Basi di Dati

prof. Letizia Tanca

(lucidi tratti dal libro
Atzeni-Ceri-Paraboschi-Torlone)

Linguaggi di interrogazione "commerciali" per il Modello Relazionale dei Dati: SQL - il DML

Linguaggi di interrogazione

Ricordiamo:

- Permettono di trovare un dato basandosi **sulle sue proprietà.**
- Permettono di trovare dati basandosi su **confronti tra i contenuti** di più tabelle.

Classificazione dei linguaggi

- LINGUAGGI **"COMMERCIALI"**
  - SQL (Structured Query Language)
  - QUEL
  - QBE (Query By Example)

Classificazione dei linguaggi

- **LINGUAGGI DI DEFINIZIONE DEI DATI (DDL)**
  - per creare gli schemi dei dati e definire le loro proprieta'
- **LINGUAGGI DI MANIPOLAZIONE DEI DATI (DML)**
  - per aggiornare le istanze dei dati
  - per l'interrogazione dei dati

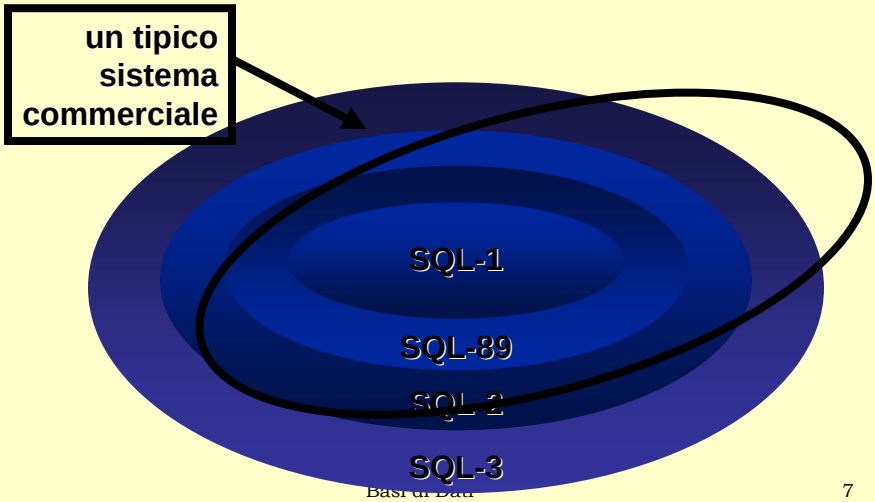
SQL

- Il nome sta per *Structured Query Language*
- Più che un semplice linguaggio di query: si compone di una parte DDL e di una DML
  - **DDL:** definizione di domini, tabelle, indici, autorizzazioni, viste, vincoli, procedure, 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)

Standardizzazione di SQL

- La standardizzazione è stata cruciale per la diffusione di SQL (nell'ambito di ANSI e ISO)
  - Dal 1983, standard de facto
  - Prima versione ufficiale nel 1986 (SQL-1), rivista nel 1989 (SQL-89)
  - Seconda versione nel 1992 (SQL-2 o SQL-92)
  - Terza versione nel 1999 (SQL-3 o SQL:1999)
- In SQL-92 si distinguono tre livelli:
  - Entry SQL (più o meno equivalente a SQL-89)
  - Intermediate SQL
  - Full SQL
- La maggior parte dei sistemi è conforme al livello Entry e offre delle estensioni proprietarie per le funzioni avanzate

Potere espressivo di standard e sistemi commerciali



Il linguaggio SQL

- Data Manipulation Language (DML): interrogazione e modifica
- Data Definition Language: definizione delle relazioni

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

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

Il linguaggio SQL

`Select A1,…,An` (*attributi da includere nel risultato)*
`from R1,…,Rm` (*nomi di relazioni da cui estrarre i dati)*
`where P` (*condizione che i dati cercati soddisfano)*

Esempio:

```
Select MATR
from STUDENTE
where CITTA' = "Bologna"
```


Interrogazioni SQL

- Le interrogazioni SQL hanno una struttura **`select-from-where`**
- Sintassi:
  `select` ***AttrEspr*** **[[as]** ***Alias*** **] {, *AttrEspr* [[as]** ***Alias*** **] }**
  `from` ***Tabella*** **[[as]** ***Alias*** **] {, *Tabella* [[as]** ***Alias*** **] }**
  [`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 T_1.Attributo_11,..,
T_h.Attributo_hm
from Tabella_1 T_1,..., Tabella_n T_n
where Condizione
```

- corrisponde all'interrogazione in algebra relazionale:

$$\pi_{\text{T\_1.Attributo\_11,...,T\_h.Attributo\_hm}}(\sigma_{\text{Condizione}}(\text{Tabella\_1} \times \ldots \times \text{Tabella\_n}))$$

Esempio

- *Trovare gli anni in cui è stato eletto un presidente repubblicano proveniente dall'Illinois*

