Linguaggio SQL

Prof. Francesco Accarino

IIS Altiero Spinelli Sesto San Giovanni

SQL: caratteristiche generali

- SQL (Structured Query Language) è il linguaggio standard de facto per DBMS relazionali, che riunisce in sé funzionalità di:
  - DDL = Data Definition Language;
  - DML = Data Manipulation Language;
  - DCL = Data Control Language.

- SQL è nato come un linguaggio **dichiarativo** (non-procedurale), ovvero non specifica la sequenza di operazioni da compiere per ottenere il risultato.

- SQL è "**relazionalmente completo**", nel senso che ogni espressione dell'algebra relazionale può essere tradotta in SQL .

- Il modello dei dati di SQL è basato su **tabelle anziché relazioni**:
  - possono essere presenti righe (tuple) duplicate;
  - in alcuni casi l'ordine delle colonne (attributi) ha rilevanza;
  - …il motivo è pragmatico (ossia legato a considerazioni sull'efficienza).

Data Definition Language (DDL)

- Il DDL di SQL permette di definire **schemi di relazioni** (o “table”, tabelle), modificarli ed eliminarli.
- Permette inoltre di specificare **vincoli**, sia a livello di tupla (o “riga”) che a livello di tabella.
- Permette di definire nuovi **domini**, oltre a quelli predefiniti
  - Per vincoli e domini si può anche fare uso del DML (quindi inizialmente non è obbligatorio definirli completamente).
- Inoltre si possono definire **viste** (“view”), ovvero tabelle virtuali, e **indici**, per accedere efficientemente ai dati.

Tipi di dato (SQL Server) 1

- bigint (8 bytes) Contiene valori numerici interi da -4294967296 a 4294967294.
- binary(n) (lunghezza fissa) Contiene dati binari (1 byte) fino ad un massimo di 8000 dati.
- bit (1 bit) Rappresenta i flag (vero/falso o true/false o si/no). Non possono avere valori nulli e non possono avere indici.
- char(n) (lunghezza fissa) Contiene caratteri ANSI (1 byte) fino ad un massimo di 8000 caratteri.
- datetime (8 bytes) Contiene date tra il 1/gen/1753 e il 31/dic/9999 (precisione al trecentesimo di secondo).
- decimal(p, s) (da 2 bytes a 17 bytes) Contiene valori tra 10^38 - 1 e - 10^38 -1. Con p cifre di precisione (massimo 28), e s cifre decimali dopo la virgola (scala).
- float (8 bytes) Contiene numeri reali positivi da 2.23E-308 a 1.79E308 e negativi da -2.23E-308 a -1.79E308 (massimo 15 cifre di precisione).

Tipi di dato (SQL Server) 2

- image Contiene fino a 2147483647 bytes di dati binari (è solitamente usato per le immagini).
- int (4 bytes) Contiene valori numerici interi da -2147483648 a 2147483647.
- money (8 bytes) Contiene valori monetari da -922337203685477.5808 a 922337203685477.5807
- nchar(n) (lunghezza fissa) Contiene caratteri UNICODE (2 bytes) fino ad un massimo di 4000 caratteri.
- ntext (lunghezza variabile) Contiene caratteri UNICODE fino ad un massimo di 1073741823 caratteri.
- numeric(p, s) E' equivalente al tipo 'decimal(p, s)'
- nvarchar(n) (lunghezza variabile) Contiene caratteri UNICODE (2 bytes) fino ad un massimo di 4000 caratteri.

Tipi di dato (SQL Server) 3

- real (4 bytes) Contiene numeri reali positivi da 1.18E-38 a 3.40E38 e negativi da -1.18E-38 a -3.40E38 (massimo 7 cifre di precisione).
- smalldatetime (4 bytes) Contiene date tra il 1/gen/1753 e il 31/dic/9999 (precisione al minuto).
- smallint (2 bytes) Contiene valori numerici interi da -32768 a 32767.
- smallmoney (4 bytes) Contiene valori monetari da -214748.3648 a 214748.3647
- sql_variant Tipo che può contenere tipi di dati diversi (int, binary, char).

