SQL

SQL

- Il nome sta per *Structured Query Language*
- 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

Definizione di tabelle

- Una tabella SQL consiste di:
  - un insieme ordinato di attributi
  - un insieme di vincoli (eventualmente vuoto)
- Comando `create table`
  - definisce lo schema di una relazione, creandone un'istanza vuota

```
create table Studente
(  Matr       character(6) primary key,
   Nome       varchar(30) not null,
   Città      varchar(20),
   CDip       char(3) );
```

Vincoli intra-relazionali

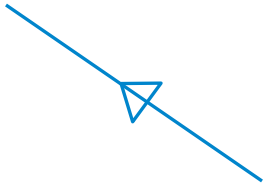
- I vincoli sono condizioni che devono essere verificate da ogni istanza della base di dati
- I vincoli intra-relazionali coinvolgono una sola relazione (distinguibili ulteriormente a livello di tupla o di tabella)
  - `not null` (su un solo attributo; a livello di tupla)
  - `unique`: permette la definizione di chiavi candidate (opera quindi a livello di tabella); sintassi:
    - per un solo attributo:
      `unique,` dopo il dominio
    - per diversi attributi:
      `unique(` *Attributo* {, *Attributo* })
  - `primary key`: definisce la chiave primaria (una volta per ogni tabella; implica *not null*); sintassi come per `unique`
  - `check`: può rappresentare vincoli di ogni tipo

Integrità referenziale

- Esprime un legame gerarchico (padre-figlio) fra tabelle
- Alcuni attributi della tabella figlio sono definiti FOREIGN KEY
- I valori contenuti nella FOREIGN KEY devono essere sempre presenti nella tabella padre

Una istanza scorretta

Matr	Nome	Città	CDip
123			
415			
702			



Esame

Matr	Cod Corso	Data	Voto
123	1	7-9-97	30
123	2	8-1-98	28
123	2	1-8-97	28
702	2	7-9-97	20
702	1	NULL	NULL
714	1	7-9-97	28

viola la chiave

viola il NULL

viola la integrità referenziale

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`
- 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

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

Proiezione

SELECT Nome, Cdip

FROM STUDENTE

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

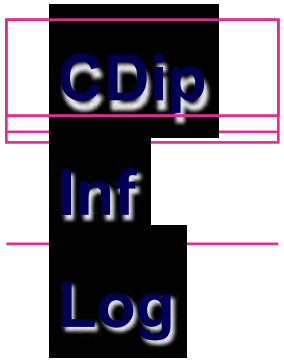
```
SELECT *
FROM STUDENTE
```

Prende tutte le colonne della tabella STUDENTE

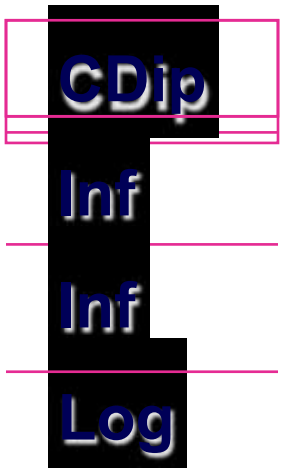
Duplicati

- In SQL, le tabelle prodotte dalle interrogazioni possono contenere più righe identiche tra loro
- I duplicati possono essere rimossi usando la parola chiave `distinct`

```
Select distinct CDip
from Studente
```




```
select CDip
from Studente
```



Selezione

```
SELECT *
FROM STUDENTE
WHERE Nome= 'Alex'
```

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

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'
```

- 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

Espressioni booleane

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

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

- Risultato:

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
```

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

Selezione e proiezione

Matr	Nome	Città	CDip
123	Carlo	Bologna	Inf
415	Alex	Torino	Inf
702	Antonio	Roma	Log

