Basi di Dati

prof. A. Longheu

4 – Il linguaggio SQL

SQL
Structured Query Language

- Una volta progettato un DB, per crearlo realmente su una macchina e per utilizzarlo (inserire, ricercare, aggiornare o cancellare dati) occorre un linguaggio. Il più usato è l'SQL
- Linguaggio realizzato dall'IBM negli anni 70; contiene sia le funzionalità di un DDL che quelle di un DML
- E' stato standardizzato dall'ISO e dell'ANSI, esistono diverse versioni (1986, 1992 - SQL2, 1999 - SQL3), oltre a svariati dialetti sviluppati per specifici DBMS.
- L'SQL viene utilizzato direttamente dagli utenti finali, ma più spesso i db mettono a disposizione una serie di procedure (gruppo già pronto di comandi SQL) e/o di interfacce grafiche (ad esempio, l'autocomposizione query di Access) per facilitare l'uso del linguaggio.

SQL
Structured Query Language

- In pratica, in SQL si lavora dando dei comandi (denominati **query**), che nella ipotesi più primitiva sono forniti attraverso un'interfaccia testuale (tipo finestra dei comandi o shell), oppure possono essere inviati all'interno di un'interfaccia grafica (che comunque dietro le quinte opera sempre con comandi testuali)
- L'esecuzione di una query può produrre un risultato visibile (ad esempio quando si richiedono dati) oppure no (ad esempio se creo un database vuoto
- In genere, il primo passo è la creazione di un DB, successivamente si creano le tabelle (con i vari campi), quindi si possono inserire i dati, che potranno poi essere modificati, cancellati o ricercati. Nel seguito si illustrano tutte queste possibilità nell'ordine indicato.

Definizione di Schema (creazione di un Database)

- SQL consente la definizione di uno **schema di base di dati** come insieme di tabelle, domini, indici, asserzioni, viste e privilegi.
- In parole povere, uno schema di DB può essere inteso come un database, composto di tabelle; in realtà, dietro le quinte, un DB non è solo composto dalle tabelle e dai dati in esse contenuti, ma sono presenti anche:
  - **Domini** (ossia gli eventuali tipi di dato personalizzati usati nei campi delle tabelle del DB)
  - **Indici**, ossia alcuni campi di alcune tabelle, che sono segnalati al DBMS perchè verranno utilizzati (più di altri campi) per accedere ai dati (ad esempio "codice fiscale" nella tabella "clienti")
  - **Asserzioni**, che sono delle regole (specificate dall'amministratore) che il DB deve verificare per essere considerato corretto (ad esempio, nella tabella "articoli" di un negozio, deve essere presente almeno un articolo da vendere)
  - **Viste**, ossia descrizione di una parte del DB (ad esempio, la visione ridotta che del DB deve avere l'impiegato che lo usera')
  - **Privilegi**, ossia tutti i diritti di tutti gli utenti del DB

Definizione di Schema (creazione di un Database)

- Per creare uno schema di base di dati il comando è:

  - create schema [nomeschema] [[authorization] author] { elemento schema }

- Author rappresenta il nome del proprietario dello schema, che se non è specificato, coincide con colui che ha lanciato il comando.
- Il nomeschema è il nome del database. Se omesso, coincide con il nome del proprietario.
- Dopo questa istruzione compaiono le definizioni dei suoi componenti (tabelle ecc.), che possono comunque essere date anche in una fase successiva.
- La versione semplificata del comando quindi è:
  - *Create schema <nome_database>*
- A seconda del DBMS utilizzato, la sintassi di questo e di altri comandi potrebbe variare, ad esempio in MySQL la sintassi è
  - *create database <nome_database>*

Creazione di Tabelle

create table *Nometabella* (
NomeAttributo Dominio [*Default*] [*Vincoli*],
NomeAttributo Dominio [*Default*] [*Vincoli*],
...)

Nometabella è obbligatorio, seguono i nomi dei singoli campi (attributi), ognuno caratterizzato da un certo tipo di dato (dominio), da eventuale valore di default e dai vincoli

Una tabella è inizialmente vuota, ed il creatore della tabella possiede tutti i privilegi su di essa e sui suoi dati.

Esempio:

```
create table Studente(
        Cognome character(20) not null,
        Nome character(20),
        Matricola character(6) unique )
```

Domini Elementari

- I valori dei campi (attributi) di ogni colonna di ogni tabella devono appartenere ad un insieme di possibili valori (dominio). Esistono vari tipi di dominio, elementare o composito.
- Il dominio coincide con il tipo di dato dei linguaggi di programmazione
- La scelta del dominio viene fatta in sede di progettazione logica, quando si stabilisce anche l'occupazione di memoria necessaria
- Segue adesso una panoramica sui vari tipi di dato disponibili:

- **character** [**varying**] [(*lunghezza*)] [**character set** *NomeSet*]
  - caratteri, stringhe a lunghezza fissa e variabile (in questo caso la *lunghezza* preceduta da **varying** indica il numero massimo di caratteri)
    - esempi:

character (20)

Stringa di 20 caratteri

char varying (500) character set Greek

Stringa di lunghezza variabile sino a 500 caratteri, set Greek

- **bit** [**varying**] [(*lunghezza*)]
  - valori 0 e 1. E' possibile definire la stringa di bit in maniera analoga alla stringa di caratteri.

Domini Elementari

numeric [(*Precisione* [, *Scala*])]
decimal [(*Precisione* [, *Scala*])]
integer
smallint

Valori in base decimale, interi e non in virgola fissa.
La *Precisione* indica il numero di cifre intere, *Scala* il numero di quelle decimali, ad esempio decimal(7.4) consente di rappresentare numeri compresi tra
-999.9999 e +999.9999.
La differenza fra numeric e decimal è che il funzionamento di decimal può dipendere dall'implementazione interna del tipo di dato.
Integer e Smallint non presentano vincoli sulla rappresentazione, il che significa che il loro funzionamento può dipendere dall'implementazione interna e quindi anche dal tipo di macchina. La differenza fra i due è che smallint consente un numero di cifre inferiore a quello di integer.

Domini Elementari

float [(*Precisione*)]
real
double precision

Tipi per la rappresentazione di numeri in virgola mobile in notazione esponenziale (scientifica), ad esempio, 0.17E16 rappresenta il numero $1{,}7 \times 10^{15}$, dove 1,7 è detto *mantissa,* mentre 15 è l'esponente. Al dominio float può essere associata una precisione che rappresenta il numero di cifre relative alla mantissa, mentre per gli altri due tipi il funzionamento può dipendere dall'implementazione interna e quindi anche dal tipo di macchina. La differenza fra i due è che real consente un numero di cifre inferiore a quello di double precision.

Domini Elementari

Date
time [(*Precisione*)] [**with time zone**]
timestamp [(*Precisione*)] [**with time zone**]

Tipi per la rappresentazione di istanti temporali.
I domini sono tutti strutturati, ossia composti da più campi. Date contiene i campi year, month e day, time contiene hour, minute e second, mentre timestamp li contiene tutti, da year a second.
È possibile fissare una precisione per time e timestamp che indica il numero di cifre decimali dopo i secondi, se non specificato il valore di *default* è 2 per time (centesimo) e di 6 per timestamp (microsecondo).
Qualora venisse richiesto l'uso di time zone, per time e timestamp si possono usare altri due campi, timezone_hour e timezone_minute, che indicano lo scarto esistente con il tempo UTC (l'ora di Greenwich), ad esempio 21:03:04+1:00 è un istante riferito all'ora italiana (GMT+1).

Specifica valori attributi

Definizione dei domini.

Per specificare ogni dominio di ogni attributo (ossia l'insieme dei valori ammessi come validi per il medesimo), di possono utilizzare i domini elementari introdotti in precedenza, oppure se ne possono creare di nuovi con il comando **create domain**.

Definizione dei valori di default.

Consente di specificare il valore che viene automaticamente inserito se non viene esplicitamente inserito un valore. Se non si specifica un valore di default, esso è NULL, altrimenti si usa il comando **default**:

default *GenericoValore*
default user
default null

Nel primo caso si può inserire il valore desiderato (ad esempio, 0 per un integer), nel secondo il valore di default è il nome dell'utente, nel terzo caso è null.

Definizione dei vincoli

Esistono in SQL un gruppo di vincoli predefiniti, che modellano le situazioni più comuni, ed altri che possono essere definiti dall'utente (illustrati nel seguito).
Nell'ambito dei vincoli predefiniti, i più semplici, di tipo ***intrarelazionale*** sono:

► **Not null**, che impedisce l'uso del valore null (obbliga chi immetterà i dati a non lasciare il campo vuoto);

► **unique**, che impone che quell'attributo non presenti duplicati (eccezion fatta per il valore null);

► **not null unique**, combinazione dei due vincoli precedenti;

► **primary key**, per la definizione della chiave primaria, che non può essere nulla e deve essere unica. La chiave primaria può essere costituita anche da più colonne. Esempio:

Nome char(20) primary key
Cognome char(20)
Dipartimento char(15)
primary key (Nome, Cognome)

Definizione dei vincoli

Il più semplice vincolo di tipo ***interrelazionale*** è quello di integrità referenziale, che stabilisce un legame fra i valori di un attributo della tabella corrente (*interna*) e i valori di un attributo di un'altra tabella (*esterna*).
Il vincolo impone che ogni riga della tabella interna abbia un valore, per quell'attributo, che sia presente nella tabella esterna. Per quest'ultima, tale attributo deve essere **unique**, ossia deve essere identificativo per la tabella esterna stessa. Solitamente, l'attributo della tabella esterna riferito è la chiave primaria di tale tabella.

Definizione dei vincoli

Esempio di vincolo interrelazionale:

Table Multe

CodVerbale	**integer**	**primary key**
Nome	**char(20)**	
Cognome	**char(20)**	
Info	**char(15)**	

Il vincolo:

foreign key (Nome, Cognome) references Vigili (N, C)

Esprime il fatto che una multa deve essere fatta da un vigile che esiste in **Vigili**.