Tipi di dato (SQL Server) 4

- text (lunghezza variabile) Contiene caratteri ANSI (1 byte) fino ad un massimo di 2147483647 caratteri.
- timestamp (8 bytes) È un contatore incrementale per colonna assegnato automaticamente da SQL Server 7.
- tinyint (1 byte) Contiene valori numerici interi da 0 a 255.
- uniqueidentifier (16 bytes) E' un identificatore unico a livello globale E' generato automaticamente da SQL Server.
- varbinary(n) (lunghezza variabile) Contiene dati binari (1 byte) fino ad un massimo di 8000 dati.
- varchar(n) (lunghezza variabile) Contiene caratteri ANSI (1 byte) fino ad un massimo di 8000 caratteri.
- xml è equivalente al tipo ‘ntext’.

Operatori (SQL Server)

+	Addizione	<>	Disuguaglianza
-	Sottrazione	AND	E logico
*	Prodotto	OR	O logico
/	Divisione	NOT	Negazione
%	Modulo		
<	Minore		
>	Maggiore		
<=	Minore o Uguale		
>=	Maggiore o Uguale		
=	Uguaglianza		

Creazione ed eliminazione di tabelle

- Mediante l‟istruzione **CREATE TABLE** si definisce lo schema di una tabella e se ne crea un‟istanza vuota:
  - per ogni attributo va specificato il dominio, un eventuale valore di default e eventuali vincoli;
  - infine possono essere espressi altri vincoli a livello di tabella.

```
CREATE TABLE Imp (
CodImp        char(4)        PRIMARY KEY,  -- chiave primaria
CF            char(16)       NOT NULL UNIQUE, -- chiave
Cognome       varchar(60)    NOT NULL,
Nome          varchar(30)    NOT NULL,
Sede          char(3)        REFERENCES Sedi(Sede), -- FK
Ruolo         char(20)       DEFAULT „Programmatore‟,
Stipendio     int            CHECK (Stipendio > 0),
UNIQUE (Cognome, Nome) -- chiave )
```

- Mediante l‟istruzione **DROP TABLE** è possibile eliminare lo schema di una tabella (e conseguentemente la corrispondente istanza):

```
DROP TABLE Imp
```

Vincoli (1)

- Valori di default e valori NULL:
  - Per vietare la presenza di valori nulli, è sufficiente imporre il vincolo **NOT NULL:**

    Cognome varchar(60) NOT NULL
  - Per ogni attributo è inoltre possibile specificare un valore di default:

    Ruolo char(20) DEFAULT ”Programmatore”
- Chiavi:
  - La definizione di una chiave avviene esprimendo un vincolo **UNIQUE, che si può** specificare in linea, se la chiave consiste di un singolo attributo:

    CF char(16) UNIQUE
  - dopo aver dichiarato tutti gli attributi, se la chiave consiste di uno o più attributi:

    UNIQUE(Cognome,Nome)
  - PRIMARY KEY definisce la chiave primaria:

    CodImp char(4) PRIMARY KEY
- la specifica di una chiave primaria non è obbligatoria;
- si può specificare al massimo una chiave primaria per tabella;
- non è necessario specificare NOT NULL per gli attributi della primary key.

Vincoli (2)

- Chiavi straniere ("foreign key")
  - La definizione di una foreign key avviene specificando un vincolo **FOREIGN KEY**, **e** indicando quale chiave viene referenziata;
  - le colonne di destinazione devono essere una chiave della tabella destinazione (non necessariamente la chiave primaria):

FOREIGN KEY (Sede) REFERENCES Sedi(Sede)

- Vincoli generici ("check constraint")
  - Mediante la clausola **CHECK è possibile esprimere vincoli di tupla arbitrari**,sfruttando tutto il potere espressivo di SQL. La sintassi è:

CHECK (<condizione>)

  - Il vincolo è violato se esiste almeno una tupla che rende falsa la <condizione>(esclusi i valori NULL):

Stipendio int CHECK (Stipendio > 0)

  - Se CHECK viene espresso a livello di tabella (anziché nella definizione Dell'attributo) è possibile fare riferimento a più attributi della tabella stessa:

CHECK (ImportoLordo = Netto + Ritenute)

Modifica degli schemi

- Il linguaggio implementa anche delle istruzioni dedicate alla modifica delle strutture esistenti utilizzando il comando *alter*, per aggiungere una nuova colonna add a quelle già esistenti, oppure per togliere una colonna *drop*.

```
ALTER TABLE persone
ADD nascita date;

ALTER TABLE persona
Drop Indirizzo;
```

