

# Potere espressivo di standard e sistemi commerciali



# Interrogazioni SQL

- Le interrogazioni SQL sono dichiarative
  - l'utente specifica **quale** informazione è di suo interesse, ma **non come** estrarla dai dati
- Le interrogazioni vengono tradotte dall'ottimizzatore (query optimizer) nel linguaggio procedurale interno al DBMS
- Il programmatore si focalizza sulla leggibilità, non sull'efficienza
- È l'aspetto più qualificante delle basi di dati relazionali

# Interrogazioni SQL

Le interrogazioni SQL hanno una struttura `select-from-where`

Sintassi:

**`select`** ***AttrEspr*** **{,** ***AttrEspr*****}**
**`from`** ***Tabella*** **{,** ***Tabella*****}**
**[** **`where`** ***Condizione*** **]**

Le tre parti della query sono chiamate:

- clausola `select` / target list
- clausola `from`
- clausola `where`

# Interrogazioni SQL

La query effettua

- il prodotto cartesiano delle tabelle nella clausola `from`,
- considera solo le righe che soddisfano la condizione nella clausola `where` e
- per ogni riga valuta le espressioni nella `select`

**Sintassi completa:**

**`select` *AttrEspr* [[ `as` ] *Alias* ] {, *AttrEspr* [[ `as` ] *Alias* ] }**
**`from` *Tabella* [[ `as` ] *Alias* ] {, *Tabella* [[ `as` ] *Alias* ] }**
**[ `where` *Condizione* ]**

# Esempio: gestione degli esami universitari

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Corso**

| COD-CORSO | TITOLO | DOCENTE |
|---|---|---|
| 1 | matematica | Barozzi |
| 2 | informatica | Meo |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# Proiezione

```
SELECT Nome, Cdip
FROM STUDENTE
```

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

è una tabella con

- schema: gli attributi Nome e Cdip (grado <=)
- istanza: la restrizione delle tuple sugli attributi Nome e CDip (cardinalità <=)

| Nome | CDip |
|---|---|
| Carlo | Inf |
| Alex | Inf |
| Antonio | Log |

# Proiezione

```
SELECT *
FROM STUDENTE
```

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Prende tutte le colonne della tabella STUDENTE**

# Duplicati

- In SQL, le tabelle prodotte dalle interrogazioni possono contenere più righe identiche tra loro

Studente

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

```
select CDip
from Studente
```

| CDip |
|---|
| Inf |
| Inf |
| Log |

- I duplicati possono essere rimossi usando la parola chiave distinct

```
select distinct CDip
from Studente
```

| CDip |
|---|
| Inf |
| Log |

# Selezione

```
SELECT *

FROM STUDENTE

WHERE Nome='Alex'
```

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

È una tabella con

- schema: lo stesso schema di STUDENTE (grado =)
- istanza: le tuple di STUDENTE che soddisfano il predicato di selezione (cardinalità <=)

| Matr | Nome | Città | CDip |
|---|---|---|---|
| 415 | Alex | Torino | Inf |

# Sintassi del predicato di selezione

**Espressione booleana di predicati semplici**

operazioni booleane:

- AND (P1 AND P2)
- OR (P1 OR P2)
- NOT (P1)

predicati semplici:

- TRUE, FALSE
- termine  comparatore termine
- attributo BETWEEN valore1 AND valore2

comparatore:

- =, <>, <, <=, >, >=

termine:

- costante, attributo
- espressione aritmetica di costanti e attributi

# Sintassi della clausola `where`

Espressione booleana di predicati semplici (come in algebra)

Estrarre gli studenti di informatica originari di Bologna:

```
select *
from Studente
where CDip = 'Inf' and Città = 'Bologna'
```

Studente

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

Estrarre gli studenti originari di Bologna o di Torino:

```
select *
from Studente
where Città = 'Bologna' or Città = 'Torino'
```

Attenzione: estrarre gli studenti originari di Bologna e originari di Torino

# Esempio di selezione

```
SELECT *

FROM STUDENTE

WHERE (Città='Torino')
OR ((Città='Roma') AND
NOT (CDip=’Log'))
```

| MATR | NOME | CITTA’ | C-DIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

# Espressioni booleane

Estrarre gli studenti originari di Roma che frequentano il corso in Informatica o in Logistica:

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

```
select *
from Studente
where Città = 'Roma' and
      (CDip = 'Inf' or
       CDip = 'Log')
```

| Matr | Nome | Città | CDip |
|---|---|---|---|
| 702 | Antonio | Roma | Log |

# Gestione dei valori nulli

I valori nulli rappresentano tre diverse situazioni:

- un valore non è applicabile
- un valore è applicabile ma sconosciuto
- non si sa se il valore è applicabile o meno

Per fare una verifica sui valori nulli: *Attributo* **`is [ not ] null`**

```
select *
from Studente
where CDip = 'Inf' or CDip <> 'Inf'
```

**è equivalente a:**

```
select *
from Studente
where CDip is not null
```

# Selezione e proiezione

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

- Estrarre il nome degli studenti iscritti al corso in informatica?

```
SELECT Nome
FROM STUDENTE
WHERE CDip= 'Inf'
```

| NOME |
|---|
| Carlo |
| Alex |

# Selezione e proiezione

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

- Nome degli studenti di Logistica non di Milano

```
SELECT NOME
FROM STUDENTE
WHERE CDip= 'Log'  AND Città<>' Milano'
```

| NOME |
|---|
| Antonio |

# Prodotto cartesiano

R , S

è una tabella (priva di nome) con

- schema: gli attributi di R e S
  (grado(RxS)= grado(R)+grado(S))

- istanza: tutte le possibili coppie di tuple di R e S
  (card(RxS)=card(R)*card(S))

# Supponiamo di voler trovare, per ogni esame

il codice del corso si trova in ESAME ma il titolo si trova in CORSO!