Se si dovesse violare questo vincolo, in fase di inserimento della multa il motore del DB segnalerà un errore

La tabella vigili riferita è:

Table Vigili

N	**char(20)**
C	**char(20)**
Indirizzo	**char(40)**

primary key (Nome, Cognome)

Definizione dei vincoli

Mentre per tutti gli altri vincoli, una loro violazione genera un errore, per l'integrità referenziale, possiamo scegliere le reazioni da adottare.
Variazioni sulla tabella interna (quella che contiene il riferimento) non ci sono particolari problemi, mentre una violazione del contenuto della tabella esterna (la *master*), è possibile modificare un valore della chiave (riferita dalla tabella interna), allora si può :

- propagare alla tabella interna la variazione (**cascade**)
- sganciare la tab. interna, settando null al riferimento (**set null**)
- sganciare la tab. interna, settando il valore di default (**set default**)
- impedire la modifica sulla tabella esterna (**no action**)

cancellare una riga, quindi un valore della chiave. Allora, si può procedere come prima, soltanto che il primo caso comporterebbe la cancellazione, nella tabella interna, di tutte le righe che si riferiscono alla chiave cancellata nella tabella esterna.
Il **cascade** sottintende uno stretto legame logico fra le tabelle, gli altri casi, sganciandole, mostrano un legame più blando.

Definizione dei vincoli

La specifica della reazione si scrive dopo il vincolo di integrità, esempio:

Table Multe
CodVerbale integer primary key
Nome char(20)
Cognome char(20)
Info char(15)
foreign key (Nome, Cognome) references Vigili (N, C)
on delete set null
on update cascade

Vincoli Generici

Oltre i vincoli predefiniti, l'SQL consente di definirli in modo personalizzato, precisamente:

check (*condizione* **)**

La condizione può essere anche complessa, ad esempio:

```
create table impiegato (
        Matricola character(6)
        Superiore char(6)
              check (    Matricola like "1%" or
                         Dip =
                         (select Dip from imp as i
                         where i.Matricola= Superiore)
```

il vincolo controlla che un impiegato abbia un superiore del proprio dipartimento, a meno che il numero di matricola non cominci per 1.

Vincoli Generici

Asserzioni

L'asserzione è un vincolo generico che non è associato ad un attributo particolare, ma a tutto lo schema del db. La sintassi è:

create assertion *nomeasserzione* **check** *vincolo*

Ad esempio, per imporre che nella tabella impiegato sia presente almeno una riga, si ha:

create assertion almenouno check
((select count (*) from impiegato) >=1)

Verifica dei vincoli

I vincoli, normali o asserzioni, possono essere verificati in maniera immediata o differita, essendo nel primo caso il controllo effettuato immediatamente ogni volta che si fa una modifica al db, mentre i secondi sono verificati al termine dell'esecuzione di una serie di operazioni (transazione). Per scegliere la modalità si usa la sintassi:

set constraints [*nomevincolo*] **immediate | deferred**

Alterazione schema

Tabelle, Attributi, Domini possono essere ***modificati*** mediante comandi:
alter domain permette l'alterazione di domini (ad esempio, vincoli o default);
alter table permette l'alterazione di tabelle (ad esempio, vincoli o default);
drop consente di rimuovere dal db uno dei suoi componenti.

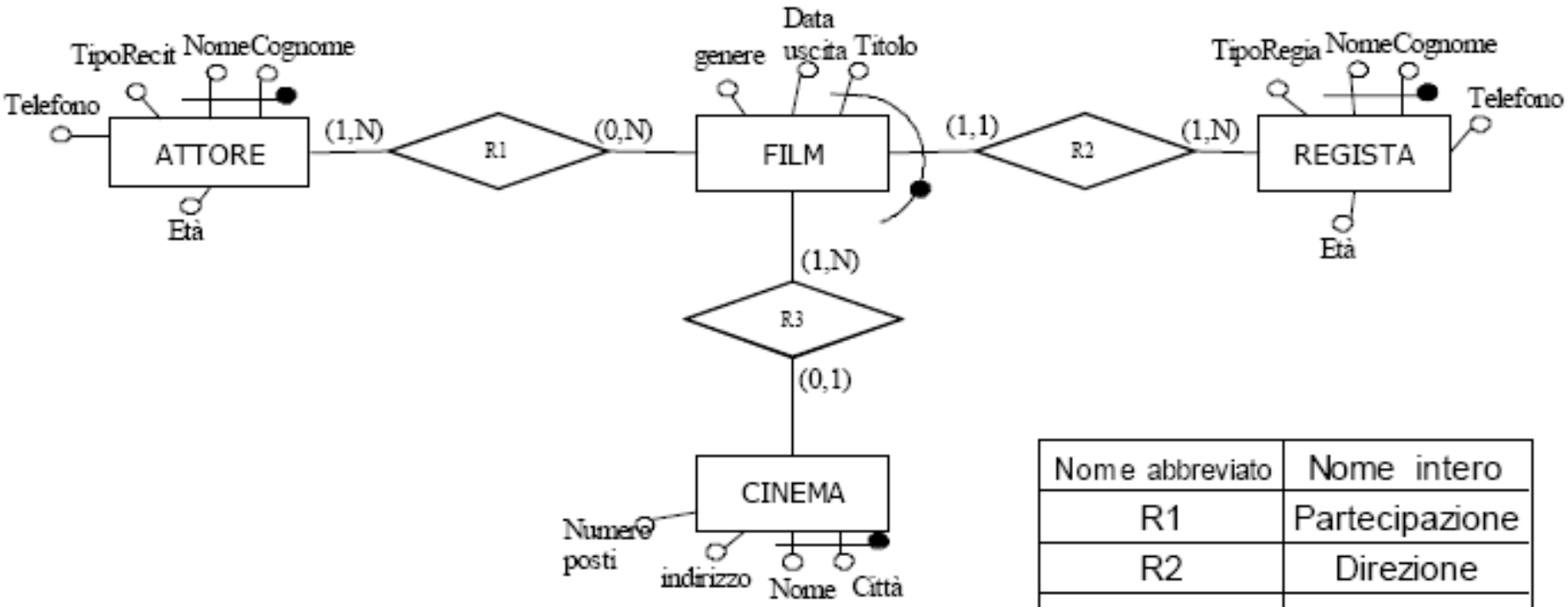
Catalogo

Tutte le informazioni relative allo schema del database, esso le memorizza in alcune tabelle interne, che si aggiungono alle tabelle del db. Tali tabelle interne prendono il nome di ***Catalogo***, che contiene i *metadati* (dati che rappresentano informazioni relative ad altri dati).

Table_Name	*Col_Name*	*Position*
Impiegato	**Nome**	**0**
Impiegato	**Cognome**	**1**
Impiegato	**Dipartimento**	**2**

Creazione DB - Esempi

Riprendendo gli schemi E-R e relazionali già visti:



Nome abbreviato	Nome intero
R1	Partecipazione
R2	Direzione
R3	Proiezione

Creazione DB - Esempi

Riprendendo gli schemi E-R e relazionali corrispondenti già visti:

Attore (Nome, Cognome, tipoR, tel, eta)

Film(titolo, nomeR, cognR, datauscita, genere)

Partecipazione(nA, cA, titolofilm, nR, cR)

Regista(nome, cognome, tipoR, tel, eta)

Cinema(nome, citta, indirizzo, posti, titolo*, nomeR*, cognR*)

Creazione DB - Esempi

I comandi SQL per creare il database e le tabelle vuote sono:

```
create schema programmazione_cinematografica
create table Attore (
        Nome character(20) not null,
        Cognome character(20) not null,
        tipoR character(15) default "drammatico"
        tel character(10)
        eta numeric(2) not null
        primary key(Nome, Cognome))
Create table Film(
        titolo character(40) not null unique,
        nomeR character(20),
        cognR character(20),
        datauscita date,
        genere character(15) default "drammatico",
        unique(nomeR, cognR),
        foreign key(nomeR, cognR) references Regista(nome, cognome)
                on update cascade on delete no action,
        primary key(titolo, nomeR, cognR))
```

Creazione DB - Esempi

...ancora:

```
Create table partecipazione(
        nA character(20),
        cA character(20),
        titolofilm character(40) unique,
        nR character(20),
        cR character(20),
        unique (nA, cA),
        foreign key(nA, cA) references Attore(Nome, Cognome)
                on update cascade on delete no action
        unique(titolofilm, nR, cR)
        foreign key(titolo film, nR cR)
                references Film(titolo, nomeR, cognR)
                on update cascade on delete no action
        primary key(nA,cA, titolofilm, nR, cR))
```