- Drop può essere utilzzato anche per eliminare una tabella oppure l'intero database
  - Drop persona;
  - Drop agenda

Modifica di tabelle

- Mediante l'"istruzione ALTER TABLE è possibile modificare lo schema di una tabella, in particolare:
  - aggiungendo o rimuovendo attributi;
  - aggiungendo o rimuovendo vincoli.

```
ALTER TABLE Imp
ADD COLUMN Sesso char(1) CHECK (Sesso in (″M″,″F″))
ADD CONSTRAINT StipendioMax CHECK (Stipendio < 4000)
DROP CONSTRAINT StipendioPositivo
DROP UNIQUE(Cognome,Nome);
```

- Se si aggiunge un attributo con vincolo NOT NULL, bisogna prevedere un valore di default, che il sistema assegnerà automaticamente a tutte le tuple già presenti:

```
ADD COLUMN Istruzione char(10) NOT NULL DEFAULT
″Laurea″
```

DML (Data Manipulation Language

- operazioni di
  - inserimento: **`insert`**
  - eliminazione: **`delete`**
  - modifica: **`update`**
- di una o più ennuple di una relazione
- sulla base di una condizione che può coinvolgere anche altre relazioni

DML (Data Manipulation Language

- Inserimento dati

INSERT INTO <NomeTabella>
[(<Campo1>, <Campo2>, … <CampoN>)]
VALUES
(<Valore1>, <Valore2>, … <ValoreN>);

<NomeTabella> -Nome della tabella in cui inserire i dati.

<Campo> - ista dei campi della tabella in cui inserire i valori specificati di seguito.

<Valore> - Lista dei valori da inserire nei rispettivi campi.

L'elenco dei campi è opzionale; se non viene specificato è necessario inserire un valore per tutti i campi della tabella.

Inserimento: esempio

```
insert into persone values('Mario',25,52)

insert into persone(nome, eta, reddito)
values('Pino',25,52)

insert into persone(nome, reddito)
values('Lino',55)

insert into persone (nome)
select padre
from paternita
where padre not in (select nome from persone)
```

Inserimento: commenti

- l'ordinamento degli attributi (se presente) e dei valori è significativo
- le due liste di attributi e di valori debbono avere lo stesso numero di elementi
- se la lista di attributi è omessa, si fa riferimento a tutti gli attributi della relazione, secondo l'ordine con cui sono stati definiti
- se la lista di attributi non contiene tutti gli attributi della relazione, per gli altri viene inserito un valore nullo (che deve essere permesso) o un valore di default

Eliminazione di ennuple

DELETE FROM <NomeTabella>
[WHERE <Condizione>];

- <NomeTabella> - Nome della tabella dalla quale verranno eliminati i dati.
- <Condizione> - Condizione che deve essere soddisfatta dai campi dei record che verranno eliminati.
- Se non viene specificata alcuna condizione WHERE, verranno eliminati tutti i record.

Eliminazione di ennuple

- E' da notare la differenza tra il comando delete ed il comando drop. Ad esempio il comando:

Delete from Dipartimento

- …elimina tutte le righe della tabella dipartimento, ma lo schema rimane immutato; il comando, infatti, cancellerà solo le istanze della tabella. Mentre il comando:

Drop table Dipartimento

- elimina tutte le istanze della tabella, nonché lo schema.

Modifica di ennuple

- **Sintassi**:

`update` ***NomeTabella***

`set` ***Attributo*** **`=`** **<** ***Espressione*** **|** ***`select`*** ***...*** **|** ***`null`*** **|** ***`default`*** **>**

[**`where`** ***Condizione*** ***]***

- **Semantica**: **vengono modificate le ennuple della tabella che soddisfano** la condizione "where"
- *Esempi:*

```
update persone set reddito = 45
where nome = 'Piero'
update persone set reddito = reddito * 1.1
where eta < 30
```

I domini

- In SQL sono utilizzabili 2 tipi di domini
  - Domini elementari (predefiniti):
    - carattere: singoli caratteri o stringhe, anche di lunghezza variabile;
    - bit: singoli booleani o stringhe;
    - numerici, esatti e approssimati;
    - data, ora, intervalli di tempo.
  - Domini definiti dall'utente (semplici): utilizzabili in definizioni di relazioni, anche con vincoli e valori di default. Si definiscono tramite l'istruzione:

```
CREATE DOMAIN Voto AS SMALLINT
DEFAULT NULL
CHECK ( value >=18 AND value <= 30 )
```

