Interrogazioni semplici in SQL

SQL come linguaggio di interrogazione

- 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`
- 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 target list

Interpretazione algebrica delle query SQL

- La query generica:

  `select` *T1.Attributo1*, …, *Th.Attributoh*
  `from` *Tabella1*, …, *Tabellan*
  `where` *Condizione*

- corrisponde all'interrogazione in algebra relazionale:

$$\pi_{T1.Attributo1,Th.Attributoh}(\sigma_{Condizione}(Tabella1 \times \ldots \times Tabellan))$$

Esempio: gestione degli esami universitari

Studente

MATR	NOME	CITTA'	CDIP
123	Carlo	Bologna	Inf
415	Paola	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

Interrogazione semplice

```
select  *
from Studente
```

MATR	NOME	CITTA'	CDIP
123	Carlo	Bologna	Inf
415	Paola	Torino	Inf
702	Antonio	Roma	Log

Interrogazione semplice

Studente

Matr	Nome	Città	CDip

```
select Nome
from Studente
where CDip = 'Log'
```

interpretazione algebrica
(a meno dei duplicati)
$\Pi_{\text{Nome}}\ \sigma_{\text{CDip='Log'}}$ **Studente**

Sintassi nella clausola `select`

```
select  *
select  Nome, Città
select distinct  Città
select  Città as LuogoDiResidenza
select RedditoCatastale * 0.05
         as TassaIci
select  sum(Salario)
```

Sintassi della clausola `from`

```
from  Studente
from  Studente as X
from  Studente, Esame
from  Studente join Esame
      on Studente.Matr=Esame.Matr
```

Sintassi della clausola `where`

- Espressione booleana di predicati semplici (come in algebra)
- Alcuni predicati aggiuntivi:
  - **between:**
    ```
    Data between 1-1-90 and 31-12-99
    ```
  - **like:**
    ```
    CDip like 'Lo%'
    Targa like 'MI_777_8%'
    ```

Congiunzione di predicati

- Estrarre gli studenti di informatica originari di Bologna:
  ```
  select *
  from Studente
  where CDip = 'Inf' and
        Città = 'Bologna'
  ```
- Risultato:

Matr	Nome	Città	CDip
123	Carlo	Bologna	Inf

Disgiunzione di predicati

- Estrarre gli studenti originari di Bologna o di Torino:
  ```
  select *
  from Studente
  where Città = 'Bologna' or
        Città = 'Torino'
  ```
- Risultato:

Matr	Nome	Città	CDip
123	Carlo	Bologna	Inf
415	Paola	Torino	Inf

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

Operatore `like`

- Estrarre gli studenti con un nome che ha una 'a' in seconda posizione e finiscono per 'o':

```
select *
from Studente
where Nome like '_a%o'
```

- Risultato:

Matr	Nome	Città	CDip
123	Carlo	Bologna	Inf

Duplicati

- In algebra relazionale e nel calcolo, i risultati delle interrogazioni non contengono elementi 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`

Duplicati

```
select
distinct CDip
from Studente
```

CDip
Inf
Log

```
select CDip
from Studente
```

CDip
Inf
Inf
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
- SQL-89 usa una logica a due valori
  - un confronto con *null* restituisce FALSE
- SQL-2 usa una logica a tre valori
  - un confronto con *null* restituisce UNKNOWN
- Per fare una verifica sui valori nulli:
  Attributo **`is`** [**`not`**] **`null`**

Predicati e valori nulli

- **logica a tre valori (V,F,U)**

V and U = U
V or U = V

F and U = F
F or U = U

U and U = U
U or U = U
not U = U

- **P =**

```
(Città is not null) and
(CDip like 'Inf%')
```

Città	CDip
Milano	Inf
Milano	NULL
NULL	Inf
Milano	Log

P	TUPLA SELEZ
V	si
U	no
F	no
F	no

Interrogazioni sui valori nulli

```
select *
from Studente
where Città  is  [not]  null
```

se `Città` ha valore *null*
`(Città = 'Milano')` ha valore Unknown

Interrogazioni sui valori nulli

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

è equivalente a:

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

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 like 'Lo%' and Voto = 30
```

NOME
Carlo

Interrogazione semplice con tre tabelle

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

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

$\Pi_{Nome}\ \sigma_{(Titolo\ like\ 'Mat\%') \wedge (Voto=30)}$ (Studente $\triangleright\!\triangleleft$ Esame $\triangleright\!\triangleleft$ Corso)

Join in SQL-2

- SQL-2 ha introdotto una sintassi alternativa per i join, rappresentadoli esplicitamente nella clausola `from`:

  `select` *AttrEspr* {, *AttrEspr*}
  `from` *Tabella* { [*TipoJoin*] `join` *Tabella* `on` *Condizioni* }
  [`where` *AltreCondizioni*]

Join di due tabelle in SQL-2

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