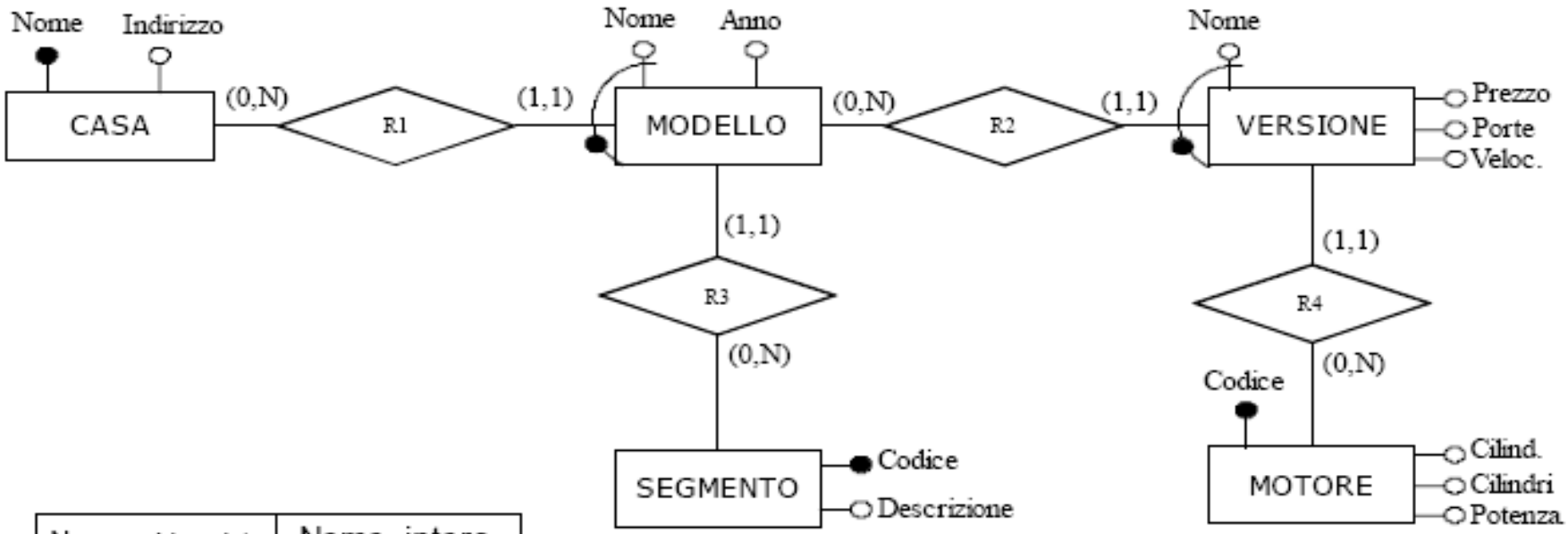
Creazione DB - Esempi

...e infine:

```
create table Regista (
        Nome character(20) not null, Cognome character(20) not null,
        tipoR character(15) default "drammatico", tel character(10),
        eta numeric(2) not null,
        primary key(Nome, Cognome))
Create table Cinema(
        Nome character(30) not null,        citta character(30) not null,
        primary key(Nome, citta),
        indirizzo character(40) default null,        posti numeric(3),
        titolo character(40) not null unique,
        nomeR character(20), cognR character(20),
        unique(titolo, nomeR, cognR),
        foreign key(titolo, nomeR, cognR)
                references Film(titolo, nomeR, cognR)
                on update cascade on delete no action    )
```

Creazione DB - Esempi

Riprendendo gli schemi E-R e relazionali già visti:



Nome abbreviato	Nome intero
R1	Produce
R2	Composto
R3	Localizzato
R4	Composta

Creazione DB - Esempi

Riprendendo gli schemi E-R e relazionali già visti:

Casa(Nome, indirizzo)

Modello(Nome, Nomecasa, anno, segmento*)

Segmento(codice, descrizione)

Versione(nomeV, nomeM, nomeC, prezzo, porte, vel, motore*)

Motore(codice, cilindrata, numcil, potenzaCV)

Creazione DB - Esempi

I comandi SQL per creare il database e le tabelle vuote sono:

```
Create database automobili
Create table casa(
        Nome character(20) not null primary key,
        via character (40),
        numero_civico numeric(4),
        CAP numeric(5))
Create table Modello(
        Nome character(10) not null,
        Nomecasa character(20) not null references casa(Nome)
                on update cascade on delete no action,
        primary key (Nome, Nomecasa),
        anno date,
        segmento character(6) not null references Segmento(codice)
                on update cascade on delete set null
)
```

Creazione DB - Esempi

... ancora:

```
Create table Segmento(
        codice character(6) not null primary key,
        descrizione character(20) default NULL
)
Create table Versione(
        nomeV character(10) not null,
        nomeM character(10) not null,
        nomeC character(20) not null,
        foreign key(nomeM, nomeC) references Modello(Nome, Nomecasa)
                on update cascade on delete no action,
        primary key(nomeV, nomeM, nomeC),
        prezzo numeric(7.2) default 0,
        porte numeric(1) default 3,
        vel numeric(3),
        motore character(6) not null references Motore(codice)
                on update cascade on delete set null      )
```

Creazione DB - Esempi

… e infine:

```
Create table Motore(
        codice character(6) not null primary key,
        cilindrata numeric(4),
        numcilindri numeric(1),
        potenzaCV numeric(4),
)
```

SQL – Operazioni sui dati

L'SQL permette le quattro operazioni fondamentali sui dati:

- inserimento
- modifica
- cancellazione
- lettura (ricerca)

SQL – Operazioni sui dati

Inserimento

Il comando è:

insert into *nometabella* [*listaattributi*] < **values (** *listavalori*) | *selectSQL* >

La forma con values permette l'inserimento di singole righe, specificando la *listavalori* delle colonne, ad esempio **insert into Dipart (Dip, Citta) values ('produzione', 'torino')**.

La seconda forma con *selectSQL* significa che tale select recupera da una qualche tabella un certo insieme di righe, che vengono tutte copiate dentro la tabella *nometabella*, ad esempio **insert into Dipart (select * from dept where country='Italy')**.

Generalmente la prima forma viene usata dagli utenti, che immettono i dati uno alla volta, o comunque si usa se i dati non esistono ancora, mentre la seconda si usa perchè i dati sono già presenti, e si desidera ricopiarli in un'altra tabella.

Per entrambe le forme, deve in ogni caso esistere una corrispondenza ordinata degli attributi (il numero di attributi in *listaattributi* deve essere pari al numero di valori in *listavalori* o al numero di colonne della relazione restituita da *selectSQL*) e deve anche esserci coerenza fra i domini.

SQL – Operazioni sui dati

Cancellazione

Il comando è:

delete from *nometabella* [**where** *condizione*]

se il where non esiste, il delete cancella tutte le righe della tabella (ma non la tabella stessa, che resterà vuota, se si vuole cancellare la tabella, si deve usare il **drop**).

Occorre cautela se delete cancella righe con riferimenti, in quanto potrebbero essere violati i vincoli di integrità referenziale.

Esempio:

delete from Dipartimenti where Dip='Produzione'.

SQL – Operazioni sui dati

Aggiornamento

Il comando è:

update *nometabella*
set *attributo1* **= <** *espressione* | *selectSQL* / **default | null** >
{ , **set** *attributo2* **= <** *espressione* | *selectSQL* / **default | null** >
… }
[**where** *condizione*]

se il where non compare, l'update si effettua per tutte le righe. Un esempio è:

update Impiegati set Stipendio = Stipendio + 100
where Dipart='Produzione'

SQL – Operazioni sui dati

Occorre prestare attenzione agli aggiornamenti, in quanto ad esempio, la sequenza:

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

update Impiegato set Stipendio= Stipendio*1.2
       where Stipendio>30
```

può creare problemi. Se esistesse infatti uno stipendio di 28 milioni, soddisfa il primo predicato e subisce l'aumento, diventando superiore a 30 milioni. A questo punto si esegue il secondo aggiornamento, e lo stesso stipendio soddisfa anche questo secondo predicato, per cui l'impiegato prende la somma dei due aumenti.

Per ovviare problemi di questo tipo, si possono invertire le query o adottare dei controlli opportuni, ad esempio usando SQL nell'ambito di un linguaggio di programmazione ad alto livello.

SQL – Interrogazioni

SQL esprime le interrogazioni in maniera *dichiarativa* (in contrapposizione al comportamento *procedurale* dell'algebra relazionale), pertanto quello che accade è che un'interrogazione SQL viene passata ad un motore sottostante che si occupa di convertire la dichiarazione in una procedura da eseguire.
La conversione è di fatto nascosta all'utente, garantendo una certa astrazione.
Esistono anche linguaggi per interrogazione di natura procedurale, nei quali però occorre una maggiore conoscenza del sistema per specificare *come* un'operazione vada fatta.

SQL – Interrogazioni

Le operazioni di interrogazione (query) vengono fatte tramite **select**:
select *listaattributi* from *listatabelle* [where *condizione*]

listaattributi è nota come target list
from *listatabelle* è nota come clausula from
where *condizione* è nota come clausula where

Data una interrogazione in linguaggio naturale, i passi sono:

1) Identificare COSA VUOLE, che diventa la listaattributi
2) Identificare DA QUALI TABELLE prendere le cose del punto 1, e questa sarebbe la listatabelle
3) Identificare CON CHE CRITERIO prendere le cose del punto 1, o in altre parole CHE INFORMAZIONI HA, e questo va nel where

Esempio

Stampare l'elenco degli indirizzi degli impiegati con piu' di 20 anni di servizio

Impiegati(ID, nome, indirizzo, anzianità)

Cosa vuole? "l'elenco degli indirizzi degli impiegati "

Da dove prenderlo? Tabella impiegati (potrebbero essere più tabelle)

Criterio? "con piu' di 20 anni di servizio"

Select indirizzo from impiegati where anzianità>20

SQL – Interrogazioni

Caratteristiche delle query

Nella ***target list*** può comparire * se si vogliono tutte le colonne, o anche una generica espressione, ad esempio

select Stipendio/12 as StipendioMensile
from StipendioAnnuale where Cognome="Bianchi"

"as StipendioMensile" è una ridenominazione del risultato consentita dalla select.
Nella target list si devono specificare anche i nomi delle tabelle, se esistono nomi uguali su tabelle diverse.

La ***clausula from***, che di fatto opera il join esterno full (prodotto cartesiano) fra le tabelle specificate, può anche contenere una sola tabella, se serve estrarre i dati solo da quella.

SQL – Interrogazioni

Caratteristiche delle query

La ***clausula where*** può contenere una generica espressione con l'uso di operatori <, >, <=, >=, <>, =, AND, OR, NOT ed anche altri operatori specifici ad esempio per l'estrazione di stringhe e/o sottostringhe ('_' per il singolo carattere e '%' per una sequenza di caratteri, usati insieme all'operatore LIKE), o per il controllo di nullità (IS NULL, IS NOT NULL).

Per quanto riguarda la ***gestione dei valori nulli***, l'esempio **stipendio>40** è vero o falso se uno stipendio è null?
SQL-89 considera il predicato falso, mentre SQL-2 restituisce il valore *unknown*, adottando di fatto una logica a tre valori. Questo richiede molta attenzione, specie se le espressioni sono complesse.

Esempi di query

```
SQL> SELECT *
  2  FROM   dept;
```

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Selezionare tutte le colonne dalla tabella dept

Selezionare una particolare colonna

```
SQL> SELECT deptno, loc
  2  FROM   dept;
```

```
    DEPTNO LOC
---------- -------------
        10 NEW YORK
        20 DALLAS
        30 CHICAGO
        40 BOSTON
```

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Espressioni Aritmetiche

- Creare espressioni attraverso l'uso di operatori aritmetici.

Operatore	Descrizione
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione

Uso degli operatori Aritmetici