Politiche di Integrità referenziale

- Anziché lasciare al programmatore il compito di garantire che afronte di cancellazioni e modifiche i vincoli di integrità referenzialesiano rispettati, si possono specificare opportune **politiche di reazione** **in fase di definizione degli schemi.**

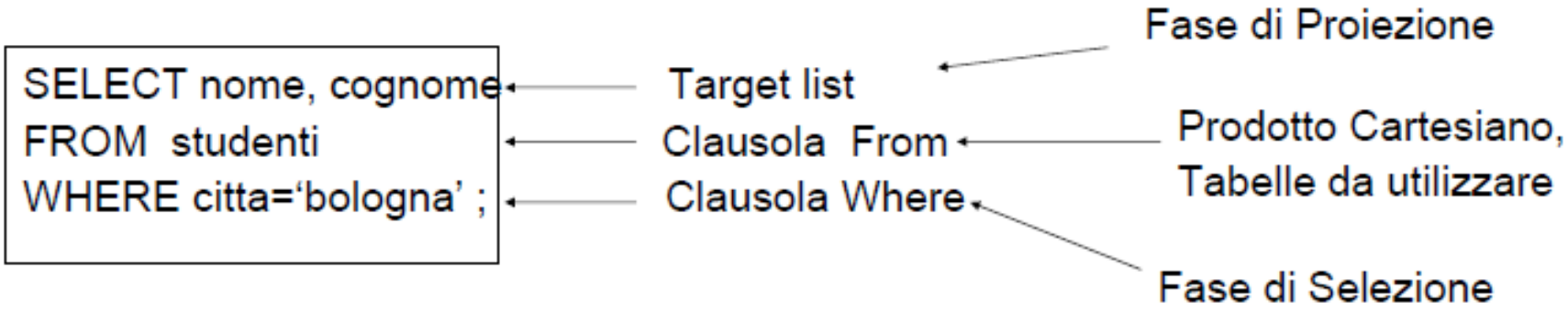
```
CREATE TABLE Imp (
CodImp char(4) PRIMARY KEY,
Sede char(3),
...
FOREIGN KEY Sede REFERENCES Sedi
ON DELETE CASCADE          -- cancellazione in cascata
ON UPDATE NO ACTION        -- modifiche non permesse
```

Altre politiche: SET NULL e SET DEFAULT.

QL (Query Language)

- Per estrarre informazioni dalla base di dati si utilizza l'istruzione SELECT.
- La sintassi completa dell'istruzione SELECT è complessa perché l'istruzione implementa varie funzionalità.

1. Per utilizzare piu' tabelle congiuntamente (join) si esegue il **prodotto cartesiano** delle tabelle coinvolte (se c'e' una sola tabella, il prodotto cartesiano non viene effettuato)
2. Si **selezionano le righe (tuple) sulla base del predicato della clausola Where**
3. Si **proietta sugli attributi della target list**



L'istruzione SELECT

- È l'istruzione che permette di eseguire interrogazioni (query) sul DB.

SELECT [ALL|DISTINCT][TOP(n)[PERCENT][WITH TIES]] A1,A2,..,Am
FROM R1,R2,..,Rn
[WHERE <condizione>]
[GROUP BY <listaAttributi>]
[HAVING <condizione>]
[ORDER BY <listaAttributi>]

– ovvero:

» **SELECT (o TARGET) list (che cosa si vuole come risultato)**
» **clausola FROM (da dove si prende)**
» **clausola WHERE (quali condizioni deve soddisfare)**
» **clausola GROUP BY (le colonne su cui raggruppare)**
» **clausola HAVING (condizioni relative ai gruppi)**
» **clausola ORDER BY (ordinamento)**

Il comando SELECT permette di realizzare le operazioni di selezione, proiezione, join, raggruppamento e ordinamento.

DB di riferimento per gli esempi

Imp

CodImp	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

Sedi

Sede	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Prog

CodProg	Citta
P01	Milano
P01	Bologna
P02	Bologna

SELECT su singola tabella

Codice, nome e ruolo dei dipendenti della sede S01

```
SELECT  CodImp, Nome, Ruolo
FROM    Imp
WHERE   Sede = ”S01”
```

CodImp	Nome	Ruolo
E001	Rossi	Analista
E003	Bianchi	Programmatore
E006	Grigi	Sistemista
E007	Violetti	Programmatore

Si ottiene in questo modo:

- la clausola FROM impone di accedere alla sola tabella IMP;
- la clausola WHERE impone di selezionare solo le tuple per cui Sede=“S01“;
- infine, si estraggono i valori degli attributi (o “colonne”) nella SELECT list.