```
select Nome
from Studente join Esame
     on Studente.Matr = Esame.Matr
where CDip like 'Lo%'and Voto = 30
```

Database d'esempio: guidatori e automobili

DRIVER

FirstName	Surname	DriverID
Mary	Brown	VR 2030020Y
Charles	White	PZ 1012436B
Marco	Neri	AP 4544442R

AUTOMOBILE

CarRegNo	Make	Model	DriverID
ABC 123	BMW	323	VR 2030020Y
DEF 456	BMW	Z3	VR 2030020Y
GHI 789	Lancia	Delta	PZ 1012436B
BBB 421	BMW	316	MI 2020030U

- Notate che Marco Neri NON guida e che la BMW 316 NON e` guidata

Inner Join (normale)

- *TipoJoin* può essere `inner`, `right`, `left` oppure `full`, consentendo la rappresentazione dei join esterni
- Estrarre i guidatori con le loro macchine

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Risultato:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta

Left join

- Estrarre i guidatori con le loro macchine, includendo anche i guidatori senza macchine:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver left join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Risultato:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL

Right join

- Estrarre i guidatori con le loro macchine, includendo anche le macchine senza guidatori:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver right join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Risultato:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
NULL	NULL	NULL	BBB 421	BMW	316

Full join

- Estrarre tutti i guidatori e tutte le automobili, includendo sia le macchine senza guidatori che i guidatori senza macchine:

```
select FirstName, Surname, Driver.DriverID
       CarRegNo, Make, Model
from Driver full join Automobile on
     (Driver.DriverID=Automobile.DriverID)
```

- Risultato:

FirstName	Surname	DriverID	CarRegNo	Make	Model
Mary	Brown	VR 2030020Y	ABC 123	BMW	323
Mary	Brown	VR 2030020Y	DEF 456	BMW	Z3
Charles	White	PZ 1012436B	GHI 789	Lancia	Delta
Marco	Neri	AP 4544442R	NULL	NULL	NULL
NULL	NULL	NULL	BBB 421	BMW	316

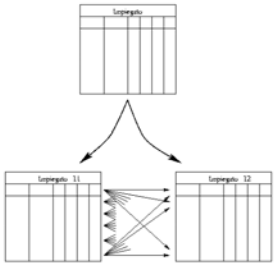
Sintassi completa

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

ALIAS:

- ridenominazione del risultato nella clausola select
- nomi di variabili relazionali nella clausola from

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

Comandi di modifica

Comandi di modifica in SQL

- Istruzioni per
  - inserimento (**insert**)
  - cancellazione (**delete**)
  - modifica dei valori degli attributi (**update**)
- Tutte le istruzioni possono operare su un insieme di tuple (set-oriented)
- Il comando può contenere una condizione, nella quale è possibile fare accesso a tabelle esterne

Inserimento

- Sintassi:

  insert into *NomeTabella* [(*ListaAttributi*)]
  < **values** (*ListaDiValori*) | *SelectSQL*>
- Usando **values**:

```
insert into Studente
values ('456878', 'Giorgio Rossi',
        'Bologna', 'Logistica')
```

- Usando una query:

```
insert into Bolognesi
  (select *
   from Studente
   where Città = 'Bologna')
```

Inserimento

- L'ordine degli attributi e dei valori è significativo (notazione posizionale, il primo valore viene associato al primo attributo, e così via)
- Se la *ListaAttributi* viene omessa, si considerano tutti gli attributi della relazione, nell'ordine in cui compaiono nella definizione della tabella
- Se la *ListaAttributi* non contiene tutti gli attributi della relazione, agli attributi rimanenti viene assegnato il valore di default (se definito, altrimenti il valore *null*)

Inserimento

- Usando **values** con *ListaAttributi*:

```
insert into Studente(Matr,Nome,Città,CDip)
values ('456878', 'Giorgio Rossi',
        'Bologna', 'Logistica')
```

- Usando una query con *ListaAttributi*:

```
insert into Bolognesi(Matr,Nome,Città,CDip)
  (select Matr, Nome, Città, CDip
   from Studente
   where Città = 'Bologna')
```

Cancellazioni

- Sintassi:
  delete from *NomeTabella* [**where** *Condizione*]
- Cancellare lo studente con matricola 678678:

```
delete from Studente
  where Matr = '678678'
```

- Cancellare gli studenti che non hanno sostenuto esami:

```
delete from Studente
  where Matr not in (select Matr
                     from Esame)
```

Cancellazioni

- L'istruzione **delete** cancella dalla tabella tutte le tuple che soddisfano la condizione
- Il comando può provocare delle cancellazioni in altre tabelle, se è presente un vincolo d'integrità referenziale con politica **cascade**
- Se si omette la clausola **where**, il comando **delete** cancella tutte le tuple
- Per cancellare tutte le tuple da STUDENTE (mantenendo lo schema della tabella):