```
SQL> SELECT ename, sal, sal+300
  2  FROM   emp;
```

```
ENAME                SAL        exp
---------- --------- ---------
KING                5000       5300
BLAKE               2850       3150
CLARK               2450       2750
JONES               2975       3275
MARTIN              1250       1550
ALLEN               1600       1900
...
14 rows selected.
```

Precedenza Operatori

```
SQL> SELECT ename, sal, 12*sal+100
  2  FROM   emp;
```

```
ENAME                SAL exp
---------- --------- ----------
KING                5000      60100
BLAKE               2850      34300
CLARK               2450      29500
JONES               2975      35800
MARTIN              1250      15100
ALLEN               1600      19300
...
14 rows selected.
```

Uso delle parentesi

```
SQL> SELECT ename, sal, 12*(sal+100)
  2  FROM   emp;
```

```
ENAME                SAL exp
---------- --------- ------------
KING                5000        61200
BLAKE               2850        35400
CLARK               2450        30600
JONES               2975        36900
MARTIN              1250        16200
...
14 rows selected.
```

Alias

Si può associare uno o più nomi fittizi ad ogni tabella o attributo, questo per potere usare delle abbreviazioni o anche per potere esprimere calcoli complessi in forma più semplice, ad esempio:

```
select I1.Cognome, I1.Nome
        from Impiegati as I1, Impegati as I2
        where I1.Cognome=I2.Cognome
                and I1.Nome<>I2.Nome
                and I2.Dipart="produzione"
```

La precedente interrogazione consente di confrontare una tabella con se stessa definendone due alias, estraendo gli impiegati con uguale cognome e nome diverso e che appartengano al dipartimento di produzione.

Alias delle colonne

- Rinominare il nome di una colonna
- Utile con dei calcoli
- Deve seguire immediatamente il nome di una colonna (SENZA VIRGOLA); può essere usata opzionalmente la parola chiave AS tra il nome della colonna e l'alias.
- Richiede doppio apice se l'alias ha degli spazi

Uso dell'Alias

```
SQL> SELECT ename AS name, sal salary
  2  FROM   emp;
```

```
NAME                 SALARY
-------------------- ---------

...
```

```
SQL> SELECT ename "Name",
  2         sal*12 "Annual Salary"
  3  FROM   emp;
```

```
Name                 Annual Salary
-------------------- -------------

...
```

Alias o Correlation Names

Esami

CORSO	MATRICOLA	VOTO
Programmazione	345678	27
Architetture	123456	30
Programmazione	234567	18
Matematica Discreta	345678	22
Architettura	345678	30

Corsi

CORSO	PROFESSORE
Programmazione	Ferro
Architetture	Pappalardo
Matematica Discreta	Lizzio

SELECT *Professore*
FROM Corsi as p, Esami as e
WHERE p.Corso = e.Corso
AND Matricola = '123456'

Professori con cui 123456 ha fatto esami

Altri usi degli alias

```
SELECT p1.Professore
FROM Corsi as p1,Corsi as p2, Esami as e1, Esami as e2
WHERE     p1.Corso = e1.Corso AND
          p2.Corso = e2.Corso AND
          p1.Professore = p2.Professore AND
          e1.Matricola = e2.Matricola AND
          NOT p1.Corso = p2.Corso.
```

- Professori che hanno fatto esami in almeno 2 corsi diversi allo stesso studente.

Duplicati

In una interrogazione SQL, in generale nei risultati sono ***ammessi i duplicati*** (righe uguali)
Esiste in SQL la possibilità di richiedere esplicitamente che un risultato di un'interrogazione venga privato (**distinct**) o meno (**all**) di eventuali duplicati presenti, ad esempio

select distinct Nome from Impiegati Where Cognome="Rossi"

Se esistono tre Mario Rossi ed un Antonio Rossi, il "distinct" farà ottenere solo due righe in risposta alla query, ossia Mario e Antonio

Se NON si usa distinct (o si mette ALL), si otterrebbero quattro righe, ossia i tre "Mario" e "Antonio" (manterrebbe i duplicati)

Righe duplicate

- Le righe duplicate sono restituite per default

```
SQL> SELECT deptno
  2  FROM   emp;
```

```
    DEPTNO
---------
        10
        30
        10
        20
...
14 rows selected.
```

Eliminazione delle righe duplicate

E' consentito dall'uso della parola chiave DISTINCT nella clausola SELECT

```
SQL> SELECT DISTINCT deptno
  2  FROM   emp;
```

```
    DEPTNO
---------
        10
        20
        30
```

Clausula WHERE: Restrizioni ed ordinamento

IMPIEGATI

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

"...selezionare tutti gli impiegati del dipartimento 10"

IMPIEGATI

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7782	CLARK	MANAGER		10
7934	MILLER	CLERK		10

Limitare le righe selezionate

- Limitare le righe tramite l'uso della clausola WHERE.

```
SELECT      [DISTINCT] {*| colonna [alias], ...}
FROM        tabella
[WHERE      condizione(i)];
```

- La clausola WHERE segue la clausola FROM.

Uso della clausola WHERE

```
SQL> SELECT ename, job, deptno
  2  FROM   emp
  3  WHERE  job='CLERK';
```

ENAME	JOB	DEPTNO
JAMES	CLERK	30
SMITH	CLERK	20
ADAMS	CLERK	20
MILLER	CLERK	10

Stringhe di caratteri e Date

- Stringhe di caratteri e le date vanno incluse tra apici.
- I caratteri sono case sensitive e le date sono format sensitive.

```
SQL> SELECT ename, job, deptno
  2  FROM   emp
  3  WHERE  ename = 'JAMES';
```

Operatori di confronto

Operatore	Significato
=	Uguale a
>	più grande di
>=	maggiore o uguale di
<	minore di
<=	minore o uguale a
<>	diverso

Uso degli Operatori di Confronto

```
SQL> SELECT ename, sal, comm
  2  FROM   emp
  3  WHERE  sal<=comm;
```

```
ENAME          SAL       COMM
---------- ---------- ----------
MARTIN         1250 <--> 1400
```

Altri Operatori di Confronto

Operatore	Significato
BETWEEN ...AND...	compreso tra due valori
IN(list)	Corrisp. ad uno dei valori nella lista
LIKE	Operatore di pattern matching
IS NULL	Valore nullo

Uso dell'operatore BETWEEN

- BETWEEN consente la selezione di righe con attributi in un particolare range.

```
SQL> SELECT   ename, sal
  2  FROM     emp
  3  WHERE    sal BETWEEN 1000 AND 1500;
```

ENAME	SAL
MARTIN	1250
TURNER	1500
WARD	1250
ADAMS	1100
MILLER	1300

Limite inferiore

Limite superiore

Predicato Between AND

- Espr1 [NOT] BETWEEN Espr2 AND Espr3.

- Equivale a
- [NOT] Espr2 ≤Espr1 AND Espr1≤Espr3

Uso dell'operatore IN

- E' usato per selezionare righe che hanno un attributo che assume valori contenuti in una lista.

```
SQL> SELECT   empno, ename, sal, mgr
  2  FROM     emp
  3  WHERE    mgr IN (7902, 7566, 7788);
```

EMPNO	ENAME	SAL	MGR
7902	FORD	3000	7566
7369	SMITH	800	7902
7788	SCOTT	3000	7566
7876	ADAMS	1100	7788

Uso dell'operatore LIKE

- LIKE è usato per effettuare ricerche *wildcard* di una stringa di valori.
- Le condizioni di ricerca possono contenere sia letterali, caratteri o numeri.
  - % denota zero o più caratteri.
  - _ denota un carattere.

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE 'S%';
```

Uso dell'operatore LIKE

- Il pattern-matching di caratteri può essere combinato.

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE '_A%';
```

```
ENAME
----------
MARTIN
JAMES
WARD
```

- I'identificatore ESCAPE (\) deve essere usato per cercare "%" o "_".

Operatori di Match

- *Attributo* [NOT] LIKE *Stringa*

- Dove *Stringa* puo' contenere anche:
- "_" che fa "match" con qualunque carattere
- "%" che fa match con qualunque sequenza di caratteri
- vale U se l'attributo e' NULL

Esempio

- SELECT Nome
- FROM Studenti
- WHERE Indirizzo LIKE “Via Etnea %”

- Fornisce tutti gli studenti che abitano in Via Etnea

Predicati

- Espr IS [NOT] NULL

- esempio:

```
SELECT Nome
FROM Studenti
WHERE Telefono IS NOT NULL
```

Operatori Logici

Operatore	Significato
AND	Restituisce TRUE if *entrambe* le condizioni sono TRUE
OR	Restituisce TRUE se *almeno* una delle condizioni è TRUE
NOT	Restituisce TRUE se la condizione è FALSE

Uso dell'operatore AND

AND richiede entrambe le condizioni TRUE.

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  AND    job='CLERK';
```

```
     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7876 ADAMS      CLERK            1100
      7934 MILLER     CLERK            1300
```

Uso dell'operatore OR

OR richiede almeno una condizione TRUE.

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  OR     job='CLERK';
```

```
     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7839 KING       PRESIDENT       5000
      7698 BLAKE      MANAGER         2850
      7782 CLARK      MANAGER         2450
      7566 JONES      MANAGER         2975
      7654 MARTIN     SALESMAN        1250
      ...
      7900 JAMES      CLERK            950
      ...
14 rows selected.
```

Uso dell'operatore NOT

```
SQL> SELECT ename, job
  2  FROM   emp
  3  WHERE  job NOT IN ('CLERK','MANAGER','ANALYST');
```

```
ENAME      JOB
---------- ---------
KING       PRESIDENT
MARTIN     SALESMAN
ALLEN      SALESMAN
TURNER     SALESMAN
WARD       SALESMAN
```

Regole di precedenza

Ordine di val.	Operatore
1	Tutti gli operatori di confronto
2	NOT
3	AND
4	OR

- L'override delle regole di precedenza è ottenuto con l'uso delle parentesi.

Regole di precedenza

```
SQL> SELECT ename, job, sal
  2  FROM   emp
  3  WHERE  job='SALESMAN'
  4  OR     job='PRESIDENT'
  5  AND    sal>1500;
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
MARTIN	SALESMAN	1250
ALLEN	SALESMAN	1600
TURNER	SALESMAN	1500
WARD	SALESMAN	1250

Regole di precedenza

L'uso delle parentesi forza la priorità

```
SQL> SELECT   ename, job, sal
  2  FROM     emp
  3  WHERE    (job='SALESMAN'
  4  OR       job='PRESIDENT')
  5  AND      sal>1500;
```

```
ENAME      JOB              SAL
---------- --------- ----------
KING       PRESIDENT      5000
ALLEN      SALESMAN       1600
```

Ordinamento

Nei database in genere non è significativo l'ordine, cosa che invece in SQL (mondo reale) potrebbe essere, per cui si prevede dopo il where la clausula

order by *nomeattributo1* [asc | desc] { , *nomeattributo2* [asc | desc] ... }

per restituire una tabella ordinata secondo i valori di una certa colonna, crescente o decrescente, proseguendo l'ordinamento secondo altre colonne, se desiderato.