- Equivale a $\pi_{CodImp,Nome,Ruolo}(\sigma_{Sede = S01}(Imp))$.

SELECT senza proiezione

Se si vogliono tutti gli attributi:

```
SELECT CodImp, Nome, Sede, Ruolo, Stipendio
FROM Imp
WHERE Sede = „S01“
```

si può abbreviare con:

```
SELECT *
FROM Imp
WHERE Sede = „S01“
```

SELECT senza selezione (condizione)

- Con proiezione sugli attributi CodImp e Nome:

```
SELECT CodImp, Nome
FROM Imp
```

- Se si vogliono tutte le tuple

```
SELECT *
FROM Imp
```

restituisce tutta l‟istanza di Imp.

Tabelle : la clausola DISTINCT

- Il risultato di una query SQL può contenere righe duplicate:

```
SELECT Ruolo
FROM Imp
WHERE Sede = „S01"
```

Ruolo
Analista
Programmatore
Sistemista
Programmatore

- Per eliminarle si usa l"opzione DISTINCT nella SELECT list:

```
SELECT DISTINCT Ruolo
FROM Imp
WHERE Sede = „S01"
```

Ruolo
Analista
Programmatore
Sistemista

Espressioni complesse

All' interno di un comando select è possibile inserire espressioni booleane con operatori AND OR e NOT:

```
SELECT  Nome
FROM  Imp
WHERE  Sede = 'S01'
OR Ruolo = 'Programmatore'
```

```
SELECT  Nome
FROM  Imp
WHERE  Sede = 'S01'
AND Ruolo = 'Programmatore'
```

Nome	Sede	Ruolo
Rossi	S01	Analista
Verdi	S02	Sistemista
Bianchi	S01	Programmatore
Gialli	S03	Programmatore
Neri	S02	Analista
Grigi	S01	Sistemista
Violetti	S01	Programmatore
Aranci	S02	Programmatore

Operatore BETWEEN

- L‟operatore BETWEEN permette di esprimere condizioni di appartenenza a un intervallo:

Nome e stipendio degli impiegati che hanno uno stipendio compreso tra 1300 e 2000 Euro (estremi inclusi)

```
SELECT Nome, Stipendio
FROM   Imp
WHERE Stipendio BETWEEN 1300 AND 2000
```

Nome	Stipendio
Rossi	2000
Verdi	1500

- Lo stesso risultato si ottiene anche come segue:

```
SELECT Nome, Stipendio
FROM Imp
WHERE Stipendio >= 1300 AND Stipendio <= 2000
```

Operatore IN

- L‟operatore IN permette di esprimere condizioni di appartenenza a un insieme:

Codici e sedi degli impiegati delle sedi S02 e S03

```
SELECT CodImp, Sede
FROM   Imp
WHERE  Sede IN (“S02‟ , ”S03‟)
```

CodImp	Sede
E002	S02
E004	S03
E005	S02
E008	S02

- Lo stesso risultato si ottiene con gli operatori:

  - **“=ANY”**

    ```
    WHERE Sede = ANY (“S02‟ , “S03‟)
    ```

  - **“=“ + “OR”**

    ```
    WHERE Sede = “S02‟ OR Sede =  “S03‟
    ```

Operatore LIKE

- L'operatore **LIKE** **permette di esprimere "pattern" su stringhe**

Mediante "caratteri jolly" :

- _ (**un carattere arbitrario**)
- % (**una stringa arbitraria**)

Nomi degli impiegati che terminano con una „i" e hanno una „i" in seconda posizione

```
SELECT Nome
FROM   Imp
WHERE Nome LIKE "_i%i"
```

Nome
Bianchi
Gialli
Violetti

Espressioni nella clausola SELECT

- La SELECT list può contenere non solo attributi, ma anche espressioni:

```
SELECT CodImp, Stipendio*12
FROM   Imp
WHERE  Sede = "S01"
```

CodImp	
E001	24000
E003	12000
E006	13200
E007	12000

- Le espressioni possono comprendere anche più attributi.
- Si noti che in questo caso la seconda colonna non ha un nome.