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

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

NOME
Carlo
Alex

Selezione e proiezione

Matr	Nome	Città	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

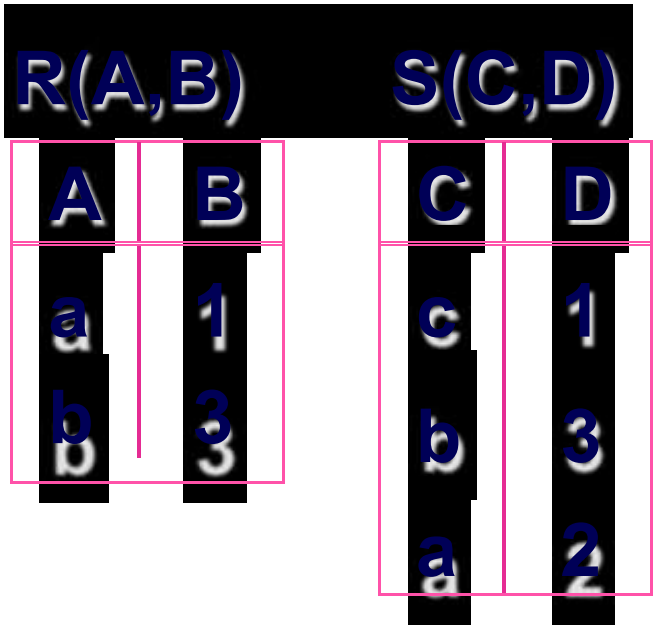
  (grado(RxS)= grado(R)+grado(S))

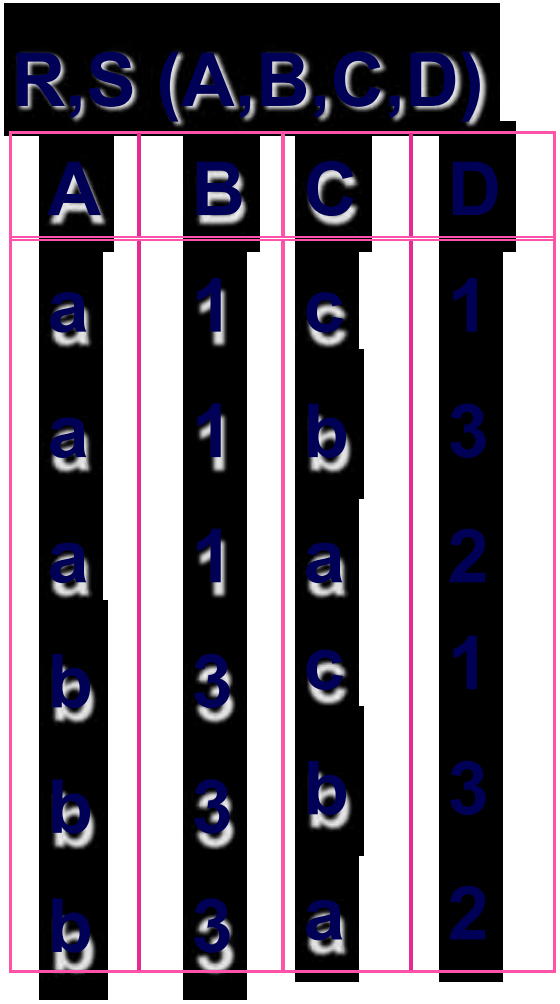
- istanza :

  tutte le possibili coppie di tuple di R e S

  (card(RxS)=card(R)*card(S))

Esempio





Select *

FROM R,S

Prodotto cartesiano con condizione → Join

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

Join

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

è equivalente alla seguente espressione (operatore derivato):