ORDINAMENTO

- ORDER BY Attributo [DESC] {, Attributo [DESC] }
- Va posto dopo il WHERE e fa si che il risultato sia ordinato secondo Attributo in senso crescente mentre se lo si vuole decrescente si deve aggiungere DESC

Esempio

```
SELECT e.Corso, e.Voto
FROM Esami e, Studenti s
WHERE e.Matricola = s.Matricola
                AND s.Nome = 'Mario Rossi'
ORDER BY Voto DESC, s.nome
```

Ordina dai più bravi in poi (voto maggiore, ordinamento decrescente), alfabeticamente (a parità di voto, si considera il nome, crescente)

Estrarre dati da più tabelle: Join

Join Naturale

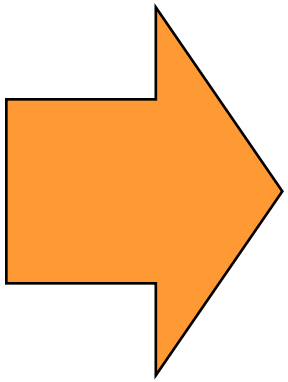
il Join naturale unisce due relazioni aventi entrambe un certo numero di colonne comuni, generando una terza relazione che contiene tutte le colonne delle due relazioni, e tutte le righe delle due relazioni che hanno valori uguali nelle colonne (attributi) comuni. Se si escludono le colonne non comuni della seconda relazione, si parla di ***semi-join***.

R

A	B	C
1	2	3
4	5	6
7	8	9

S

C	E
3	1
6	2
6	8
7	2



R ▷◁ S

A	B	C	E
1	2	3	1
4	5	6	2
4	5	6	8

Esempio di join fra due tabelle

NOME	MATRICOL	INDIRIZZO	TELEFONO
Mario Rossi	123456	Via Etnea 1	222222
Ugo Bianchi	234567	Via Roma 2	333333
Teo Verdi	345678	Via Enna 3	444444

CORSO	MATRICOLA	VOTO
Programmazione	345678	27
Architetture	123456	30
Programmazione	234567	18
Matematica Discreta	345678	22
Architettura	345678	30

Quali esami ha superato Mario Rossi?

- SELECT *Corso* FROM *Esami,Studenti*
  WHERE *Esami.Matricola = Studenti.Matricola* AND Nome='Mario Rossi'

Architetture

Esempio di join fra tre tabelle

NOME	MATRICOL	INDIRIZZO	TELEFONO
Mario Rossi	123456	Via Etnea 1	222222
Ugo Bianchi	234567	Via Roma 2	333333
Teo Verdi	345678	Via Enna 3	444444

CORSO	PROFESSORE
Programmazione	Ferro
Architetture	Pappalardo
Matematica Discreta	Lizzio

CORSO	MATRICOLA	VOTO
Programmazione	345678	27
Architetture	123456	30
Programmazione	234567	18
Matematica Discreta	345678	22
Architettura	345678	30

Quali Professori hanno dato piu' di 24 a Teo Verdi ed in quali corsi?

SELECT *Professore, Corsi.Corso*
FROM *Corsi,Esami,Studenti*
WHERE *Corsi.Corso = Esami.Corso* AND
Esami.Matricola = Studenti.Matricola
AND Nome='Teo Verdi' AND Voto > 24

Ferro Programmazione
Pappalardo Architetture

Prodotto Cartesiano

STUDENTI

matricola	cognome
11	Paolino
12	Pluto
10	Mouse

FACOLTA

Facolta	stud
ing	10
let	11
ing	12

Il join nasce dal **prodotto cartesiano** di due relazioni.
Questo è una relazione che contiene tutte le possibili combinazioni di tuple prese da entrambe le relazioni, operazione *raramente significativa*.

STUDENTI x FACOLTA

matricola	cognome	Facolta	Stud
11	Paolino	ing	10
12	Pluto	ing	10
10	Mouse	ing	10
11	Paolino	let	11
12	Pluto	let	11
10	Mouse	let	11
11	Paolino	ing	12
12	Pluto	ing	12
10	Mouse	ing	12

Join

- Il join fra due tabelle opera anzitutto il prodotto cartesiano, escludendo però le righe della prima e seconda relazione che non hanno attributi comuni uguali (quindi "non accoppiabili").
- Il join quindi è un prodotto cartesiano "intelligente"
- Se si vogliono comunque includere le righe della prima, seconda, o entrambe, si usano i ***join sinistro, destro, o totale***.
- Quando sono coinvolte a qualunque titolo le righe con valori non coincidenti negli attributi comuni si parla di ***outer join*** (***join esterno***), altrimenti si parla di ***inner join*** (***join interno***)