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

**Corso**

| COD-CORSO | TITOLO | DOCENTE |
|---|---|---|
| 1 | matematica | Barozzi |
| 2 | informatica | Meo |

# Prodotto cartesiano con condizione →Join

```
SELECT *
FROM STUDENTE , ESAME
WHERE STUDENTE.Matr=ESAME.Matr
```

Attributi omonimi sono resi non ambigui usando la notazione “puntata”:

ESAME.Matr

STUDENTE.Matr

# Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Informatica” che hanno preso almeno un 30

```
select Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
  and CDip = 'Inf' and Voto = 30
```

| NOME |
| --- |
| Carlo |

**Studente**

| MATR | NOME | CITTA’ | CDIP |
| --- | --- | --- | --- |
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
| --- | --- | --- | --- |
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Informatica” che hanno preso almeno un 30

```
select distinct Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
  and CDip = 'Inf' and Voto = 30
```

| NOME |
|---|
| Carlo |

**Studente**

| MATR | NOME | CITTA’ | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di "Informatica" che hanno preso SEMPRE 30

Occorrono altri costrutti

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# Interrogazione semplice con tre tabelle

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

**Corso**

| COD-CORSO | TITOLO | DOCENTE |
|---|---|---|
| 1 | matematica | Barozzi |
| 2 | informatica | Meo |

Estrarre il nome degli studenti di "Matematica" che hanno preso qualche 30

```
select Nome
from Studente, Esame, Corso
where  Studente.Matr = Esame.Matr
  and Corso.CodCorso = Esame.CodCorso
  and Titolo = 'Matematica' and Voto = 30
```

# Join in SQL

SQL ha una sintassi per i join, li rappresenta esplicitamente nella clausola `from`:

**`select`** ***AttrEspr*** **{,** ***AttrEspr*****}**

**`from`** ***Tabella*** **{ [***TipoJoin***]** **`join`** ***Tabella*** **`on`** ***Condizioni*** **}**

**[** **`where`** ***AltreCondizioni*** **]**

TipoJoin può essere: inner, right [ outer ], left [outer] or full [ outer ]

# Prodotto cartesiano con condizione →Join

```
SELECT *
FROM STUDENTE , ESAME
WHERE STUDENTE.Matr=ESAME.Matr
```

Si può scrivere anche con l'operatore derivato:

```
SELECT *
FROM STUDENTE JOIN ESAME ON STUDENTE.Matr=ESAME.Matr
```

# Join

```
FROM STUDENTE JOIN ESAME ON STUDENTE.Matr=ESAME.Matr
```

produce una tabella con

- schema: la concatenazione degli schemi di STUDENTE e ESAME
- istanza: le tuple ottenute concatenando quelle tuple di STUDENTE e di ESAME che soddisfano il predicato

| STUDENTE. Matr | Nome | Città | CDip | ESAME. Matr | Cod-Corso | Data | Voto |
|---|---|---|---|---|---|---|---|
| 123 | Carlo | Bologna | Inf | 123 | 1 | 7-9-97 | 30 |
| 123 | Carlo | Bologna | Inf | 123 | 2 | 8-1-98 | 28 |
| 702 | Antonio | Roma | Log | 702 | 2 | 7-9-97 | 20 |

# Sintassi del predicato di join

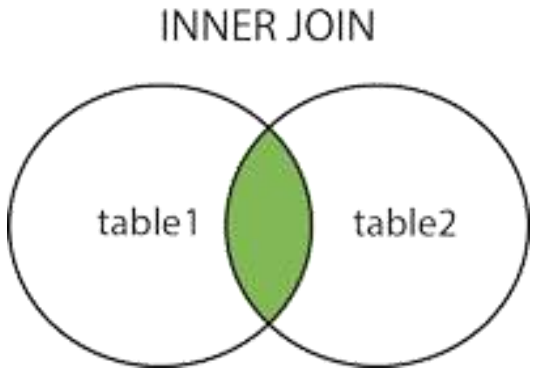
Espressione congiuntiva di predicati semplici:

ATTR1 comp ATTR2

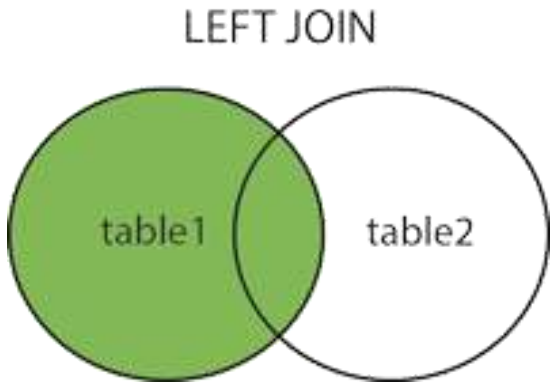
dove ATTR1 appartiene a TAB1
ATTR2 appartiene a TAB2
comp: =, <>, <, <=, >, >=

# LEFT / RIGHT / FULL JOIN

- Il JOIN specificato finora è un “INNER” join e seleziona i record che trovano un match in entrambe le tabelle




- Il LEFT JOIN restituisce tutti i record della tabella di sinistra e – se esiste – il match per la tabella di destra; se non ci sono match per la parte di destra restituisce NULL



# Esempio di LEFT JOIN

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

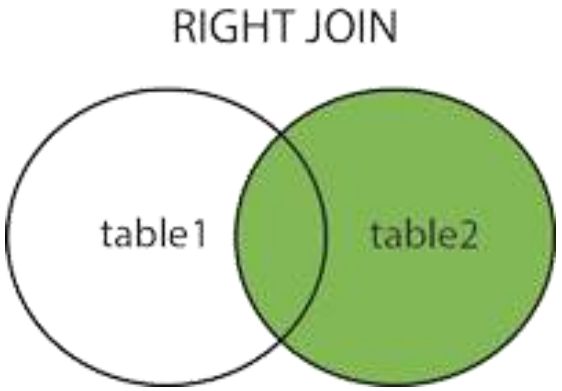
| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

```
SELECT *
FROM STUDENTE LEFT JOIN ESAME ON STUDENTE.Matr=ESAME.Matr
```