```
delete from Studente
```

- Per cancellare completamente la tabella STUDENTE (contenuto e schema):

```
drop table Studente cascade
```

Modifiche

- Sintassi:

update *NomeTabella*
set *Attributo* **=** < *Espressione* | *SelectSQL* | **null** | **default** >
{ **,** *Attributo* **=** < *Espressione* | *SelectSQL* | **null** | **default** >}
[**where** *Condizione*]

- Esempi:

```
update Esame
set Voto = 30
where Data = 1-4-03

update Esame
set Voto = Voto + 1
where Matr = '787989'
```

Modifiche

- Poiché il linguaggio è set-oriented, è molto importante l'ordine dei comandi

```
update Impiegato
  set Stipendio = Stipendio * 1.1
  where Stipendio <= 30
update Impiegato
  set Stipendio = Stipendio * 1.15
  where Stipendio > 30
```

- Se i comandi sono scritti in questo ordine, alcuni impiegati possono ottenere un aumento doppio

Interrogazioni con ordinamento e raggruppamento

Classificazione delle interrogazioni complesse

- Query con ordinamento
- Query con aggregazioni
- Query con raggruppamento
- Query binarie
- Query nidificate

Esempio: gestione ordini

Cliente

CODCLI	INDIRIZZO	P-IVA

Ordine

CODORD	CODCLI	DATA	IMPORTO

Dettaglio

CODORD	CODPROD	QTA

Prodotto

CODPROD	NOME	PREZZO

Istanza di ordine

Ordine

CODORD	CODCLI	DATA	IMPORTO
1	3	1-6-97	50.000.000
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

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

- Le interrogazioni con funzioni aggregate non possono essere rappresentate in algebra relazionale
- Il risultato di una query con funzioni aggregate dipende dalla valutazione del contenuto di un insieme di righe
- SQL-2 offre 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
```

MaxImp	MinImp
50.000.000	1.500.000

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(Importo)`** per ciascun gruppo

CodCli	sum (Importo)	count (Importo)
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(Importo) >= 2`**

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

Passo 5 : Produzione del risultato

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
```

Query con raggruppamento e ordinamento

- È possibile ordinare il risultato delle query con raggruppamento

```
select  …..
from …..
[ where ….. ]
group by …..
[ having ….. ]
order by ….
```

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(Importo) >= 2
order by sum(Importo) 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

Ordine		Dettaglio			
CodCli	Ordine. CodOrd	Dettaglio. CodOrd	CodProd	Qta	
1	3	3	1	30	gruppo 1,1
1	4	4	1	20	
1	3	3	2	30	gruppo 1,2
1	5	5	2	10	
2	3	3	1	60	gruppo 2,1
3	1	1	1	40	
3	2	2	1	30	gruppo 3,1
3	6	6	1	25	

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

Interrogazioni binarie

Query binarie (set queries)

•Costruite concatenando due query SQL tramite operatori insiemistici
•Sintassi:
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
```

Unione con all

- Estrarre i codici degli ordini i cui importi superano 500 euro oppure in cui qualche prodotto è presente con quantità superiore a 1000, ripetendo il codice tante volte quante e` presente in ordini e dettagli.

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

Unioni compatibili ma differenti

```
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

Unioni compatibili ma differenti, 2

```
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

Altro esempio di uso di **`all`**

- Estrarre i padri di persone con nome "Giorgio" o "Giovanni", presentando due volte i padri che hanno due figli con ciascuno dei nomi

```
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
```

Differenza con all

- Estrarre i codici degli ordini che presentano n >=1 linee d'ordine con quantità maggiore di 10 e non presentano un numero m >= n di linee d'ordine con quantità superiore a 1000

```
select CodOrd
from Dettaglio
where Qta > 10
      except all
select CodOrd
from Dettaglio
where Qta > 1000
```

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
```

Interrogazioni nidificate

Query nidificate

- Nella clausola **`where`** 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`** è chiamata query nidificata

Query nidificate, 2

- Altre query:

  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

  <*exists* | *not exists*> *SelectSQL*
  - **`exists`**: il predicato è vero se la query possiede tuple
  - **`not exists`**: il predicato è vero se la query non possiede tuple

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

- Equivalente a (senza query nidificata)

```
select CodOrd
from Dettaglio D, Prodotto P
where D.CodProd = P.CodProd
  and Prezzo > 100
```

Query nidificate con any, 2

- Estrarre i prodotti ordinati assieme al prodotto avente codice 'ABC'
  - con una query nidificata:

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

  - senza query nidificata, a meno di duplicati:

```
select D1.CodProd
from Dettaglio D1, Dettaglio D2
where D1.CodOrd = D2.CodOrd 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')
```

- In alternativa:

```
select CodOrd
from Ordine
   except
select CodOrd
from Dettaglio
where CodProd = 'ABC'
```

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')
```