Join

- Inner Join (a destra)
- Left Outer Join

$R \triangleright\!\!\triangleleft S$

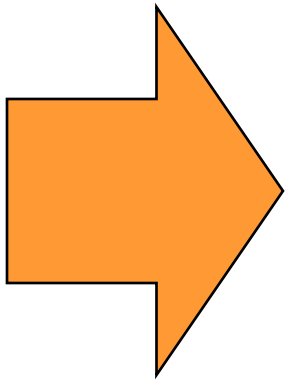
A	B	C	E
1	2	3	1
4	5	6	2
4	5	6	8

R

A	B	C
1	2	3
4	5	6
7	8	9

S

C	E
3	1
6	2
6	8
7	2



$R \triangleright\!\!\triangleleft S$ left

A	B	C	E
1	2	3	1
4	5	6	2
4	5	6	8
7	8	9	Null

Join

- Inner Join (a destra)
- Right Outer Join

$R \triangleright\!\!\triangleleft S$

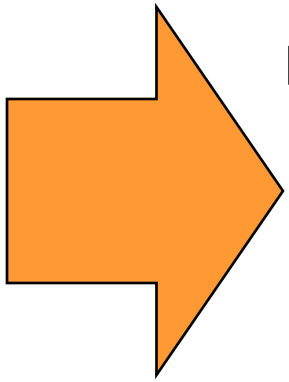
A	B	C	E
1	2	3	1
4	5	6	2
4	5	6	8

R

A	B	C
1	2	3
4	5	6
7	8	9

S

C	E
3	1
6	2
6	8
7	2



$R \triangleright\!\!\triangleleft S$ right

A	B	C	E
1	2	3	1
4	5	6	2
4	5	6	8
Null	Null	7	2

Join

- Inner Join (a destra)
- Full Outer Join

$R \rhd\lhd S$

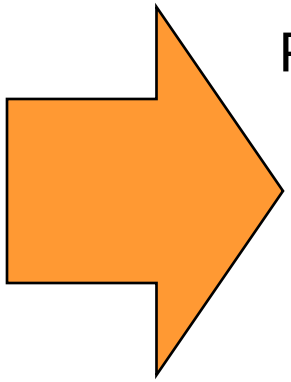
A	B	C	E
1	2	3	1
4	5	6	2
4	5	6	8

R

A	B	C
1	2	3
4	5	6
7	8	9

S

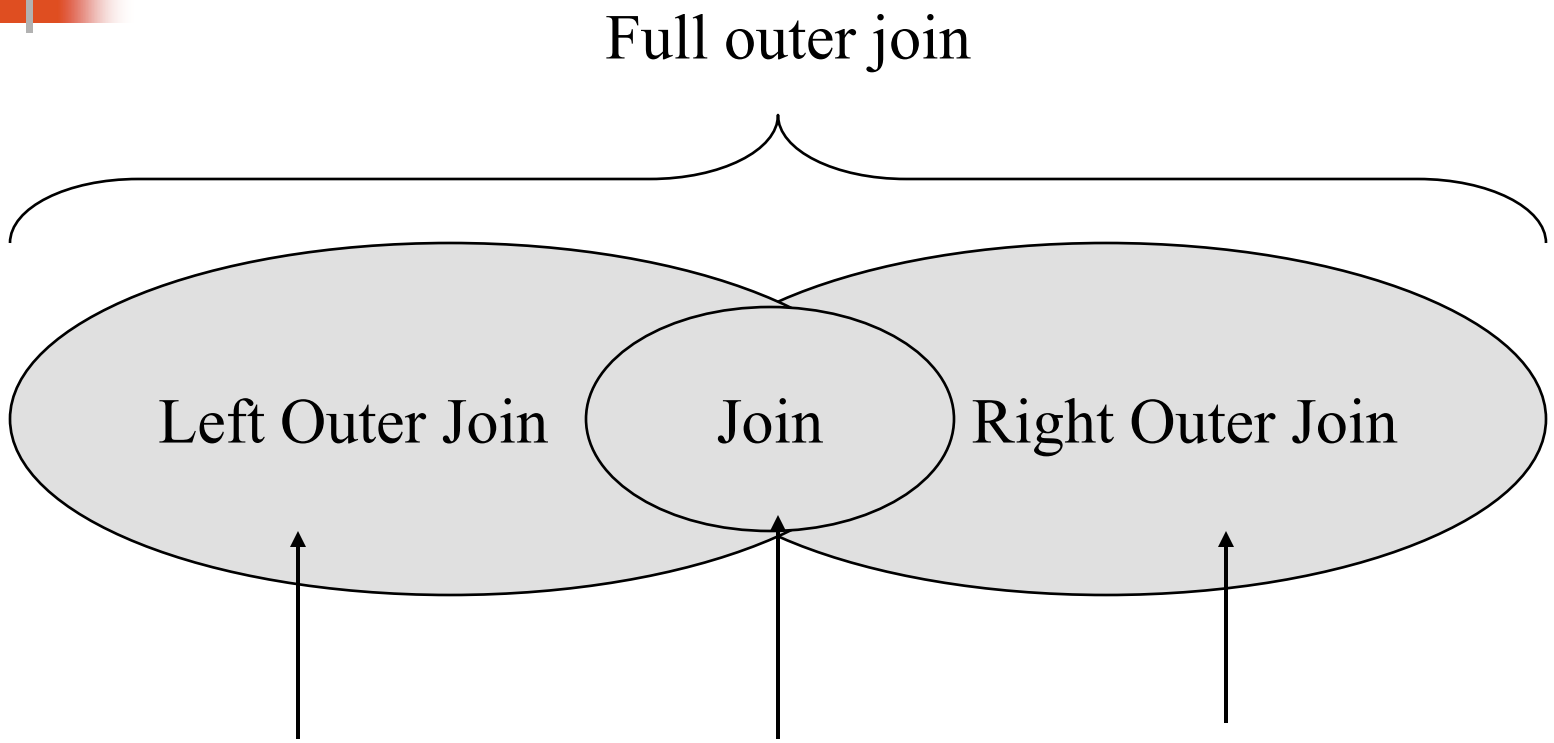
C	E
3	1
6	2
6	8
7	2



$R \rhd\lhd S$ full

A	B	C	E
1	2	3	1
4	5	6	2
4	5	6	8
7	8	9	Null
Null	Null	7	2

Outer Join Operators



Join

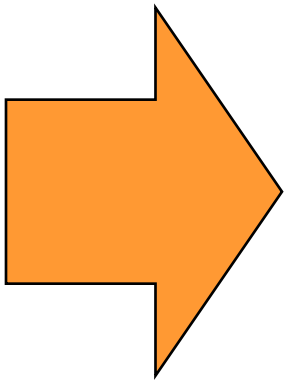
- **L'equi-join** si ha quando non esistono attributi comuni, ma si decide comunque di voler unire le due relazioni, specificando quali attributi (colonne) debbano essere uguali.
- Se l'espressione è più generale (non necessariamente una uguaglianza fra due attributi) , il join prende il nome di **theta-join**

R

Imp.	Progetto
Polda	H4
Versi	H4
Rossi	N6

S

Codice	Nome
H4	Marte
N6	Urano
G8	Giove



$R \bowtie_{progetto=codice} S$

Imp.	Progetto	Codice	Nome
Polda	H4	H4	Marte
Versi	H4	H4	Marte
Rossi	N6	N6	Urano

Esempio di Natural Join

► Natural Join

SELECT Studenti.Nome,Esami.Corso,Esami.Voto
FROM *Esami* NATURAL JOIN *Studenti*

► *Nome,Corso e Voto degli esami*

Esempio di join naturale

```
SQL> SELECT   emp.empno,   emp.ename, emp.deptno,
  2           dept.deptno, dept.loc
  3  FROM     emp, dept
  4  WHERE    emp.deptno=dept.deptno;
```

```
EMPNO ENAME      DEPTNO DEPTNO LOC
----- ------ ------ ------ ---------
 7839 KING         10     10 NEW YORK
 7698 BLAKE        30     30 CHICAGO
 7782 CLARK        10     10 NEW YORK
 7566 JONES        20     20 DALLAS
...
14 rows selected.
```

Altro Esempio di Join

- Agenti(CodiceAgente,Nome,Zona Supervisore,Commissione)
- Clienti(CodiceCliente,Nome,Citta',Sconto)
- Ordini(CodiceOrdine,CodiceCliente,CodiceAgente Articolo,Data,Ammontare)

```
SELECT Agenti.CodiceAgente,Ordini.Ammontare
FROM Agenti JOIN Ordini
ON Agenti.Supervisore = Ordini.CodiceAgente
```

- Codice agente ed ammontare degli ordini dei supervisori

```
SELECT Agenti.CodiceAgente,Ordini.Ammontare
FROM Agenti NATURAL LEFT JOIN Ordini
```

- Codice agente ed ammontare degli agenti incluso quelli che non hanno effettuato ordini (avranno ammontare NULL)

Condizioni di Ricerca addizionali
Uso dell'operatore AND

EMP

EMPNO	ENAME	DEPTNO
7839	KING	10
7698	BLAKE	30
7782	CLARK	10
7566	JONES	20
7654	MARTIN	30
7499	ALLEN	30
7844	TURNER	30
7900	JAMES	30
7521	WARD	30
7902	FORD	20
7369	SMITH	20

...
14 rows selected.

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
30	SALES	CHICAGO
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
30	SALES	CHICAGO
30	SALES	CHICAGO
30	SALES	CHICAGO
30	SALES	CHICAGO
20	RESEARCH	DALLAS
20	RESEARCH	DALLAS

...
14 rows selected.

Condizioni di Ricerca
Uso dell'operatore AND

```
SQL> SELECT   emp.empno,   emp.ename, emp.deptno,
  2           dept.deptno, dept.loc
  3  FROM     emp, dept
  4  WHERE    emp.deptno=dept.deptno AND ENAME='KING'
```

Join di piu' di due Tabelle

CLIENTI

NAME	CUSTID
JOCKSPORTS	100
TKB SPORT SHOP	101
VOLLYRITE	102
JUST TENNIS	103
K+T SPORTS	105
SHAPE UP	106
WOMENS SPORTS	107
...	...

9 rows selected.

ORDINI

CUSTID	ORDID
101	610
102	611
104	612
106	601
102	602
106	
106	
...	

21 rows s

PROD.

ORDID	ITEMID
610	3
611	1
612	1
601	1
602	1
...	

64 rows selected.

Join di piu' di due Tabelle

```
SQL> SELECT   *
  2  FROM     clienti, ordini, prod
  3  WHERE    clineti.custid=ordini.custid
              AND prod.ordid=prod.ordid;
```

Visualizzare Dati da piu' tabelle

Per visualizzare dati da più tabelle uno strumento è il join

Ma ne esistono anche altri, ad esempio gli operatori Insiemistici…

Giunzioni ed Operatori Insiemistici

Sintassi SQL per i tipi di join (giunzione) visti sonora e per gli operatori insiemistici

- *Giunzione* ::= [CROSS|UNION|NATURAL] [LEFT| RIGHT | FULL] JOIN
- *OpInsiem* ::= (UNION | INTERSECT | EXCEPT) [CORRESPONDING [BY "(" Attributo {,Attributo}")"]]
- USING e ON solo con JOIN; LEFT, RIGHT,FULL solo con NATURAL JOIN e JOIN
- Cross Join e' il prodotto cartesiano
- Union Join e' **l'unione esterna** (outer join) cioe' si estendono le due tabelle con le colonne dell'altra con valori nulli e si fa l'unione delle due stesse tabelle.
- Natural Join e' quella classica
- Join... Using e' la natural join sui dati attributi
- Join…On su quelli che soddisfano una data condizione

Altro Esempio di join

Agenti(CodiceAgente,Nome,Zona Supervisore,Commissione)

Clienti(CodiceCliente,Nome,Citta',Sconto)

Ordini(CodiceOrdine,CodiceCliente,CodiceAgente,Articolo,Ammontare)

SELECT Agenti.CodiceAgente,Ordini.Ammontare

FROM Agenti JOIN Ordini

ON Agenti.Supervisore = Ordini.CodiceAgente

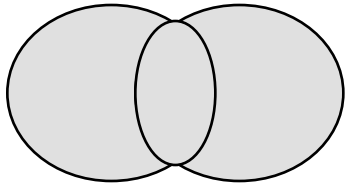
Codice agente ed ammontare degli ordini dei supervisori

SELECT Agenti.CodiceAgente,Ordini.Ammontare

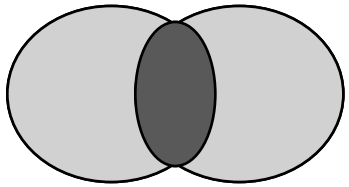
FROM Agenti NATURAL LEFT JOIN Ordini

Codice agente ed ammontare degli agenti incluso quelli che non hanno effettuato ordini (avranno ammontare NULL)

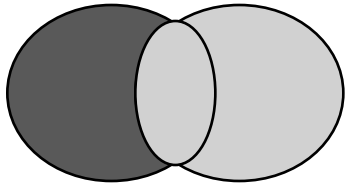
Le operazioni Insiemistiche



A UNION B



A INTERSECT B



A MINUS B

Le operazioni Insiemistiche

- *OpInsiem* ::= (UNION | INTERSECT | EXCEPT) [CORRESPONDING [BY "(" Attributo {,Attributo}")"]]
- Union,Intersect,Except sono $\cup$,$\cap$,-. CORRESPONDING fa proiettare sugli attributi comuni e poi si applica l'operatore insiemistico. Se c'e' anche BY si specificano su quali comuni attributi proiettare

UNIONE

SELECT *
FROM Clienti UNION CORRESPONDING Agenti
Fornisce tutti i nomi dei clienti e degli agenti.

In effetti nei sistemi commerciali sarebbe
SELECT Nome FROM Clienti
UNION
SELECT Nome FROM Agenti

Operatori Aggregati

Gli operatori aggregati valutano proprietà che dipendono da un insieme di tuple, ad esempio sapere il numero di impiegati del dipartimento di produzione, ossia un semplice conteggio del numero di righe.
L'SQL prevede cinque operatori: **count, sum, avg, max, min**

Operatori Aggregati

count (*)

effettua il conteggio righe, ad esempio

select count(*) from Impiegati

count [distinct | all] ***listaattributi***

determina il numero di valori diversi (**distinct**) o semplicemente non nulli anche se ripetuti (**all**) sugli attributi ***listaattributi***, ad esempio **select count (distinct Nome) from Impiegato.** Conta quanti sono i nomi distinti e non nulli su tutti gli impiegati, quindi se abbiamo tre Mario e due Carlo e tre null, la query torna solo 2 ("Mario" e "Carlo"); una select count(*) invece tornerebbe 8

sum | avg | max | min ([distinct | all] ***attrexpr*** **)**

i quattro operatori hanno sintassi uguale. **Sum** effettua la somma di tutti i valori che può avere l'espressione costruita sugli attributi, per esempio **select sum (StipendioAnnuale/12) from Stipendi** effettua la somma di tutti gli stipendi mensili. **avg** calcola la media, **max** il massimo e **min** il minimo.