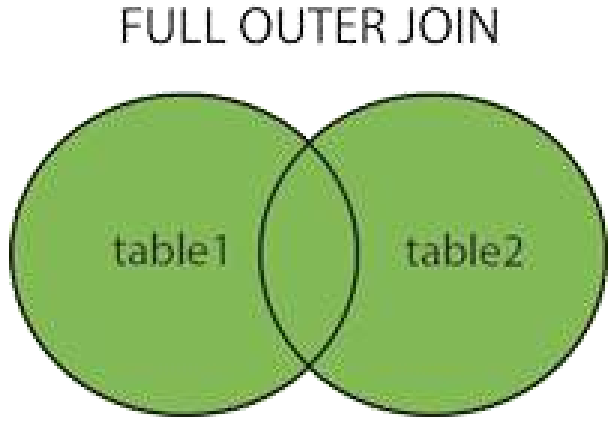
| STUDENTE. Matr | Nome | Città | CDip | ESAME. Matr | Cod-Corso | Data | Voto |
|---|---|---|---|---|---|---|---|
| 123 | Carlo | Bologna | Inf | 123 | 1 | 7-9-97 | 30 |
| 123 | Carlo | Bologna | Inf | 123 | 2 | 8-1-98 | 28 |
| 702 | Antonio | Roma | Log | 702 | 2 | 7-9-97 | 20 |
| 415 | Alex | Torino | Inf | NULL | NULL | NULL | NULL |

# LEFT / RIGHT / FULL JOIN

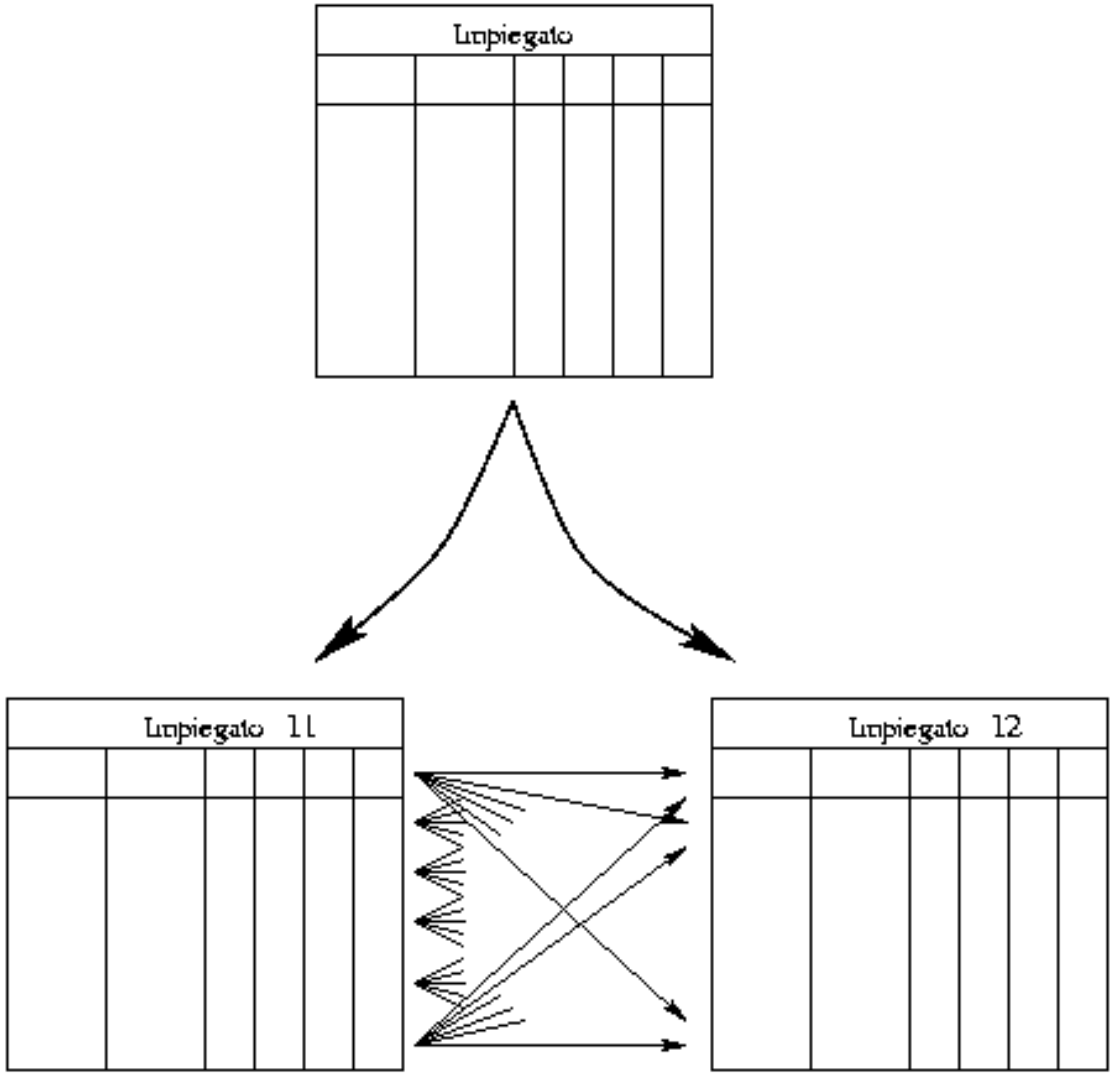
- Il RIGHT JOIN restituisce tutti i record della tabella di destra  e – se esiste – il match per la tabella di sinistra




- Il FULL OUTER JOIN restituisce tutte le righe di entrambe le tabelle, indipendentemente dalla presenza o meno di valori corrispondenti nelle tabelle.
- È possibile inserire una clausola WHERE in un full outer join per ottenere solo le righe per le quali non esistono corrispondenze nelle tabelle.



# SELF JOIN - Variabili in SQL



# Interrogazione semplice con variabili relazionali

**Impiegato**

| Matr | Nome | DataAss | Salario | MatrMgr |
|---|---|---|---|---|
| 1 | Piero | 1-1-95 | 3 M | 2 |
| 2 | Giorgio | 1-1-97 | 2,5 M | null |
| 3 | Giovanni | 1-7-96 | 2 M | 2 |

Chi sono i dipendenti di Giorgio?