Altro esempio con "in"

- Estrarre nome e indirizzo dei clienti che hanno emesso qualche ordine di importo superiore a 10.000

```
select Nome, Indirizzo
from Cliente
where CodCli in
  select CodCli
      from Ordine
      where Importo > 10000
```

Query nidificate a più livelli

- Estrarre nome e indirizzo dei clienti che hanno emesso qualche ordine che comprende il prodotto "Pneumatico"

```
select Nome, Indirizzo
from Cliente
where CodCli in
       select CodCli
       from Ordine
       where CodOrd in
              select CodOrd
              from Dettaglio
              where CodProd in
                    select CodProd
                    from Prodotto
                    where Nome = 'Pneumatico'
```

La query equivalente

• La query precedente equivale (a meno di duplicati) a:

```
select C.Nome, Indirizzo
from Cliente as C, Ordine as O,
     Dettaglio as D, Prodotto as P
where C.CodCli = O.CodCli
  and O.CodOrd = D.CodOrd
  and D.CodProd = P.CodProd
  and P.Nome = 'Pneumatico'
```

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)
```

L'operatore **exists**

- Si può usare il quantificatore esistenziale sul risultato di una query SQL
- Sintassi:

  exists *SelectStar*

- il predicato è vero se la query *SelectStar* restituisce un risultato non nullo
  (selectStar: si consiglia di usare sempre **select *** perché è irrilevante la proiezione)

Query nidificate complesse

- La query nidificata può usare variabili della query esterna
  - Interpretazione: la query nidificata viene valutata per ogni tupla della query esterna
- Estrarre tutti i clienti che hanno emesso più di un ordine nella stessa giornata:

```
select CodCli
from Ordine O
where exists (select *
              from Ordine O1
              where O1.CodCli = O.CodCli
                and O1.Data = O.Data
                and O1.CodOrd <> O.CodOrd)
```

Query nidificate complesse

- Estrarre tutte le persone che [non] hanno degli omonimi:

```
select *
from Persona P
where [not] exists
        (select *
         from Persona P1
         where P1.Nome = P.Nome
           and P1.Cognome = P.Cognome
           and P1.CodFisc <> P.CodFisc)
```

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)
- La query precedente può essere espressa così:

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

Commenti sulle query nidificate

- L'uso di query nidificate può produrre query 'meno dichiarative', ma spesso si migliora la leggibilità
- Le sottointerrogazioni non possono contenere operatori insiemistici ("l'unione si fa solo al livello esterno"); la limitazione non è significativa, ed è superata da alcuni sistemi

Commenti sulle query nidificate

- Query complesse, che fanno uso di variabili, possono diventare molto difficili da comprendere
- L'uso delle variabili deve rispettare le regole di visibilità
  - una variabile può essere usata solamente all'interno della query dove viene definita o all'interno di una query che è ricorsivamente nidificata nella query dove è definita
  - se un nome di variabile è ambiguo, si assume il riferimento alla variabile più vicina

Visibilità delle variabili

- Query scorretta:

```
select *
from Cliente
where CodCli in
          (select CodCli
           from Ordine O1
           where CodOrd = 'AZ1020')
   or CodCli in
          (select CodCli
           from Ordine O2
           where O2.Data = O1.Data)
```

- La query è scorretta poiché la variabile `O1` non è visibile nella seconda query nidificata

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

Viste

- Offrono la "visione" di tabelle virtuali (schemi esterni)
- Classificate in:
  - semplici (selezione e proiezione su una sola tabella)
  - complesse
- Sintassi:
  `create view` *NomeVista* [(*ListaAttributi*)] **`as`** *SelectSQL*
  [**`with`** [**`local`** | **`cascaded`**] **`check option`**]

Viste

- Le viste in SQL-2 possono contenere nella definizione altre viste precedentemente definite, ma non vi può essere mutua dipendenza (la ricorsione è stata introdotta in SQL:1999)
- 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

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)
```

- Questa soluzione può non essere riconosciuta da tutti i sistemi SQL

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
```

PROVA IN ITINERE QUI!!!

- Buon lavoro a tutti……

Esempio di vista semplice

- Ordini di importo superiore a 10.000

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

Ordine

1	3	1-6-96	50.000
4	1	1-7-97	12.000
6	3	3-9-97	27.000

VISTA : ordini principali

Viste semplici 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
with check option
```

Modifiche tramite le viste

- Vista:

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

- Modifica:

```
update OrdiniPrincipali
  set Importo = Importo * 1.05
  where CodCli = '45'
```

- Composizione della vista con la modifica:

```
update Ordine
  set Importo = Importo * 1.05
  where CodCli = '45'
    and Importo > 10000
```