Tutte le funzioni escludono i valori NULL

Cosa sono?

- Operano su insiemi di righe per dare un risultato per gruppo.

IMPIEGATI

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

"Salario Massimo"

MAX(SAL)
5000

Esempi

- SELECT MIN(Voto),MAX(Voto),AVG(Voto)
- FROM Esami
- WHERE Matricola = '123456'

- SELECT COUNT(*)
- FROM Esami
- WHERE Corso = 'Database 1'

Uso operatori aggregati

```
SELECT        [column,] group_function(column)
FROM          table
[WHERE        condition]
[GROUP BY     column]
[ORDER BY     column];
```

Uso di AVG e SUM

- Possono essere usati su dati numerici.

```
SQL> SELECT   AVG(sal), MAX(sal),
  2           MIN(sal), SUM(sal)
  3  FROM     emp
  4  WHERE    job LIKE 'SALES%';
```

AVG(SAL)	MAX(SAL)	MIN(SAL)	SUM(SAL)
1400	1600	1250	5600

Uso di MIN e MAX

- Possono essere usati su qualsiasi tipo.

```
SQL> SELECT   MIN(hiredate), MAX(hiredate)
  2  FROM     emp;
```

```
MIN(HIRED MAX(HIRED
--------- ---------
17-DEC-80 12-JAN-83
```

Uso di COUNT

- COUNT(*) ritorna il numero di righe di una tabella.

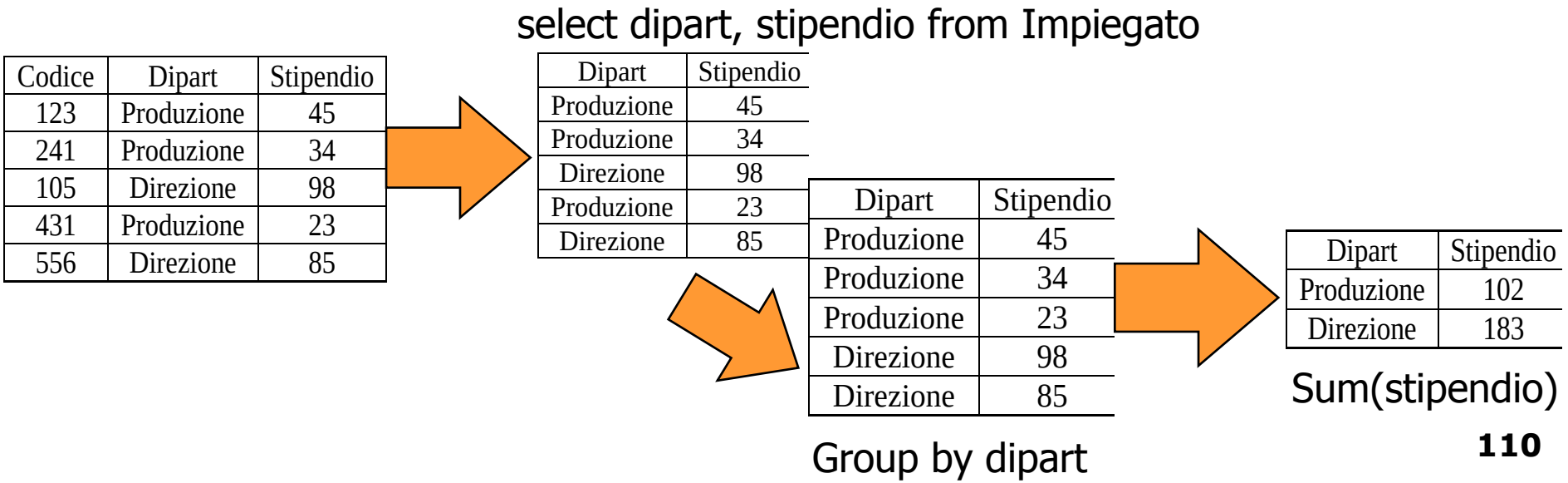
```
SQL> SELECT   COUNT(*)
  2  FROM     emp
  3  WHERE    deptno = 30;
```

```
 COUNT(*)
---------
         6
```

Raggruppamento

L'operatore aggregato **group by** permette di raggruppare le righe in sottoinsiemi, specificando come criterio di raggruppamento un insieme di attributi dopo la clausula. Ogni sottoinsieme conterrà le righe aventi tutte lo stesso valore per quello o quegli attributi, analogamente per tutti gli altri sottoinsiemi. Ad esempio:

select dipart, sum (stipendio) from Impiegato group by dipart



Codice	Dipart	Stipendio
123	Produzione	45
241	Produzione	34
105	Direzione	98
431	Produzione	23
556	Direzione	85

Dipart	Stipendio
Produzione	45
Produzione	34
Direzione	98
Produzione	23
Direzione	85

Dipart	Stipendio
Produzione	45
Produzione	34
Produzione	23
Direzione	98
Direzione	85

Dipart	Stipendio
Produzione	102
Direzione	183

Raggruppamento

►Il group by può anche includere una eventuale condizione che i sottoinsiemi devono verificare, azione che si concretizza con la clausula having, ad esempio select dipart, sum (stipendio) from Impiegato group by dipart having sum(stipendio)>120 produrrebbe, applicato allo stesso esempio, la sola seconda riga, avente il valore 183>120.
►Having si può usare anche senza group by, essendo in questo caso il sottoinsieme pari a tutte le righe.
►Having ammette come argomento una espressione che normalmente contiene operatori aggregati. Potrebbero in teoria esserci direttamente anche gli attributi, ma espressioni su attributi sono generalmente messi in where. Questa è una regola normalmente applicata.

Creare gruppi di dati

IMPIEGATI

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

2916.6667

2175

1566.6667

"salario medio in IMPIEGATI per ogni dipartimento"

DEPTNO	AVG(SAL)
10	2916.6667
20	2175
30	1566.6667

Creare gruppi con GROUP BY

```
SELECT        column, group_function(column)
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[ORDER BY     column];
```

- Divide le righe di una tabella in gruppi piu' piccoli.

Raggruppamento

- GROUP BY Attributo {, Attributo} [HAVING Condizione]
- Va posto dopo WHERE e opera una partizione delle righe del risultato in base ad eguali valori su quegli attributi (NULL incluso). Quindi si produce una n-upla per ogni classe di equivalenza che soddisfa la condizione HAVING

Uso di GROUP BY

- Tutte le colonne della SELECT che non sono in funzioni di gruppo devono essere nella GROUP BY.

```
SQL> SELECT   deptno, AVG(sal)
  2  FROM     emp
  3  GROUP BY deptno;
```

```
    DEPTNO   AVG(SAL)
---------- ----------
        10 2916.6667
        20       2175
        30 1566.6667
```

Uso GROUP BY

- La colonna di GROUP BY non deve essere necessariamente nella SELECT.

```
SQL> SELECT   AVG(sal)
  2  FROM     emp
  3  GROUP BY deptno;
```

```
 AVG(SAL)
---------
2916.6667
      2175
1566.6667
```

Raggruppare piu' di una colonna

IMPIEGATI

DEPTNO	JOB	SAL
10	MANAGER	2450
10	PRESIDENT	5000
10	CLERK	1300
20	CLERK	800
20	CLERK	1100
20	ANALYST	3000
20	ANALYST	3000
20	MANAGER	2975
30	SALESMAN	1600
30	MANAGER	2850
30	SALESMAN	1250
30	CLERK	950
30	SALESMAN	1500
30	SALESMAN	1250

"sommare I salari in IMPIEGATI per ogni lavoro, Raggruppati per dipartimento"

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

Uso di GROUP BY su colonne multiple

```
SQL> SELECT   deptno, job, sum(sal)
  2  FROM     emp
  3  GROUP BY deptno, job;
```

```
    DEPTNO JOB          SUM(SAL)
---------- --------- ----------
        10 CLERK           1300
        10 MANAGER         2450
        10 PRESIDENT       5000
        20 ANALYST         6000
        20 CLERK           1900
...
9 rows selected.
```

Query illegali con raggruppamento

- Ogni colonna o espressione della SELECT che non e' argomento di funzioni di raggruppamento deve essere nella GROUP BY.

```
SQL> SELECT   deptno, COUNT(ename)
  2  FROM     emp;
```

```
SELECT deptno, COUNT(ename)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Query illegali raggrupamento

- Non puo' essere usata la WHERE per restringere I gruppi.
- Deve essere usata la HAVING.