Ridenominazione delle colonne

- Ad ogni elemento della SELECT list è possibile associare un nome a piacere:

```
SELECT CodImp AS Codice, Stipendio*12 AS StipendioAnnuo
FROM   Imp
WHERE  Sede = "S01"
```

Codice	StipendioAnnuo
E001	24000
E003	12000
E006	13200
E007	12000

- La parola chiave AS può anche essere omessa:

```
SELECT CodImp Codice,...
```

Ma per chiarezza è opportuno metterla sempre

Pseudonimi

- Per chiarezza ogni nome di colonna può essere scritto aggiungendo ad esso, come prefisso, il nome della tabella (obbligatorio in caso di ambiguità):

```
SELECT Imp.CodImp AS Codice, Imp.Stipendio*12 AS StipendioAnnuo
FROM   Imp
WHERE  Imp.Sede = "S01"
```

si può anche usare uno pseudonimo (alias) in luogo del nome della tabella

```
SELECT I.CodImp AS Codice, I.Stipendio*12 AS StipendioAnnuo
FROM Imp I -- oppure Imp AS I
WHERE I.Sede = "S01"
```

Valori nulli

- I valori nulli non sono considerati come dati, quindi la query:

```
SELECT CodImp
FROM Imp
WHERE Stipendio > 1500
OR Stipendio <= 1500
```

restituisce solo

CodImp
E001
E002
E003
E005
E007
E008

Imp

CodImp	Sede	...	Stipendio
E001	S01		2000
E002	S02		1500
E003	S01		1000
E004	S03		NULL
E005	S02		2500
E006	S01		NULL
E007	S01		1000
E008	S02		1200

Logica a 3 valori in SQL

- Nel caso di espressioni complesse, SQL ricorre alla **logica a 3 valori**:vero (V), falso (F) e “sconosciuto” (?).

```
SELECT CodImp, Sede, Stipendio
FROM   Imp
WHERE (Sede = “S03“)
 OR      (Stipendio > 1500)
```

CodImp	Sede	Stipendio
E001	S01	2000
E004	S03	NULL
E005	S02	2500

- Per verificare se un valore è NULL si usa l‟operatore **IS.**
  - NOT (**IS NULL) si scrive anche: IS NOT NULL.**

```
SELECT CodImp
FROM   Imp
WHERE  Stipendio IS NULL
```

CodImp
E004
E006

Ordinamento del risultato

- Per ordinare il risultato di una query secondo i valori di una o più colonne si introduce la clausola **ORDER BY**, **e per ogni colonna si** specifica se l'ordinamento è per valori "ascendenti" (**ASC**, **di** default) o "discendenti" **DESC**)

```
SELECT Nome, Stipendio
FROM   Imp
ORDER BY Stipendio DESC
```

Nome	Stipendio
Neri	2500
Rossi	2000
Verdi	1500
Aranci	1200
Grigi	1100
Bianchi	1000
Gialli	1000
Violetti	1000

Ordinamento e clausola TOP

- Può essere molto utile usare la clausola TOP in combinazione con ORDER BY.

Nome dell'impiegato con ruolo "Programmatore" che percepisce lo stipendio più basso

```
SELECT TOP(1) Nome, Stipendio
FROM Imp
WHERE Ruolo = 'Programmatore'
ORDER BY Stipendio
```

Nome	Stipendio
Bianchi	1000

```
SELECT TOP(1) WITH TIES Nome, Stipendio
FROM Imp
WHERE Ruolo = 'Programmatore'
ORDER BY Stipendio
```

Nome	Stipendio
Bianchi	1000
Gialli	1000
Violetti	1000

N.B. WITH TIES si può usare solo in presenza di ORDER BY e i "pareggi" (TIES) si riferiscono alla combinazione degli attributi di ordinamento.