# Chi sono i dipendenti di Giorgio?

```
SELECT X.Nome,X.MatrMgr,Y.Matr,Y.Nome

FROM Impiegato AS X, Impiegato AS Y

WHERE X.MatrMgr = Y.Matr

  AND Y.Nome = 'Giorgio'
```

| X.Nome | X.MatrMgr | Y.Matr | Y.Nome |
|---|---|---|---|
| Piero | 2 | 2 | Giorgio |
| Giovanni | 2 | 2 | Giorgio |

# Ordinamento

La clausola `order by`, che compare in coda all'interrogazione, ordina le righe del risultato

Sintassi:

```
order by AttributoOrdinamento [ asc | desc ]
                {, AttributoOrdinamento [ asc | desc ] }
```

Le condizioni di ordinamento vengono valutate in ordine

- a pari valore del primo attributo, si considera l'ordinamento sul secondo, e così via

# Query con ordinamento

ORDINE (CodOrd, CodCli, Data, Importo)
DETTAGLIO (CodOrd, CodProd, Qta)
CLIENTE (CodCli, Nome, Cognome, Città)

```
select  *
from Ordine
where Importo > 100.000
order by Data
```

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 1 | 3 | 1-6-97 | 50.000.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 2 | 4 | 3-8-97 | 8.000.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |

# order by CodCli

```
select  *
from Ordine
where Importo > 100.000
order by CodCli
```

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 1 | 3 | 1-6-97 | 50.000.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 2 | 4 | 3-8-97 | 27.000.000 |

# order by CodCli asc, Data desc

```
select  *
from Ordine
where Importo > 100.000
order by CodCli asc,
         Data desc
```

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| **5** | **1** | **1-8-97** | 1.500.000 |
| **4** | **1** | **1-7-97** | 12.000.000 |
| **6** | **3** | **3-9-97** | 5.500.000 |
| **3** | **3** | **1-9-97** | 1.500.000 |
| **1** | **3** | **1-6-97** | 50.000.000 |
| **2** | **4** | **3-8-97** | 27.000.000 |

# Funzioni aggregate

Il risultato di una query con funzioni aggregate dipende dalla valutazione del contenuto di un insieme di righe

Cinque operatori aggregati:

- `count` cardinalità
- `sum` sommatoria
- `max` massimo
- `min` minimo
- `avg` media

# Operatore `count`

`count` restituisce il numero di righe o valori distinti; sintassi:

```
count(< * | [ distinct | all ] ListaAttributi >)
```

Esempi:

- Estrarre il numero di ordini:

```
select count(*)
from Ordine
```

# Operatore `count`

- Estrarre il numero di valori distinti dell'attributo CodCli per tutte le righe di Ordine:

```
select count(distinct CodCli)
from Ordine
```

- Estrarre il numero di righe di Ordine che posseggono un valore non nullo per l'attributo CodCli:

```
select count(all CodCli)
from Ordine
```

# `sum`, `max`, `min`, `avg`

Sintassi:

< `sum` | `max` | `min` | `avg` > ([ `distinct` | `all` ] *AttrEspr* )

L'opzione `distinct` considera una sola volta ciascun valore

- utile solo per le funzioni `sum` e `avg`

L'opzione `all` considera tutti i valori diversi da *null*

# Query con massimo

- Estrarre l'importo massimo degli ordini

```
select max(Importo) as MaxImp
from Ordine
```

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 1 | 3 | 1-6-97 | 50.000.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| ... | ... | ... | ... |

| MaxImp |
|---|
| 50.000.000 |

# Query con sommatoria

- Estrarre la somma degli importi degli ordini relativi al cliente numero 1

```
select sum(Importo) as SommaImp
from Ordine
where CodCliente = 1
```

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 1 | 3 | 1-6-97 | 50.000.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 2 | 4 | 3-8-97 | 8.000.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |

| SommaImp |
|---|
| 13.500.000 |

# Funzioni aggregate con join

ORDINE (CodOrd, CodCli, Data, Importo)
DETTAGLIO (CodOrd, CodProd, Qta)
CLIENTE (CodCli, Nome, Cognome, Città)

Estrarre l'ordine (importo) massimo tra quelli contenenti il prodotto con codice 'ABC' :

```
select max(Importo) as MaxImportoABC
from Ordine, Dettaglio
where Ordine.CodOrd = Dettaglio.CodOrd and
CodProd = 'ABC'
```

# Funzioni aggregate e target list

Estrarre il codice dell'ordine / la data / altro attributo avente importo massimo:

Query scorretta:

```
select Data, max(Importo)
from Ordine, Dettaglio
where Ordine.CodOrd = Dettaglio.CodOrd and CodProd = 'ABC'
```

La data di quale ordine? La target list deve essere omogenea

# Funzioni aggregate e target list

Date in cui l'importo è massimo (restituire data e importo)

```
select Data, Importo
from Ordine, Dettaglio
where Ordine.CodOrd = Dettaglio.CodOrd and CodProd = 'ABC'
  and Importo   = (select max(Importo)
                   from Ordine)
```

# Funzioni aggregate e target list

Estrarre il massimo e il minimo importo degli ordini:

```
select max(Importo) as MaxImp,
       min(Importo) as MinImp
from Ordine
```

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 1 | 3 | 1-6-97 | 50.000.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 2 | 4 | 3-8-97 | 8.000.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |

| MaxImp | MinImp |
|---|---|
| 50.000.000 | 1.500.000 |

# Aggregati

Trovare l'importo medio degli ordini:

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 1 | 3 | 1-6-97 | 50.000.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| ... | ... | ... | ... |

```
select avg(Importo)
from Ordine
```

E se volessimo trovare l'importo medio **per ogni cliente**?