```
SQL> SELECT    deptno, AVG(sal)
  2  FROM      emp
  3  WHERE     AVG(sal) > 2000
  4  GROUP BY  deptno;
```

```
WHERE AVG(sal) > 2000
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

Escludere gruppi

IMPIEGATI

DEPTNO	SAL
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

5000

3000

2850

"salario massimo per dipartmento maggiore di $2900"

DEPTNO	MAX(SAL)
10	5000
20	3000

Clausola HAVING

- Uso di HAVING per restringere gruppi
  - Le righe sono raggruppate.
  - La funzione di raggruppamento e' applicata.

```
SELECT        column, group_function
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[HAVING       group_condition]
[ORDER BY     column];
```

Uso di HAVING

```
SQL> SELECT   deptno, max(sal)
  2  FROM     emp
  3  GROUP BY deptno
  4  HAVING   max(sal)>2900;
```

```
    DEPTNO  MAX(SAL)
---------- ---------
        10      5000
        20      3000
```

Uso di HAVING

```
SQL> SELECT     job, SUM(sal)  PAYROLL
  2  FROM       emp
  3  WHERE       job NOT LIKE 'SALES%'
  4  GROUP BY   job
  5  HAVING     SUM(sal)>5000
  6  ORDER BY   SUM(sal);
```

```
JOB           PAYROLL
--------- ---------
ANALYST          6000
MANAGER          8275
```

Funzioni di raggruppamento annidate

```
SQL> SELECT   max(avg(sal))
  2  FROM     emp
  3  GROUP BY deptno;
```

```
MAX(AVG(SAL))
-------------
    2916.6667
```

Esempio

```
SELECT Nome, Matricola MIN(Voto),MAX(Voto),AVG(Voto)
FROM Esami, Studenti
WHERE Esami.Matricola = Studenti.Matricola
GROUP BY Nome,Matricola
HAVING COUNT(*) > 8
```

Sommario

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column];
```

- Ordine di valutazione delle clausole:
  - WHERE
  - GROUP BY
  - HAVING

Interrogazioni nidificate

Si tratta di interrogazioni in cui la clausula where è un'espressione che contiene il risultato di un'altra select. Generalmente, l'espressione contiene a primo membro un attributo (ossia è del tipo **where** *nomecolonna=valore*), se il *valore* è un'altra select, il problema è la non compatibilità fra i due membri che impedisce il confronto (attributo vs tabella).
Per tale motivo, si usano i due operatori **any** e **all**, il primo che richiede l'uguaglianza fra l'attributo ed almeno una riga della tabella proveniente dall'altra select, mentre **all** impone che tutte le righe soddisfino questa condizione.

Interrogazioni nidificate

Esempio:
select * from impiegato
where dipart = any (select dip from Dipartimenti where citta="FI")
La query seleziona le righe di impiegato per cui il dipartimento coincide con almeno un valore di dipartimento di Firenze posto entro il campo dip dentro la tabella dipartimenti.
Interrogazioni come queste si possono anche fare con i join, generalmente le interrogazioni nidificate si scelgono quando si deve migliorare il grado di leggibilità, ad esempio nei casi più complicati.

SQL fornisce anche **in** e **not in** per stabilire se un elemento appartiene o meno ad un insieme. Di fatto sono identici a **"= any"** e **"<> all"**.

Interrogazioni nidificate

L'interpretazione intuitiva delle query annidate è che prima venga effettuata la query più interna e successivamente quelle sempre più esterne.
Può capitare tuttavia che la query interna si riferisca alla query più esterna che la contiene; solitamente questo riferimento è rappresentato da una variabile, ad esempio un alias, definito nella query esterna, ed usato in quella più interna, caso in cui la interpretazione intuitiva non va più bene, ed occorre affidarsi alla definizione di query select, ossia si deve prima costruire il prodotto cartesiano delle tabelle, seguito dalla selezione (where) ed infine dalla proiezione (target list).
Poichè in questo caso la selezione contiene un'altra select, si deve anche qui procedere secondo la definizione. In definitiva, si deve, prima valutare la query esterna, e poi, su ogni riga così ottenuta, valutare quella interna, vedendo se l'espressione in cui si trova è vera o meno, e procedendo così per tutte le altre righe della query esterna.

Interrogazioni nidificate

L'operatore **exist** consente di usare una query internamente ad un'altra senza bisogno di usare **any** o **all**, ossia senza dovere costruire un'espressione di confronto fra un attributo e la query interna. L'operatore restituisce vero se la query posta di seguito restituisce almeno un elemento, mentre restituisce falso in caso contrario. Questo operatore può essere usato in maniera significativa quando esiste un riferimento fra query contenuta e contenente.

Interrogazioni nidificate

```
SELECT e.Matricola
FROM Esami e
WHERE EXISTS
(SELECT *  FROM  Corsi c
WHERE c.Corso = e.Corso AND    c.Professore='Ferro')
```

EQUIVALE A

```
SELECT e.Matricola
FROM Esami e, Corsi c
WHERE c.Corso = e.Corso AND c.Professore='Ferro'
```

► E' piu' efficiente.

Interrogazioni nidificate

Come esempio sia di query annidata che di exist, si ha:

select * from Persona as P where exist
(select * from Persona as P1 where
P1.Nome=P.Nome and
P1.Cognome=P.Cognome and
P1. CF<>P.CF)

La query estrae tutti gli omonimi (stesso nome e cognome con diverso codice fiscale), usando due alias, che creano un legame fra query interna ed esterna. Infatti in quella interna si fa riferimento a P, nome che ha un senso solo se prima di eseguire la query interna è stata fatta quella esterna **select * from Persona as P**. Come detto in precedenza, infatti, in caso di interazione fra query contenente e contenuta, si esegue prima la contenente, ottenendo in questo caso tutte le persone in Persona, e poi si considera ognuna di queste persone (tupla) e si applica la select interna, in cui P.* sono campi "bloccati", variando invece solo P1.*.

Altro Esempio interrogazione nidificata

- Agenti(CodiceAgente,Nome,Zona Supervisore,Commissione)
- Clienti(CodiceCliente,Nome,Citta',Sconto)
- Ordini(CodiceOrdine,CodiceCliente,CodiceAgente, Articolo,Data,Ammontare)

Esempio

Supponiamo di voler trovare i codici di quei clienti che hanno fatto ordini a TUTTI gli agenti di Catania.

Per ogni agente z di Catania esiste un ordine y del nostro cliente x a z.

```
SELECT c.CodiceCliente
FROM     Clienti c
WHERE NOT EXISTS
                (SELECT *  FROM    Agenti a
                  WHERE a.Zona = 'Catania'
                  AND NOT EXISTS
                            ( SELECT *
                               FROM Ordini v
                               WHERE v.CodiceCliente = c.CodiceCliente
                               AND v.CodiceAgente = a.CodiceAgente) )
```

SQL – Viste

Viste

Per definire una vista in SQL, si usa la sintassi:

create view *nomevista* [**(** *listaattributi* **)**] **as** *selectSQL*
[**with** [**local | cascaded**] **check option**]

l'interrogazione SQL che consente di prelevare i dati che popolano la vista deve avere lo stesso numero e tipo di attributi della *listaattributi*, per associarli ordinatamente e correttamente. Le viste possono anche essere definite una in funzione dell'altra, esempio:

```
create view imp_ammin (Mat, N, C, Stip) as
        select M, No, Co, S from impiegato
        where Dip='Amm' and S>10
create view imp_ammin_poveri as
        select * from imp_ammin
        where Stip<50 with check option
```

SQL – Viste

Sulle viste è permesso effettuare un aggiornamento che si può o no ripercuotere all'indietro, sulle tabelle da cui la vista dipende.
Questa azione diventa complessa se le tabelle di partenza vengono in qualche modo elaborate per generare la vista, sicchè lo standard SQL permette aggiornamenti solo quando ogni riga della vista proviene da una sola riga di ciascuna tabella (così è possibile sapere come propagare all'indietro).
Certi sistemi commerciali consentono gli aggiornamenti soltanto se la tabella di partenza è una sola.

SQL – Viste

Check option specifica che gli aggiornamenti possono essere fatti solo sulle righe della vista, e che dopo gli agg. le righe ottenute devono soddisfare ancora il vincolo presente nella definizione della stessa (appunto, *check option*).
Se inoltre una vista è definita in termini di altre, **local** specifica che il controllo sul fatto che le righe vengono rimosse dalla vista deve essere effettuato solo nella vista in esame, mentre **cascaded** propaga tale controllo a tutti i livelli (quest'ultima è l'opzione di default).
Nell'esempio precedente, assegnare a Stip nella seconda vista ("figlia" della prima) il valore di 8 milioni non va, perché 8<50, ma il default per chek option è cascaded, per cui si controlla anche che sia >10 (vista "padre"), e 8<10, quindi non va. Se invece si scrivesse check option local, 8 è ammesso.

SQL – Autorizzazioni

Controllo dell'accesso

SQL consente di gestire privilegi per utenti/gruppi sugli tabelle e viste

Per ***assegnare privilegi***, si usa:
grant [*privilegi* / **all privileges**] **on** *risorsa* **to** *utenti* [**with grant option**]
I *privilegi* sono **insert, delete** (validi per tabelle e viste), **update, select** (validi anche per gli attributi), **references** (per tabelle e attributi, che consente, a chi ottiene il privilegio, di potere fare riferimento ad una tabella non propria), **usage**, che consente di usare un certo dominio (non proprio) nell'ambito della definizione dello schema di una propria tabella.
Drop e **alter** sono comandi che restano di competenza di chi ha creato l'oggetto.
All privileges consente di assegnare tutti i privilegi.
With grant option consente a chi ottiene il privilegio di poterlo propagare a qualcun altro.

SQL – Autorizzazioni

Per ***revocare privilegi***, si usa:
revoke *privilegio* **on** *risorsa* **from** *utenti* [**restrict | cascade**]
l'opzione **restrict** impedisce la revoca se questa provoca reazioni a catena, ad esempio cancellare una tabella di un utente a cui prima era stato concesso di crearla.
Cascade forza la revoca, il che potrebbe avere anche conseguenze a catena indesiderate.

SQL – Uso nei linguaggi

Uso di SQL nei linguaggi di programmazione

Di fatto, quasi mai si ricorre alla complicatezza di SQL, ma si fa uso di procedure precostituite, sia per la semplicità, che perché spesso le azioni devono essere fatte periodicamente o in maniera non interattiva (batch).

Per integrare SQL con i linguaggi di programmazione, all'interno del programma scritto con un linguaggio ad alto livello si pongono le chiamate alla procedura SQL. Il compilatore del linguaggio intercetta la chiamata, e la sostituisce con le primitive di livello più basso che realizzano la funzione richiesta.

SQL – Uso nei linguaggi

Un ***problema di integrazione*** è dovuto al fatto che i linguaggi accedono ad una riga alla volta (approccio *tuple-oriented*), non intervenendo quindi su un'intera tabella nel suo insieme (approccio *set-oriented*, proprio dell'SQL). Questa diversa filosofia rende necessatio uniformare in qualche modo il comportamento.
Poiché talora un approccio di tipo *tuple-oriented* risulta più utile, si è dotato l'SQL di questa capacità (uniformandolo quindi ai linguaggi di programmazione), introducendo il concetto di **cursore**.
Un cursore è uno strumento che consente di accedere ad una riga per volta, e si associa ad una singola select. Un cursore può essere aperto, posizionato su una specifica riga, spostato in avanti o indietro, e consente di prelevare i dati della singola riga e memorizzarli in variabili (non SQL, ma del linguaggio di programmazione in cui i comandi SQL sono posti).