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

attributi omonimi sono resi non ambigui usando la notazione "puntata":

```
ESAME.Matr
STUDENTE.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 diSTUDENTE 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

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

Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di "Logistica" che hanno preso almeno un 30

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

Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di "Logistica" che hanno preso almeno un 30

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

NOME
Carlo

Interrogazione semplice con due tabelle

Estrarre il nome degli studenti di "Logistica" che hanno preso sempre 30

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

NOME
Carlo

Interrogazione semplice con tre tabelle

- Estrarre il nome degli studenti di “Matematica” che hanno preso 30

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

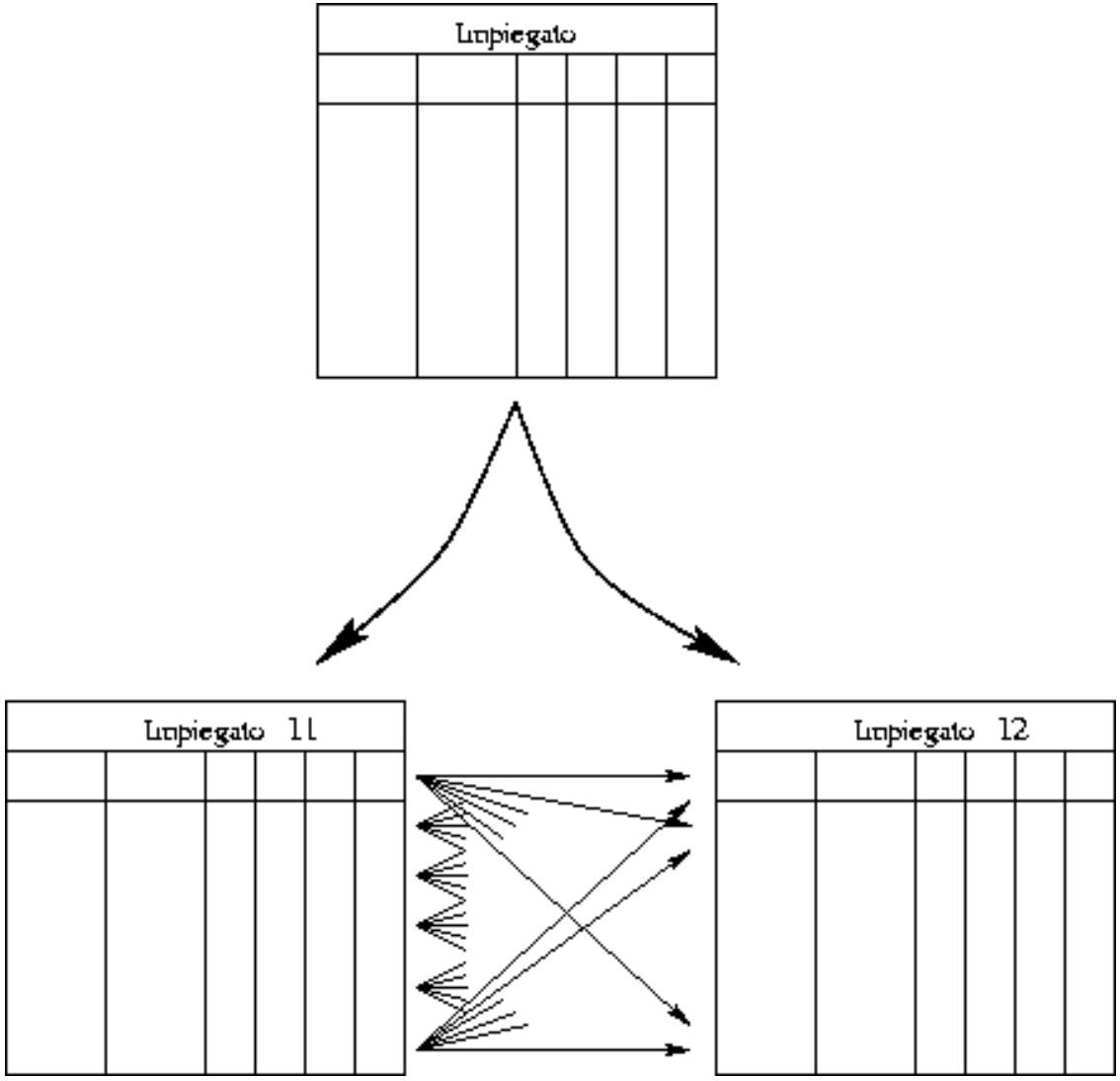
Join in SQL

- 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`] oppure `full` [`outer`], consentendo la rappresentazione dei join esterni

Variabili in SQL



Interrogazione semplice con variabili relazionali

Chi sono i dipendenti di Giorgio?

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?