Check option

- La **check option** interviene quando viene aggiornato il contenuto di una vista, per verificare che la tupla inserita/modificata appartenga alla vista
- Se l'opzione è **local**, il controllo viene fatto solo rispetto alla vista su cui viene invocato il comando
- Se l'opzione è **cascaded**, il controllo viene fatto su tutte le viste coinvolte
- Es.:

```
create view OrdiniPrinc70 as
  select *
  from OrdiniPrincipali
  where CodCli = '70'
  with local check option
```

Check option

-
```
update OrdiniPrinc70
set CodCli = '71'
where CodOrd = '754'
```

  viene rifiutato con check option **local** e **cascaded**

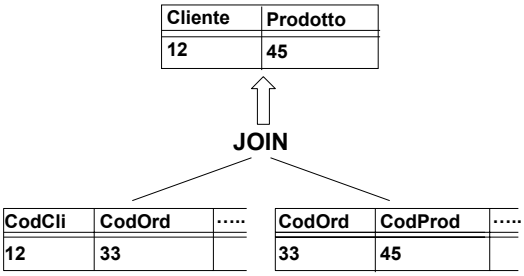
-
```
update OrdiniPrinc70
set Importo = 5000
where CodOrd = '754'
```

  viene accettato dalla **local**, rifiutato dalla **cascaded**

Esempio di vista complessa

```
create view CliPro(Cliente,Prodotto) as
   select CodCli, CodProd
   from Ordine join Dettaglio
     on Ordine.CodOrd = Dettaglio.CodOrd
```

Vista complessa (JOIN)



Interrogazione sulla vista complessa

- Query:

```
select Cliente
from CliProd
where Prodotto = '45'
```

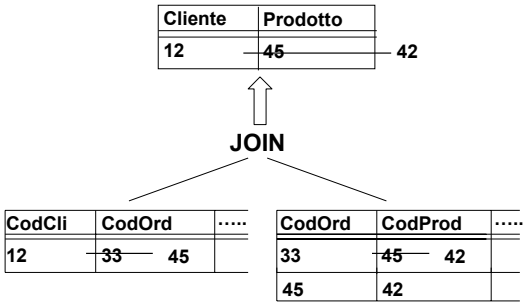
- Composizione della vista con la query:

```
select CodCli
from Ordine join Dettaglio
  on Ordine.CodOrd = Dettaglio.CodOrd
where CodProd = '45'
```

Modifiche sulla vista complessa

- Non è possibile modificare le tabelle di base tramite la vista perché la interpretazione è ambigua
- Es.: `update CliProd`
  `set Prodotto = '42'`
  `where Cliente = '12'`
- Due alternative per la realizzazione sulle tabelle di base
  - il cliente ha cambiato l'ordine
  - il codice del prodotto è cambiato

Vista complessa (JOIN)



Ricorsione in SQL:1999

```
with recursive Raggiungibile (Orig,Dest,Costo) as
( select Orig, Dest, Costo
  from Volo where Orig = 'Milano'
  union
  select V.Orig, R.Dest, V.Costo+R.Costo
  from Volo V join Raggiungibile R
       on V.Dest = R.Orig )

select distinct Dest, Costo
from Raggiungibile R
where Costo = (select min(Costo)
               from Raggiungibile R1
               where R.Dest = R1.Dest)
```

Vincoli nel DDL

Qualità dei dati

- Qualità dei dati:
  - correttezza, completezza, attualità
- In molte applicazioni reali i dati sono di scarsa qualità (5% - 40% di dati scorretti)
- Per aumentare la qualità dei dati:
  - Regole di integrità
  - Manipolazione dei dati tramite programmi predefiniti (procedure e trigger)

Vincoli di integrità generici

- Predicati che devono essere veri se valutati su istanze corrette (legali) della base di dati
- Espressi in due modi:
  - negli schemi delle tabelle
  - come asserzioni separate

Vincoli d'integrità generici

- La clausola **check** può essere usata per esprimere vincoli arbitrari nella definizione dello schema
- Sintassi:

  check (*Condizione*)
- *Condizione* è ciò che può apparire in una clausola `where` (comprese le query nidificate)
- Es., la definizione di un attributo *Superiore* nello schema della tabella IMPIEGATO:

```
Superiore character(6)
check (Matr like "1%" or
       Dipart = (select Dipart
                 from Impiegato I
                 where I.Matr = Superiore)
```

Esempio: gestione magazzino

Magazzino

CodProd	QtaDisp	QtaRiord
1	150	100
3	130	80
4	170	50
5	500	150

Riordino

CodProd	Data	QtaOrd

Esempio: definizione di Magazzino

```
create table Magazzino as
  ( CodProd    char(2) primary key,
    QtaDisp    integer not null
               check(QtaDisp > 0),
    QtaRiord   integer not null
               check(QtaRiord > 10))
```