Interrogazioni su più tabelle

- L'interrogazione

```
SELECT  I.Nome, I.Sede, S.Citta
FROM    Imp as I, Sedi as S
WHERE   I.Sede = S.Sede
AND     I.Ruolo = „Programmatore"
```

si interpreta come segue:

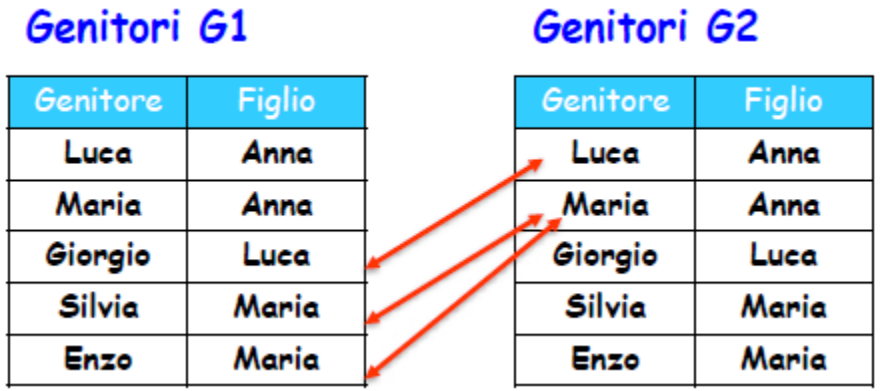
- si esegue il prodotto Cartesiano di Imp e Sedi;
- si applicano i predicati della clausola WHERE;
- si estraggono le colonne della SELECT list.

- Il predicato **I.Sede = S.Sede** **è detto** **predicato di join** **in** quanto stabilisce il criterio con cui le tuple di **Imp e di Sedi devono** essere combinate.

Self Join

- L‟uso di alias è forzato quando si deve seguire un self-join:

Chi sono i nonni di Anna?




```
SELECT   G1.Genitore AS Nonno
FROM     Genitori G1, Genitori G2
WHERE    G1.Figlio = G2.Genitore
AND      G2.Figlio = 'Anna'
```

Join espliciti

- Anziché scrivere i predicati di join nella clausola WHERE, è possibile "costruire" una **joined table** **direttamente nella clausola FROM:**

```
SELECT I.Nome, I.Sede, S.Citta
FROM Imp I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE I.Ruolo = "Programmatore"
```

in cui JOIN si può anche scrivere INNER JOIN.

- Altri tipi di join espliciti sono:
  - **LEFT [OUTER] JOIN**
  - **RIGHT [OUTER] JOIN**
  - **FULL [OUTER] JOIN**
  - **NATURAL JOIN**

Operatori insiemistici

- L‟istruzione SELECT non permette di eseguire unione, intersezione e differenza di tabelle.

- Ciò che si può fare è combinare in modo opportuno i risultati di dueistruzioni SELECT, mediante gli operatori:

UNION, INTERSECT, EXCEPT

- In tutti i casi gli elementi delle SELECT list devono avere tipi compatibili e gli stessi nomi se si vogliono colonne con un’intestazione definita.

- L‟ordine degli elementi è importante (notazione posizionale).

- Il risultato è in ogni caso privo di duplicati, per mantenerli occorre aggiungere l’opzione ALL:

UNION ALL, INTERSECT ALL, EXCEPT ALL

Operatori insiemistici: esempi (1)

R

A	B
1	a
1	a
2	a
2	b
2	c
3	b

S

C	B
1	a
1	b
2	a
2	c
3	c
4	d

```
SELECT A
FROM R
UNION
SELECT C
FROM S
```

1
2
3
4

```
SELECT B
FROM R
UNION ALL
SELECT B
FROM S
```

B
a
a
a
b
c
b
a
b
a
c
c
d

```
SELECT A
FROM R
UNION
SELECT C AS A
FROM S
```

A
1
2
3
4

```
SELECT A,B
FROM R
UNION
SELECT B,C AS A
FROM S
```

Non corretta!

Operatori insiemistici: esempi (2)

R

A	B
1	a
1	a
2	a
2	b
2	c
3	b

```
SELECT B
FROM R
INTERSECT
SELECT B
FROM S
```

B
a
b
c

```
SELECT B
FROM S
EXCEPT
SELECT B
FROM R
```

B
d

S

C	B
1	a
1	b
2	a
2	c
3	c
4	d

```
SELECT B
FROM R
INTERSECT ALL
SELECT B
FROM S
```

B
a
a
b
c

```
SELECT B
FROM R
EXCEPT ALL
SELECT B
FROM S
```

B
a
b

Funzioni aggregate (1)