# Query con raggruppamento

Nelle interrogazioni si possono applicare gli operatori aggregati a sottoinsiemi di righe

Si aggiungono le clausole

- **`group by`** (raggruppamento)
- **`having`** (selezione dei gruppi)

```
select …
from …
where …
group by …
having …
```

# Query con raggruppamento

Estrarre la somma degli importi degli ordini successivi al 10-6-97 per quei clienti che hanno emesso almeno 2 ordini

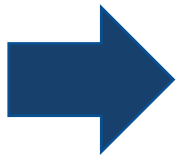
| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 1 | 3 | 1-6-97 | 50.000.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 2 | 4 | 3-8-97 | 8.000.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
```

# Passo 1: Valutazione `where`

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
```

| CODORD | CODCLI | DATA | IMPORTO |
|---|---|---|---|
| 1 | 3 | 1-6-97 | 50.000.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 2 | 4 | 3-8-97 | 8.000.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |



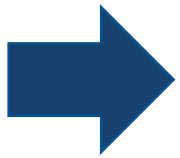
| CodOrd | CodCli | Data | Importo |
|---|---|---|---|
| 2 | 4 | 3-8-97 | 8.000.000 |
| 3 | 3 | 1-9-97 | 5.500.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 27.000.000 |

# Passo 2 : Raggruppamento

Si valuta la clausola **group by**

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
```

| CodOrd | CodCli | Data | Importo |
|---|---|---|---|
| 2 | 4 | 3-8-97 | 8.000.000 |
| 3 | 3 | 1-9-97 | 5.500.000 |
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 27.000.000 |



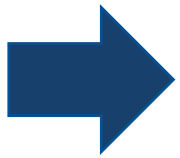
| CodOrd | CodCli | Data | Importo |
|---|---|---|---|
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |
| 2 | 4 | 3-8-97 | 8.000.000 |

# Passo 3 : Calcolo degli aggregati

Si calcolano `sum(Importo)` e `count(*)` per ciascun gruppo

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
```

| CodOrd | CodCli | Data | Importo |
|---|---|---|---|
| 4 | 1 | 1-7-97 | 12.000.000 |
| 5 | 1 | 1-8-97 | 1.500.000 |
| 3 | 3 | 1-9-97 | 1.500.000 |
| 6 | 3 | 3-9-97 | 5.500.000 |
| 2 | 4 | 3-8-97 | 8.000.000 |



| CodCli | sum(Importo) | count(*) |
|---|---|---|
| 1 | 13.500.000 | 2 |
| 3 | 7.000.000 | 2 |
| 4 | 8.000.000 | 1 |

# Passo 4 : Estrazione dei gruppi

Si valuta il predicato `count(*) >= 2`

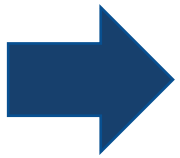
```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
```

| CodCli | sum(Importo) | count(*) |
|---|---|---|
| 1 | 13.500.000 | 2 |
| 3 | 7.000.000 | 2 |
| 4 | 8.000.000 | 1 |

# Passo 5 : Produzione del risultato (esecuzione della clausola Select)

```
select CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
```

| CodCli | sum (Importo) | count(*) |
|---|---|---|
| 1 | 13.500.000 | 2 |
| 3 | 7.000.000 | 2 |



| CodCli | sum(Importo) |
|---|---|
| 1 | 13.500.000 |
| 3 | 7.000.000 |

# Query con group by e target list

Query scorretta:

```
select Importo
from Ordine
group by CodCli
```

Query scorretta:

```
select O.CodCli, count(*), C.Città
from Ordine O join Cliente C
     on (O.CodCli = C.CodCli)
group by O.CodCli
```

Query corretta:

```
select O.CodCli, count(*), C.Città
from Ordine O join Cliente C
     on (O.CodCli = C.CodCli)
group by O.CodCli, C.Città
```

# `where` o `having`?

Soltanto i predicati che richiedono la valutazione di funzioni aggregate dovrebbero comparire nell'argomento della clausola `having`

Estrarre i dipartimenti in cui lo stipendio medio degli impiegati che lavorano nell'ufficio 20 è maggiore di 25:

```
select Dipart
from Impiegato
where Ufficio = '20'
group by Dipart
having avg(Stipendio) > 25
```

# Raggruppamento e ordinamento

Estrarre la somma degli importi degli ordini successivi al 10-6-97 per quei clienti che hanno emesso almeno 2 ordini, in ordine decrescente di codice cliente

```
select  CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
order by CodCli desc
```

| CodCli | sum(Importo) |
|---|---|
| 3 | 32.500.000 |
| 1 | 7.000.000 |

# Raggruppamento e ordinamento

Estrarre la somma degli importi degli ordini successivi al 10-6-97 per quei clienti che hanno emesso almeno 2 ordini, in ordine decrescente di somma di importo

```
select  CodCli, sum(Importo)
from Ordine
where Data > 10-6-97
group by CodCli
having count(*) >= 2
order by sum(Importo) desc
```

| CodCli | sum(Importo) |
|---|---|
| 3 | 7.000.000 |
| 1 | 13.500.000 |

# Doppio raggruppamento

ORDINE (CodOrd, CodCli, Data, Importo)
DETTAGLIO (CodOrd, CodProd, Qta)
CLIENTE (CodCli, Nome, Cognome, Città)

Estrarre la somma delle quantità dei dettagli degli ordini emessi da ciascun cliente per ciascun prodotto, purché la somma superi 50

```
select CodCli, CodProd, sum(Qta)
from Ordine as O, Dettaglio as D
Where O.CodOrd = D.CodOrd
group by CodCli, CodProd
having sum(Qta) > 50
```

# Situazione dopo il join e il raggruppamento

| Ordine | | Dettaglio | | |
|---|---|---|---|---|
| **CodCli** | Ordine.<br>CodOrd | Dettaglio.<br>CodOrd | **CodProd** | Qta |
| **1** | 3 | 3 | **1** | **30** |
| **1** | 4 | 4 | **1** | **20** |
| **1** | 3 | 3 | **2** | **30** |
| **1** | 5 | 5 | **2** | **10** |
| **2** | 3 | 3 | **1** | **60** |
| **3** | 1 | 1 | **1** | **40** |
| **3** | 2 | 2 | **1** | **30** |
| **3** | 6 | 6 | **1** | **25** |

gruppo 1,1

gruppo 1,2

gruppo 2,1

gruppo 3,1

# Estrazione del risultato

Si valuta la funzione aggregata `sum(Qta)` e il predicato `having`

| CodCli | CodProd | sum(Qta) |
|---|---|---|
| ~~1~~ | ~~1~~ | ~~50~~ |
| ~~1~~ | ~~2~~ | ~~40~~ |
| 2 | 1 | 60 |
| 3 | 1 | 95 |

# Esame 7 settembre 2018

AUTOBUS(TARGA, ANNOIMMATRIC, MODELLO, MARCA, NO_POSTI)