```
SELECT ANNO
FROM   ELEZIONI, PRESIDENTI
WHERE ELEZIONI.NOME-P = PRESIDENTI.NOME-P
AND ELEZIONI.DATA-N-PRES = PRESIDENTI.DATA-N
AND  STATO = "ILLINOIS"
AND  PARTITO = "REPUBBLICANO"
```

Nota: le condizioni di join, anche naturale, vanno espresse **sempre esplicitamente** (come già accade nel TRC)

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)

$\Pi_{\text{Nome}}\ \sigma_{\text{CDip='Log'}}\ \text{Studente}$

Sintassi nella clausola `select`: esempi

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


Sintassi nella clausola `from`: esempi

```
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	P	TUPLA SELEZ
Milano	Info	V	si
Milano	NULL	U	no
NULL	Inf	F	no
Milano	Log	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 i nomi 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

Join in SQL-2

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

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

- ***TipoJoin*** può essere **`inner`**, **`right`** **[`outer`]**, **`left`** **[`outer`]** oppure **`full`** **[`outer`]**, consentendo la rappresentazione dei join esterni
- La parola chiave **`natural`** può precedere ***TipoJoin*** (però è implementato di rado)

Join di due tabelle in SQL-2

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

oppure

```
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

Left join

- Estrarre i guidatori con le loro macchine, includendo 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

Full join

- Estrarre tutti i guidatori e tutte le automobili, mostrando le possibili relazioni tra di loro:

```
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

Interrogazione semplice con tre tabelle

- Estrarre i nomi degli studenti di “Matematica” che hanno preso almeno un 30

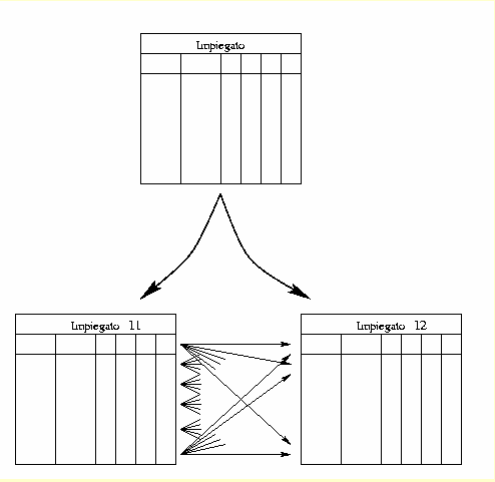
```
select Nome
from Studente, Esame, Corso
where  Studente.Matr = Esame.Matr
  and Corso.CodCorso = Esame.CodCorso
  and Titolo like 'Mat%' and Voto = 30
```

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

Variabili in SQL

- Gli alias di tabella possono essere interpretati come variabili, che assumono valori su intere tabelle
- L'uso delle variabili può servire a implementare l'operatore di ridenominazione ρ dell'algebra relazionale

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

Esempio di schema di base di dati

PRESIDENTI (NOME-P, DATA-N, DATA-M, PARTITO, STATO, NOME-M)

CONGRESSI (# CONGRESSO, %S-REP, %C-REP, %S-DEM, %C-DEM)

AMMINISTRAZIONI (# AMMIN, DATA-IN, VICE-PRES, DATA-N-VP, NOME-P, DATA-N-P)

ELEZIONI (ANNO, VOTI-PRES, NOME-P, DATA-N-PRES, NOME-PERD, DATA-N-PERD, VOTI-PERD)

STATI (STATO, POPOLAZ, # AMMIN.)

PRESID-CONGR (NOME-P, DATA-N, # CONGR)

Interrogazioni

- ***Trovare l'anno di nascita del presidente J. F. Kennedy***

```
select DATA-N
from PRESIDENTI
where  NOME-P="J.F. KENNEDY"
```

- ***Trovare gli anni in cui è stato eletto un presidente repubblicano proveniente dall'Illinois***

```
select E.ANNO
from PRESIDENTI as P, ELEZIONI as E
where  P.NOME-P=E.NOME-P AND P.DATA-
  N=E.DATA-N-PRES AND
  P.PARTITO="REPUBBLICANO" AND
  P.STATO="ILLINOIS"
```

Interrogazioni

- ***Trovare i numeri di congressi presieduti dal presidente eletto nel 1955***

```
select #CONGR
from PRESID-CONGR as P JOIN ELEZIONI as E
       ON P.NOME-P=E.NOME-P AND P.DATA-
       N=E.DATA-N-PRES
where  E.ANNO=1955
```

- ***Trovare i perdenti delle elezioni vinte da qualche presidente di nome Roosevelt***

```
select NOME-PERD,DATA-N-PERD
from ELEZIONI
where  NOME-P="ROOSVELT"
```