Asserzioni

- Le asserzioni permettono la definizione di vincoli al di fuori della definizione delle tabelle
- Utili in molte situazioni (es., per esprimere vincoli inter-relazionali di tipo generico)
- Una asserzione associa un nome a una clausola `check`; sintassi:

  create assertion *NomeAsserzione* **check** (*Condizione*)
- Es., la tabella IMPIEGATO deve contenere almeno una tupla:

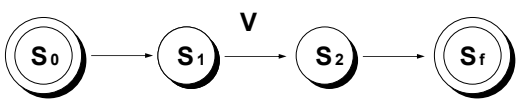
```
create assertion SempreUnImpiegato
  check (1 <= (select count(*)
               from Impiegato))
```

Significato dei vincoli

La verifica dei vincoli può essere:

a immediate (immediata):
la loro violazione annulla l'ultima modifica

b deferred (differita):
la loro violazione annulla l'intera applicazione



Modifica dinamica del significato dei vincoli

• Ogni vincolo è definito di un tipo (normalmente "immediate")
•L'applicazione può modificare il tipo iniziale dei vincoli:
- **set constraints immediate**
- **set constraints deferred**

•Tutti i vincoli vengono comunque verificati, prima o poi

Controllo dell'accesso

Controllo dell'accesso

- Privatezza: protezione selettiva della base di dati in modo da garantire l'accesso solo agli utenti autorizzati
- Meccanismi per identificare l'utente (tramite *parola chiave* o *password*):
  - Quando si collega al sistema informatico
  - Quando accede al DBMS
- Utenti individuali e gruppi di utenti

Autorizzazioni

- Ogni componente dello schema può essere protetto (tabelle, attributi, viste, domini, etc.)
- Il proprietario di una risorsa (il creatore) assegna privilegi (autorizzazioni) agli altri utenti
- Un utente predefinito **`_system`** rappresenta l'amministratore di sistema e ha pieno accesso a tutte le risorse
- Un privilegio è caratterizzato da:
  - la risorsa
  - l'utente che concede il privilegio
  - l'utente che riceve il privilegio
  - l'azione che viene consentita sulla risorsa
  - la possibilità di passare il privilegio ad altri utenti

Tipi di privilegi

- SQL offre 6 tipi di privilegi
  - **`insert`**: per inserire un nuovo oggetto nella risorsa
  - **`update`**: per modificare il contenuto della risorsa
  - **`delete`**: per rimuovere un oggetto dalla risorsa
  - **`select`**: per accedere al contenuto della risorsa in una query
  - **`references`**: per costruire un vincolo di integrità referenziale che coinvolge la risorsa (può limitare la modificabilità della risorsa)
  - **`usage`**: per usare la risorsa in una definizione di schema (es., un dominio)
- **`all privileges`** li riassume tutti

`grant` e `revoke`

- Per concedere un privilegio a un utente:
  `grant` < *Privilegi* | **`all privileges`** > **`on`** *Risorsa* **`to`** *Utenti* [**`with grant option`**]
  - **`grant option`** specifica se deve essere garantita la possibilità di propagare il privilegio ad altri utenti
- Per revocare un privilegio:
  `revoke` *Privilegi* **`on`** *Risorsa* **`from`** *Utenti* [**`restrict`** | **`cascade`**]

Esempi

```
grant all privileges on Ordine to User1
grant update(Importo) on Ordine to User2
grant select on Ordine to User2, User3

revoke update on Ordine from User1
revoke select on Ordine from User3
```

Esempio di uso, grant option

1 Database administrator

```
grant all privileges on Ordine to User1
  with grant option
```

2 User1

```
grant select on Ordine to User2
  with grant option
```

3 User2

```
grant select on Ordine to User3
```

Revoca di un privilegio con cascata

1 Database administrator

```
grant select on Ordine to User1
    with grant option
```

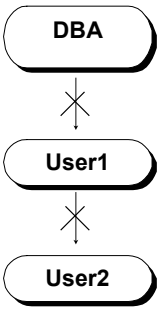
2 User1

```
grant select on Ordine to User2
```

3 Database administrator

```
revoke select on Ordine from User1 cascade
```

Revoca di un privilegio con cascata

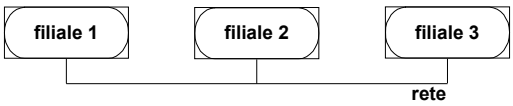


Viste e autorizzazioni di accesso

Viste = unità di autorizzazione

- Consentono la gestione ottimale della privatezza

Esempio: gestione dei conti correnti

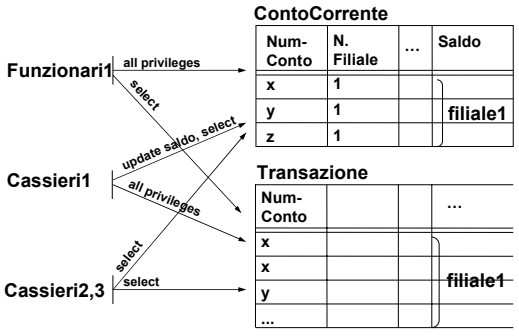


Banca

ContoCorrente(<u>NumConto</u>, Filiale, Cliente, CodFisc, DataApertura, Saldo)