```
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

```
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

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

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* `>)`

- Estrarre il numero di ordini:

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

- 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
```

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
```

SommaImp
13.500.000

Funzioni aggregate con join

- Estrarre l'ordine 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

- 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
- Estrarre il massimo e il minimo importo degli ordini:

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

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

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

Passo 1: Valutazione `where`

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`**

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

CodCli	sum(Importo)	count(*)
1	13.500.000	2
3	32.500.000	2
4	5.000.000	1

Passo 4 : Estrazione dei gruppi

- si valuta il predicato **`count(*) >= 2`**

CodCli	sum (Importo)	count(*)
1	13.500.000	2
3	32.500.000	2
4	5.000.000	1

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

CodCli	sum(Importo)
1	13.500.000
3	32.500.000

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
```

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 2 desc
```

Risultato dopo la clausola di ordinamento

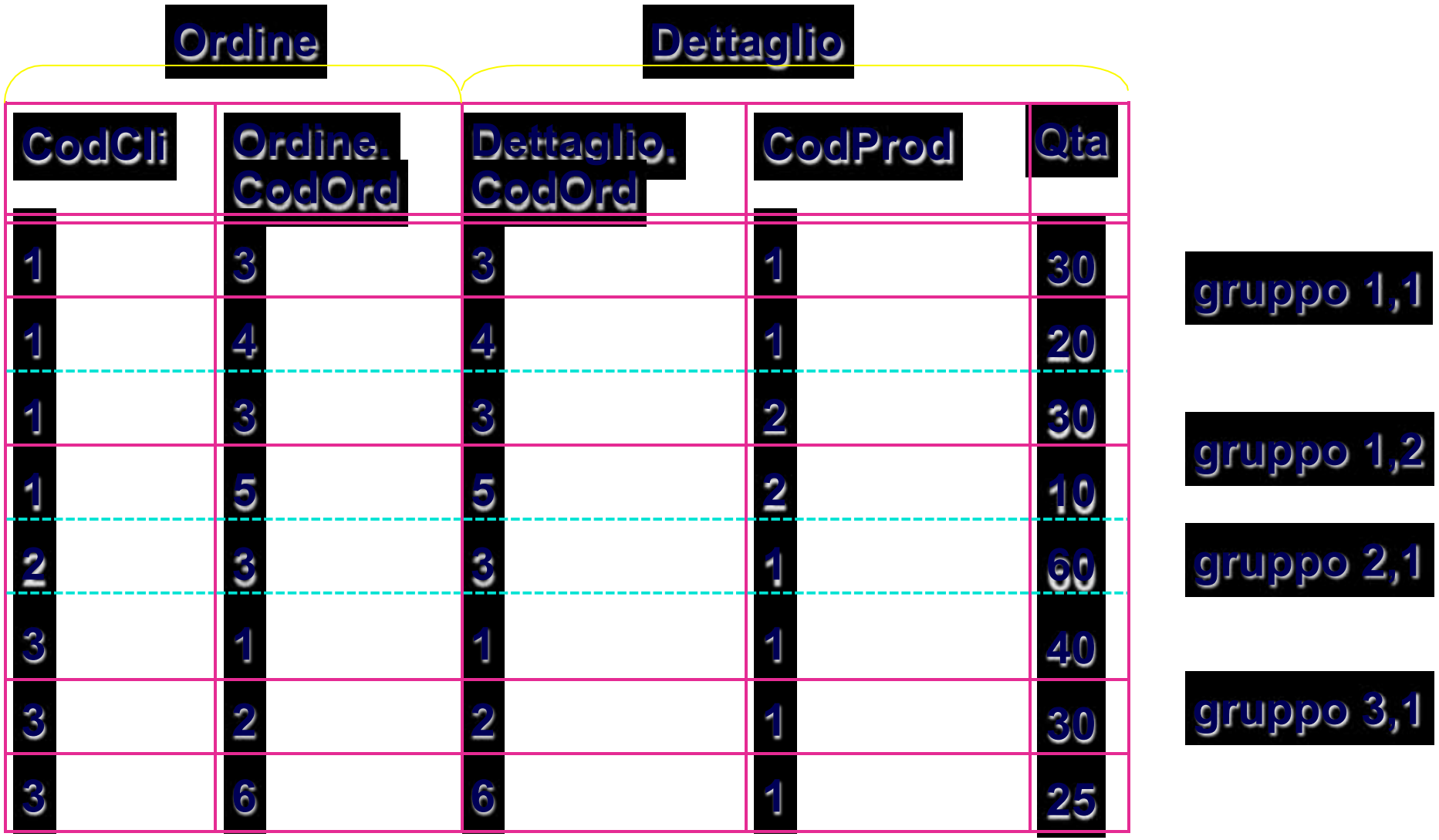
CodCli	sum(Importo)
3	32.500.000
1	13.500.000

Doppio raggruppamento

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

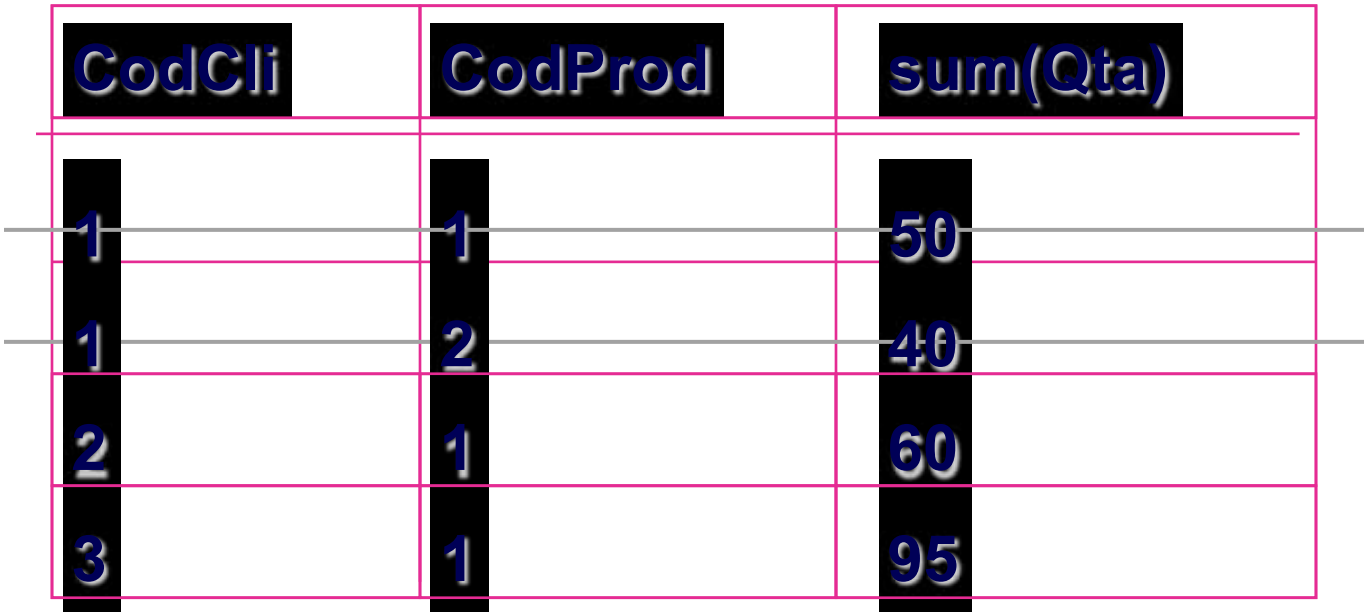
```
select CodCli, CodProd, sum(Qta)
from Ordine as O, Dettaglio as D
Where O.CodOrd = D.CodOrd
group by CodCli, CodProd
having sum(Qta) > 50
```

Situazione dopo il join e il raggruppamento



Estrazione del risultato

- si valuta la funzione aggregata **`sum(Qta)`** e il predicato **`having`**



Query nidificate

- Nella clausola **`where`** e nella clausola **`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
```

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')
```

  - 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')
```

Query nidificate

- *AttrExpr Operator* < **`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 CodOrd in
          (select CodOrd
           from Dettaglio
           where CodProd = 'ABC')
```

- L'operatore `not in` è equivalente a `<> all`

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

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)
```

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)
```

Uso di `in` nelle modifiche

- Aumentare di 5 euro l'importo di tutti gli ordini checomprendono 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)
```

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
```