CONDUCENTE(MATRCOND, NOMECONDUCENTE, TELEFONO)

CORSA(IDCORSA, CITTÀPARTENZA, CITTÀARRIVO, ORAPARTENZA, TEMPOSTIMATO)

TURNO (IDCORSA, DATA, MATRCOND, BUS)

# Query 3

Estrarre, per ogni autista che ha almeno 300 giornate di servizio, il numero di autobus distinti che ha guidato. (3 p.)

```
select MatrCond, count( distinct Bus ) as NumeroBusDistintiGuidati
from Turno
group by MatrCond
having count( distinct Data ) >= 300
```

# Query binarie (set queries)

Costruite concatenando due query SQL tramite operatori insiemistici

***SelectSQL*** **{ < `union` | `intersect` | `except` > [ `all` ]** ***SelectSQL*** **}**

| | |
|---|---|
| union | unione |
| intersect | intersezione |
| except (minus) | differenza |

Si eliminano i duplicati, a meno che non venga usata l'opzione all

# Unione

Estrarre i codici degli ordini i cui importi superano 500 euro oppure in cui qualche prodotto è presente con quantità superiore a 1000

```
select CodOrd
from Ordine
where Importo > 500
      union
select CodOrd
from Dettaglio
where Qta > 1000
```

ORDINE (CodOrd, CodCli, Data, **Importo**)
DETTAGLIO (CodOrd, CodProd, **Qta**)
CLIENTE (CodCli, Nome, Cognome, Città)

# Nome degli attributi nel risultato

```
select Padre
from Paternita
union
select Madre
from Maternita
```

| PADRE | FIGLIO |
|---|---|
| Luigi | Giorgio |
| Stefano | Giovanni |

| MADRE | FIGLIO |
|---|---|
| Anna | Giorgio |
| Paola | Giovanni |

Quali nomi per gli attributi del risultato?

- Nessuno
- Quelli del primo operando
- …

# Notazione posizionale

```
select Padre, Figlio
from Paternita
union
select Figlio, Madre
from Maternita
```

```
select Padre, Figlio
from Paternita
union
select Madre, Figlio
from Maternita
```

Sono interrogazioni diverse!

Esempio:

| PADRE | FIGLIO |
|---|---|
| Luigi | Giorgio |
| Stefano | Giovanni |

| MADRE | FIGLIO |
|---|---|
| Anna | Giorgio |
| Paola | Giovanni |

# Notazione posizionale

```
select Padre, Figlio
from Paternita
union
select Figlio, Madre
from Maternita
```

| | |
|---|---|
| Luigi | Giorgio |
| Stefano | Giovanni |
| Giorgio | Anna |
| Giovanni | Paola |

```
select Padre, Figlio
from Paternita
union
select Madre, Figlio
from Maternita
```

| | |
|---|---|
| Luigi | Giorgio |
| Stefano | Giovanni |
| Anna | Giorgio |
| Paola | Giovanni |

# Uso della parola chiave `all`

Estrarre i padri di persone con nome “Giorgio” o “Giovanni”, presentando due volte i padri che hanno due figli chiamati uno Giorgio e uno Giovanni.

```
select Padre
from Paternita
where Figlio = 'Giorgio'
      union all
select Padre
from Paternita
where Figlio = 'Giovanni'
```

# Differenza

Estrarre i codici degli ordini i cui importi superano 500 euro ma in cui nessun prodotto è presente con quantità superiore a 1000

```
select CodOrd
from Ordine
where Importo > 500
      except
select CodOrd
from Dettaglio
where Qta > 1000
```

**Può essere rappresentata usando una query nidificata**

# Intersezione

Estrarre i codici degli ordini i cui importi superano 500 euro e in cui qualche prodotto è presente con quantità superiore a 1000

```
select CodOrd
from Ordine
where Importo > 500
      intersect
select CodOrd
from Dettaglio
where Qta > 1000
```

**Può essere rappresentata usando una query nidificata**

# Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Informatica” che hanno preso SEMPRE 30

```
select Matr, Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
   and CDip = 'Inf' and Voto = 30
EXCEPT
select Matr, Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
   and CDip = 'Inf' and Voto < 30
```

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# Query nidificate

Nelle clausole **`where`** e **`having`** possono comparire predicati che:

- confrontano un attributo (o un'espressione sugli attributi) con il risultato di una query SQL; sintassi:

  *AttrExpr Operator* < **`any`** | **`all`** > *SelectSQL*

  - **`any`**: il predicato è vero se almeno una riga restituita dalla query *SelectSQL* soddisfa il confronto
  - **`all`**: il predicato è vero se tutte le righe restituite dalla query *SelectSQL* soddisfano il confronto
  - *Operator*: uno qualsiasi tra **`=`**, **`<>`**, **`<`**, **`<=`**, **`>`**, **`>=`**

La query che appare nella clausola **`where`** e nella clausola **`having`** è chiamata query nidificata

Nelle query nidificate posso usare variabili definite esternamente

# Uso di `any` e `all`

```
select CodOrd
from Ordine
where Importo > any
          select Importo
          from Ordine
```

```
select CodOrd
from Ordine
where Importo >= all
          select Importo
          from Ordine
```

| COD-ORD | IMPORTO |
|---|---|
| 1 | 50 |
| 2 | 300 |
| 3 | 90 |

| ANY | ALL |
|---|---|
| F | F |
| V | V |
| V | F |

# Query nidificate con any

Estrarre gli ordini di prodotti con un prezzo superiore a 100

```
select distinct CodOrd

from Dettaglio

where CodProd = any(select CodProd
                    from Prodotto
                    where Prezzo > 100)
```

Equivalente a (senza query nidificata)

```
select distinct CodOrd

from Dettaglio D, Prodotto P

where D.CodProd = P.CodProd
  and Prezzo > 100
```

ORDINE (CodOrd, CodCli, Data, Importo)

DETTAGLIO (CodOrd, CodProd, Qta)C

# Query nidificate con any

Estrarre i prodotti ordinati assieme al prodotto avente codice 'ABC'

- con una query nidificata:

```
select distinct CodProd
from Dettaglio
where CodProd<>'ABC' and CodOrd = any
   (select CodOrd
    from Dettaglio
    where CodProd = 'ABC')
```