Transazione(<u>NumConto, Data, Progr,</u> Causale, Ammontare)

Requisiti di accesso



Viste relative alla prima filiale

```
create view Conto1 as
( select *
  from ContoCorrente
  where Filiale = 1)

create view Transazione1 as
( select *
  from Transazione
  where NumConto in
        ( select NumConto
          from Conto1 ) )
```

Autorizzazioni relative ai dati della prima filiale

```
grant all privileges on Conto1
        to Funzionari1
grant update(Saldo) on Conto1
        to Cassieri1
grant select on Conto1
        to Cassieri1, Cassieri2, Cassieri3
grant select on Transazione1
        to Funzionari1
grant all privileges on Transazione1
        to Cassieri1
grant select on Transazione1
        to Cassieri2, Cassieri3
```

Procedure

Procedure

- Moduli di programma che svolgono una specifica attività di manipolazione dei dati
- SQL-2 permette la definizione di procedure (anche note come *stored procedures*), ma solo in forma molto limitata
- La maggior parte dei sistemi offrono delle estensioni che permettono di scrivere procedure complesse (es., Oracle PL/SQL), con strutture di controllo, variabili, eccezioni, etc.
  - si ottiene un linguaggio di programmazione completo
- Due momenti:
  - dichiarazione (DDL)
  - invocazione (DML)
- Con architettura client-server sono normalmente:
  - invocate dai client
  - memorizzate ed eseguite presso i server

Esempio : prelievo dal magazzino

Magazzino

CodProd	QtaDisp	QtaRiord
1	150	100
3	130	80
4	170	50
5	500	150

Riordino

CodProd	Data	QtaOrd

Specifica

- L'utente indica un prelievo dando il codice del prodotto e la quantità da prelevare
- Se la quantità disponibile in magazzino non è sufficiente la procedura si arresta con una eccezione
- Viene eseguito il prelievo, modificando la quantità disponibile in magazzino
- Se la quantità disponibile in magazzino è inferiore alla quantità di riordino si predispone un nuovo ordine d'acquisto

Interfaccia

```
procedure Prelievo
     ( Prod   integer,
       Quant  integer )
```

Invocazione

```
Prelievo(4,150)
```

Stato iniziale nella base di dati

CodProd	QtaDisp	QtaRiord
4	170	50

Realizzazione della procedura

1. Dichiarazione variabili
2. Lettura dello stato
3. Se la quantità disponibile è insufficiente: eccezione
4. Aggiornamento dello stato
5. Se la nuova quantità disponibile è inferiore alla quantità di riordino: emissione di un ordine

Procedura

```
procedure Prelievo (Prod integer, Quant integer) is
begin
 Q1, Q2 integer;
 X exception;
 select QtaDisp, QtaRiord into Q1, Q2
   from Magazzino
   where CodProd = Prod;
 if Q1 < Quant then raise(X);
 update Magazzino
   set QtaDisp = QtaDisp - Quant
   where CodProd = Prod;
 if Q1 - Quant < Q2 then
  insert into Riordino
    values(Prod,sysdate,Q2)
end;
```

Esempio di invocazione

```
Prelievo(4,150)
```

Prod=4, **Quant**=150

```
select QtaDisp, QtaRiord into Q1, Q2
     from Magazzino
     where CodProd = Prod;
```

CodProd	QtaDisp	QtaRiord
4	170	50

Q1 = 170, **Q2** = 50

Invocazione (continua)

```
if Q1 < Quant then raise(X)
```
non scatta

```
update Magazzino
set QtaDisp = QtaDisp - Quant
where CodProd = Prod
```

CodProd	QtaDisp	QtaRiord
4	20	50

```
Q1 - Quant < Q2
```
è vero:

```
  insert into Riordino
     values(Prod, sysdate, Q2)
```

CodProd	Data	QtaRiord
4	10-10-97	50

Problemi del progetto di procedure

- Decomposizione modulare delle applicazioni
- Aumento di:
  - efficienza
  - controllo
  - riuso
- Aumenta la responsabilità dell'amministratore della base di dati (rispetto al programmatore applicativo)
- Si sposta "conoscenza" dalle applicazioni allo schema della base di dati.