Interrogazioni

- ***Trovare i nomi delle mogli dei presidenti provenienti dalla California eletti dopo il 1960***

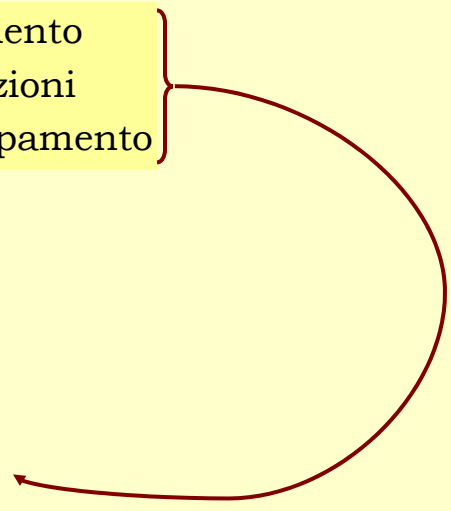
```
select P.NOME-M
from PRESIDENTI as P JOIN ELEZIONI as
  E   ON P.NOME-P=E.NOME-P AND P.DATA-
  N=E.DATA-N-PRES
where  P.STATO="CALIFORNIA" AND
  E.ANNO>1960
```

Interrogazioni con ordinamento e raggruppamento

Classificazione delle interrogazioni complesse

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

Non possono essere espresse in Algebra Relazionale!!!



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 tuple
- 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? Infatti non è detto che l'importo massimo corrisponda a una sola data!!!

SQL vieta di indicare nella target list valori a livello di tupla insieme con valori aggregati.

- **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 insiemi di tuple*
- Si aggiungono le clausole
  - **group by** (raggruppamento)
  - **having** (condizione di selezione *sui 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
```

Notare: in questo caso abbiamo indicato nella target list valori a livello di tupla insieme con valori aggregati, ma qui dipende dal fatto che si tratta dell'attributo di grouping!!!

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
```

In questo caso a ogni valore di `CodCli` possono corrispondere diversi valori di `Importo.`

E' richiesto che ad ogni valore dell'attributo di grouping corrisponda **solo una tupla** del risultato

Query con group by e target list

Query scorretta:

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

Questa query sembra richieda, per ogni cliente, il numero di ordini e la città. Teoricamente potrebbe essere corretta, perchè possiamo notare che **a ogni cliente corrisponde una sola città.**

SQL non l'ammette perchè non viene fatto nessun check sul fatto che **il cliente determina univocamente la propria città**

Query con group by e target list

Si risolve così:

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

abbiamo raggruppato **anche rispetto alla città.**

where o having?

- Soltanto i predicati che richiedono la valutazione di funzioni aggregate possono comparire nell'argomento della clausola **having**

Estrarre i dipartimenti in cui lo stipendio medio degli impiegati assunti dopo il 1 luglio 1997 è maggiore di 25:

```
select Dipart
from Impiegato
where Data_ass > 1-7-97
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 e nidificate

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
```

Nome degli attributi nel risultato

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

- 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
```

- Anche in questo caso la query può essere rappresentata usando una query nidificata

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:

  ScalarValue 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 semplici

- 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, a meno di duplicati)

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

Query nidificate semplici

- Estrarre i prodotti ordinati assieme al prodotto avente codice 'ABC'
  - senza query nidificate:

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

  - con una query nidificata:

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


Query nidificate

- **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 e min con query nidificate

- Gli operatori aggregati **max** e **min** possono essere espressi tramite query nidificate
- Estrarre l'ordine con il massimo importo
  - usando **max**:

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

  - con una query nidificata:

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

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
```


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
  (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) [not] 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
- La prima versione di SQL prevedeva solo la forma nidificata (o strutturata) con una sola relazione nella clausola **`from`**, il che è insoddisfacente
- 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 omesso, si assume il riferimento alla variabile più vicina (non lo fate mai!!!)

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

Un vincolo sulle query con group by

Ricordarsi sempre di inserire nella target list gli attributi di grouping!!!

Un'anticipazione del prossimo gruppo di slides

Viste

- Offrono la "visione" di tabelle virtuali (schemi esterni)
- Già viste con Datalog
- Classificate in:
  - semplici *(selezione e proiezione su una sola tabella)*
  - complesse
- Sintassi:

create view *NomeVista* [(*ListaAttributi*)] **as**
SelectSQL
[**with** [**local** | **cascaded**] **check option**]

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
   (Matr1,Nome1,Cognome1,Stipendio1) as
select Matr, Nome, Cognome, Stipendio
from Impiegato
where Dipart = 'Amministrazione'
  and Stipendio > 10


create view ImpiegatoAmminJunior as
select *
from ImpiegatoAmmin
where Stipendio1 < 50
```


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