DETTAGLIO (CodOrd, CodProd, Qta)

- senza query nidificata, a meno di duplicati:

```
select distinct D1.CodProd
from Dettaglio D1, Dettaglio D2
where D1.CodOrd = D2.CodOrd and
      D1.CodProd<>'ABC' and D2.CodProd='ABC'
```

# Negazione con query nidificate

Estrarre gli ordini che non contengono il prodotto 'ABC':

```
select distinct CodOrd
from Ordine
where CodOrd <> all (select CodOrd
                     from Dettaglio
                     where CodProd = 'ABC')
```

ORDINE (CodOrd, CodCli, Data, Importo)

DETTAGLIO (CodOrd, CodProd, Qta)C

# Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Informatica” che hanno preso SEMPRE 30

```
select Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
   and CDip = 'Inf' and Voto = 30
   and  Matr <> ALL (select Matr
                     from Esame
                     where Voto < 30)
```

**Studente**

| MATR | NOME | CITTA’ | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Informatica” che hanno preso SEMPRE 30

```
select Nome
from Studente
where CDip = 'Inf' and
    Matr <> ALL (select Matr
                 from Esame
                 where Voto < 30)
    and Matr = ANY (select Matr
                    from Esame
                    where Voto = 30)
```

**Studente**

| MATR | NOME | CITTA' | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# Query nidificate

*AttrExpr* < **`in`** | **`not in`** > *SelectSQL*

- **`in`**: il predicato è vero se almeno una riga restituita dalla query *SelectSQL* e' presente nell'espressione
- **`not in`**: il predicato è vero se nessuna riga restituita query e' presente nell'espressione

# Operatori **in** e **not in**

L'operatore **in** è equivalente a **= any**

```
select CodProd
from Dettaglio
where CodProd<>'ABC' and

      CodOrd in
         (select CodOrd
          from Dettaglio
          where CodProd = 'ABC')
```

```
select distinct CodProd
from Dettaglio
where CodProd<>'ABC' and
      CodOrd = any
         (select CodOrd
          from Dettaglio
          where CodProd = 'ABC')
```

# Operatori **in** e **not in**

L'operatore **not in** è equivalente a **<> all**

```
select distinct CodOrd
from Ordine
where CodOrd not in (select CodOrd
                     from Dettaglio
                     where CodProd = 'ABC')
```

```
select distinct CodOrd
from Ordine
where CodOrd <> all (select CodOrd
                     from Dettaglio
                     where CodProd = 'ABC')
```

# Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di “Informatica” che hanno preso SEMPRE 30

```
select Nome
from Studente, Esame
where Studente.Matr = Esame.Matr
   and CDip = 'Inf' and Voto = 30
   and Matr NOT IN (select Matr
                    from Esame
                    where Voto < 30)
```

**Studente**

| MATR | NOME | CITTA’ | CDIP |
|---|---|---|---|
| 123 | Carlo | Bologna | Inf |
| 415 | Alex | Torino | Inf |
| 702 | Antonio | Roma | Log |

**Esame**

| MATR | COD-CORSO | DATA | VOTO |
|---|---|---|---|
| 123 | 1 | 7-9-97 | 30 |
| 123 | 2 | 8-1-98 | 28 |
| 702 | 2 | 7-9-97 | 20 |

# `max` con query nidificata

Gli operatori aggregati `max` (e `min`) possono essere espressi tramite query nidificate

Estrarre l'ordine con il massimo importo

- Con una query nidificata, usando `max`:

```
select CodOrd
from Ordine
where Importo in (select max(Importo) from Ordine)
```

- con una query nidificata, usando all:

```
select CodOrd
from Ordine
where Importo >= all (select Importo from Ordine)
```

# Costruttore di tupla

Il confronto con la query nidificata può coinvolgere più di un attributo

Gli attributi devono essere racchiusi da un paio di parentesi tonde (costruttore di tupla)

Esempio: estrarre gli omonimi

```
select *

from Persona P

where (Nome,Cognome) in
      (select Nome, Cognome
       from Persona P1
       where P1.CodFisc <> P.CodFisc)
```

Persona(CodFisc, Cognome, Nome)

# Costruttore di tupla

Esempio: estrarre le persone che non hanno omonimi

```
select *

from Persona P

where (Nome,Cognome) not in
      (select Nome, Cognome
       from Persona P1
       where P1.CodFisc <> P.CodFisc)
```

# Esame 7 settembre 2018

AUTOBUS(TARGA, ANNOIMMATRIC, MODELLO, MARCA, NO_POSTI)

CONDUCENTE(MATRCOND, NOMECONDUCENTE, TELEFONO)

CORSA(IDCORSA, CITTÀPARTENZA, CITTÀARRIVO, ORAPARTENZA, TEMPOSTIMATO)

TURNO (IDCORSA, DATA, MATRCOND, BUS)

# Query 1

Estrarre la targa dei bus con almeno 50 posti di capienza che hanno viaggiato a luglio 2018 ma nel luglio 2018 non sono partiti da Milano. (3 p.)

```
select Targa
from Autobus join Turno on Bus = Targa
where No_Posti >= 50 and Data between 1/7/18 and 31/7/18
    and Targa not in  ( select Bus
            from Corsa natural join Turno
            where Data between 1/7/18 and 31/7/18 and CittàPartenza = “Milano” )
```

# Uso di `in` nelle modifiche

Aumentare di 5 euro l'importo di tutti gli ordini che comprendono il prodotto 456

```
update Ordine
   set Importo = Importo + 5
   where CodOrd in
      select CodOrd
      from Dettaglio
      where CodProd = '456'
```

# Uso di query nidificate  nelle modifiche

Assegnare a TotPezzi la somma delle quantità delle linee di un ordine

```
update Ordine O
  set TotPezzi =
    (select sum(Qta)
     from Dettaglio D
     where D.CodOrd = O.CodOrd)
```

# Viste

Offrono la "visione" di tabelle virtuali (schemi esterni)

Le viste possono essere usate per formulare query complesse

- Le viste decompongono il problema e producono una soluzione più leggibile

Le viste sono talvolta necessarie per esprimere alcune query:

- query che combinano e nidificano diversi operatori aggregati
- query che fanno un uso sofisticato dell'operatore di unione

Sintassi:

**`create view`** *NomeVista* [ (*ListaAttributi*) ] **`as`** *SelectSQL*

# Composizione delle viste con le query

- Vista:

```
create view OrdiniPrincipali as
    select *
    from Ordine
    where Importo > 10000
```

- Query:

```
select CodCli
from OrdiniPrincipali
```

- Composizione della vista con la query:

```
select CodCli
from Ordine
where Importo > 10000
```

# Viste e query

Estrarre il cliente che ha generato il massimo fatturato (senza usare le viste):

```
select CodCli
from Ordine
group by CodCli
having sum(Importo) >= all
            (select sum(Importo)
             from Ordine
             group by CodCli)
```

Non tutti i sistemi consentono di specificare query nidificate all'interno della clausola having

# Viste e query

Estrarre il cliente che ha generato il massimo fatturato (usando le viste):

```
create view CliFatt(CodCli, FattTotale) as
      select CodCli, sum(Importo)
      from Ordine
      group by CodCli
```

```
select CodCli
from CliFatt
where FattTotale = (select max(FattTotale)
                    from CliFatt)
```

# Viste e query

Estrarre il numero medio di ordini per cliente:

- Soluzione scorretta (SQL non permette di applicare gli operatori aggregati in cascata):

```
select avg(count(*))
from Ordine
group by CodCli
```

- Soluzione corretta (usando una vista):

```
create view CliOrd(CodCli,NumOrdini) as
select CodCli, count(*)
from Ordine
group by CodCli

select avg(NumOrdini)
from CliOrd
```

# Viste in cascata

```
create view ImpiegatoAmmin (Matr,Nome,Cognome,Stipendio) as
select Matr, Nome, Cognome, Stipendio
from Impiegato
where Dipart = 'Amministrazione' and Stipendio > 10


create view ImpiegatoAmminJunior as
select *
from ImpiegatoAmmin
where Stipendio < 50
```

# Esame 7 settembre 2018

AUTOBUS(TARGA, ANNOIMMATRIC, MODELLO, MARCA, NO_POSTI)

CONDUCENTE(MATRCOND, NOMECONDUCENTE, TELEFONO)

CORSA(IDCORSA, CITTÀPARTENZA, CITTÀARRIVO, ORAPARTENZA, TEMPOSTIMATO)

TURNO (IDCORSA, DATA, MATRCOND, BUS)

# Query 2

Estrarre targa e anno di immatricolazione dei bus che sono stati allocati il maggior numero di volte su tratte con arrivo a Roma e sono stati guidati almeno due volte da “Marco Rossi”. (4 p.)

```
create view BusGuidatiDueVolteDaMarcoRossi( Bus ) as (
    select Bus
    from Turno natural join Conducente
    where NomeConducente = “Mario Rossi”
    group by Bus
    having count(*) > 1
)
```

# Query 2

Estrarre targa e anno di immatricolazione dei bus che sono stati allocati il maggior numero di volte su tratte con arrivo a Roma e sono stati guidati almeno due volte da “Marco Rossi”. (4 p.)

```
create view NumeroTurniVersoRomaDeiBusGuidatiDaRossi( Targa, QteVolte ) as
(
   select Bus, count( * )
   from BusGuidatiDueVolteDaMarcoRossi natural join Turno natural join Corsa
   where CittàArrivo = Roma
   group by Bus
)
```

# Query 2

Estrarre targa e anno di immatricolazione dei bus che sono stati allocati il maggior numero di volte su tratte con arrivo a Roma e sono stati guidati almeno due volte da “Marco Rossi”. (4 p.)

```
select Targa, AnnoImmatricolazione
from Autobus
where Targa in      ( select Targa
                      from NumeroTurniVersoRomaDeiBusGuidatiDaRossi
                      where QteVolte   =   (    select max( QteVolte )
                                                from NumeroTurniVersoRomaDeiBusGuidatiDaRossi )  )
```

# Esame Gennaio 2018

GIORNALISTA (IdGiornalista, Cognome, Nome, DataNascita, Nazionalità)

GIORNALE (NomeGiornale, Direttore, CittàSede, Editore)

ARTICOLO (NomeGiornale, Data, NumeroArticolo, Titolo, Autore, NumeroParole)

COMMENTO (IdCommento, NomeGiornale, Data, NumeroArticolo, NicknameUtente, Testo)

1. Trovare gli articoli scritti da giornalisti che non hanno mai collaborato con giornali editi da RCS. (3 punti)

2. Trovare i id, cognome e nome dei giornalisti italiani che hanno collaborato con un solo giornale. (3.5 punti)

3. Trovare il giornale su cui sono stati pubblicati più articoli nel 2016. (3.5 punti)

# Esame Giugno 2018

BREVETTO (IdBrevetto, Titolo, Descrizione, Categoria, Data)

REGIONE (IdRegione, NomeRegione, Nazione)

INVENTORE (IdInventore, Cognome, Nome, DataNascita, IdRegione)

BREVETTOINVENTORE (IdBrevetto, IdInventore)

1. Trovare id, cognome, nome e numero di brevetti degli inventori italiani che non hanno mai ottenuto brevetti la cui descrizione contiene la stringa 'carbon nanofiber'. (3 punti)

2. Trovare tutte le coppie (id inventore, data del brevetto) riferite ai soli brevetti con almeno due inventori. Restituire i risultati ordinati in ordine crescente per id dell'inventore e quindi per data del brevetto. (3 punti)

3. Trovare le coppie di regioni distinte con il più alto numero di co-invenzioni. Un brevetto rappresenta una co-invenzione tra la regione r1 e la regione r2 se ha almeno un inventore proveniente da r1 e almeno un inventore proveniente da r2. (4 punti)

# Link

- SQLZOO: http://sqlzoo.net/
- MySQL / PostgreSQL / MS Access / … oppure

  SQLite:

  - SQLiteStudio portable: https://sqlitestudio.pl
  - Sample database: http://www.sqlitetutorial.net/sqlite-sample-database/
  - Tutorial per utilizzare SQLite in Python: https://softpython.readthedocs.io/it/latest/exercises/database/database-solution.html