Basi di dati attive

- Basi di dati con componente per la gestione di regole **Evento-Condizione-Azione** (regole di produzione):
- Hanno comportamento **reattivo** (in contrasto con **passivo**): eseguono non solo le transazioni utenti ma anche le regole
- Le regole sono simili alle procedure, ma l'invocazione è automatica
- Nell'ambito dei DBMS commerciali si parla di **trigger** (standardizzati in SQL-3)

Paradigma evento-condizione-azione (ECA)

- Intuitivamente:
  - quando si verifica l'*evento* (attivazione)
  - se la *condizione* è soddisfatta (considerazione)
  - allora esegui l'*azione* (esecuzione)
- I trigger sono definiti con istruzioni DDL (`create trigger`)
  - evento: modifica dei dati, specificata con `insert, delete, update`
  - condizione (opzionale) predicato SQL
  - azione: sequenza di istruzioni SQL (o estensioni, ad esempio PL/SQL in Oracle)
- Ogni trigger fa riferimento ad una tabella (target): risponde ad eventi relativi a tale tabella

Granularità

- Granularità
  - di tupla (row-level): attivazione per ogni tupla coinvolta nell'operazione
  - di operazione (statement-level): una sola attivazione per ogni istruzione SQL, con riferimento a tutte le tuple coinvolte (set-oriented)

Regole attive (trigger)

Regola attiva

```
create trigger GestioneRiordino
after update of QtaDisp on Magazzino
when (new.QtaDisp < new.QtaRiord)
for each row
X exception
begin
  if new.QtaDisp < 0 then raise(X);
  insert into Riordino
    values(new.CodProd,sysdate,new.QtaRiord)
end
```

Esempio: gestione automatica del riordino

1. evento:

   `update(QtaDisp)` in Magazzino
2. condizione: la nuova quantità disponibile è inferiore alla (nuova) quantità di riordino:

   `new.QtaDisp < new.QtaRiord`
3. azione:

   se la quantità disponibile è insufficiente: eccezione

   emissione di un ordine

Esecuzione dell'applicazione

```
update Magazzino
  set QtaDisp = QtaDisp - 150
  where CodProd = 4
```

CodProd	QtaDisp	QtaRiord
4	20	50

Esecuzione del trigger

evento : **`update(QtaDisp) on Magazzino`**
condizione : VERA
azione : **`if new.QtaDisp < 0 then raise(X)`**
non scatta

```
insert into Riordino values
 (new.CodProd,sysdate,new.QtaRiord)
```

CodProd	Data	Qta
4	10-10-97	50

Integrità referenziale

- Supponiamo che la tabella magazzino abbia oltre alla voce QtaRiord e QtaDisp anche QtaMax
  - Rappresenta la quantità massima di un articolo che può essere immagazzinato.
- Si introducono due vincoli di integrità referenziale
  - La quantità di riordino non può MAI superare la quantità massima
  - La quantità disponibile non può MAI superare la quantità massima

Politiche di reazione

- Diverse sono le politiche di reazione alla violazione di uno dei due vincoli.
  - Abortire la transazione
  - Modificare il contenuto della quantità eccedente in modo che il valore sia inferiore o uguale al valore massimo
- Attenzione agli effetti nascosti di operazioni automatiche

Quantità disponibile

- Realizziamo la seguente politica per la gestione della quantità disponibile
- Si impedisce la transizione
- Per cui avremo

```
create trigger TroppoDisp
after update of QtaDisp on Magazzino
when (new.QtaDisp > new.QtaMax)
X exeception
Raise(x)
```

Politiche per la quantità di riordino

- La politica di reazione
  - ridurre del 33% la quantità disponibile fino a quando il valore non è inferiore alla quantità massima

```
create trigger TroppoRiod
after update of QtaRiod on Magazzino
when (new.QtaRiod > new.QtaMax)
Update Magazzino set
  QtaRiod=new.QtaRiod*0.77
Where Magazzino.CodProd=new.CodProd
```

Esecuzione dell'applicazione

```
update Magazzino
  set QtaRiord = 200
  where CodProd = 4
```

CodProd	QtaDisp	QtaRiord	QtaMax
4	180	200	190

Descrizione

1. evento:
   `update(QtaRiod)` in Magazzino
2. condizione: la nuova quantità di riordino è maggiore di quella massima prevista per quell'articolo
3. azione:
   Si impone che la quantità di riordino sia diminuita del 33 % rispetto a la valore precedente

Esecuzione del trigger

evento : **`update(QtaRiord) on Magazzino`**
condizione : VERA
azione : **`Update Magazzino set QtaRiod=new.QtaRiod*.77`**
`Where Magazzino.CodProd=new.CodProd`

- **Notare che l'azione fa scatenare ancora un volta lo stesso trigger**

CodProd	QtaDisp	QtaRiord	QtaMax
4	180	154	190

Esecuzione del trigger 2

evento : **`update(QtaRiord) on Magazzino`**
condizione : FALSA
azione : **`NON ESEGUITA`**

Conclusioni

- I trigger possono risultare utili in molti ambiti
- L'errata progettazione del trigger può portare effetti indesiderati
- L'eccessiva proliferazione dei trigger rallenta il DBMS perché si devono controllare tutti i trigger che scattano sull'evento
- Limitazioni alla portabilità verso altri DBMS