- Lo standard SQL mette a disposizione una serie di funzioni aggregate (o “di colonna”):
  - **MIN minimo;**
  - **MAX massimo;**
  - **SUM somma;**
  - **AVG media aritmetica;**
  - **STDEV deviazione standard;**
  - **VARIANCE varianza;**
  - **COUNT contatore.**

```
SELECT  SUM(Stipendio) AS ToTStipS01
FROM    Imp
WHERE   Sede = ‘S01’
```

TotStipS01
5100

Funzioni aggregate (2)

- L'argomento di una funzione aggregata è una **qualunque espressione** che può figurare nella SELECT list (ma NON un'altra funzione aggregata!)

```
SELECT    SUM(Stipendio*12) AS ToTStipAnnuiS01
FROM      Imp
WHERE     Sede = 'S01'
```

TotStipAnnuiS01
61200

- Tutte le funzioni, ad eccezione di COUNT, ignorano i valori nulli.
- Il risultato è NULL se tutti i valori sono NULL.
- L'opzione DISTINCT considera solo i valori distinti:

```
SELECT    SUM(DISTINCT Stipendio)
FROM      Imp
WHERE     Sede = 'S01'
```

4100

COUNT e valori nulli

- La forma COUNT(*) **conta le tuple del risultato**; viceversa, specificando una colonna, si omettono quelle con valore nullo in tale colonna:

Imp

CodImp	Sede	...	Stipendio
E001	S01		2000
E002	S02		1500
E003	S01		1000
E004	S03		NULL
E005	S02		2500
E006	S01		NULL
E007	S01		1000
E008	S02		1200

```
SELECT   COUNT(*) AS NumImpS01
FROM     Imp
WHERE    Sede = 'S01'
```

NumImpS01
4

```
SELECT   COUNT(Stipendio)
AS NumStipS01
FROM     Imp
WHERE    Sede = 'S01'
```

NumStipS01
3

Funzioni aggregate e tipo del risultato

- Per alcune funzioni aggregate, al fine di ottenere il risultato desiderato, è necessario operare un *casting* dell'argomento:

Imp

...	Stipendio
	2000
	1500
	1000
	1000
	2500
	1100
	1000
	1200

```
SELECT    AVG(Stipendio) AS AvgStip
FROM      Imp          -- valore esatto 1412.5
```

AvgStip
1412

```
SELECT  AVG(CAST(Stipendio  AS  Decimal(6,2)))
        AS AvgStip FROM Imp
```

AvgStip
1412.50

Clausola SELECT e funzioni aggregate

- Se si usano funzioni aggregate, la SELECT list non può includere altri elementi che non siano a loro volta funzioni aggregate:

```
SELECT    Nome, MIN(Stipendio)
FROM      Imp
```

non va bene!
(viceversa, `SELECT MIN(Stipendio), MAX(Stipendio)..` è corretto)

- Il motivo è che una funzione aggregata restituisce un singolo valore, mentre il riferimento a una colonna è in generale un insieme di valori (eventualmente ripetuti).
- Nel caso specifico (chi sono gli impiegati con stipendio minimo?) è necessario ricorrere a un'altra soluzione, che vedremo più avanti.

Funzioni aggregate e raggruppamento

- I valori di sintesi calcolati dalle funzioni aggregate si riferiscono a **tutte le tuple che soddisfano le condizioni delle clausola WHERE.**
- Viceversa in molti casi è opportuno fornire i suddetti valori per **gruppi omogenei di tuple (es: impiegati di una stessa sede).**
- La clausola **GROUP BY serve a definire tali gruppi, specificando una o** più colonne (di raggruppamento) sulla base della/e quale/i le tuple sono raggruppate per va lori uguali.

```
SELECT    Sede, COUNT(*) AS NumProg
FROM      Imp
WHERE     Ruolo = 'Programmatore'
GROUP BY Sede
```

Sede	NumProg
S01	2
S03	1
S02	1

- **La SELECT list può includere solo le colonne di raggruppamento, ma non altre!**

Come si ragiona con il GROUP BY

- Le tuple che soddisfano la clausola WHERE...

CodImp	Nome	Sede	Ruolo	Stipendio
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

- ...sono raggruppate per valori uguali della/e colonna/e presenti nella clausola GROUP BY...

CodImp	Nome	Sede	Ruolo	Stipendio
E003	Bianchi	S01	Programmatore	1000
E007	Violetti	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

- ...e infine a ciascun gruppo si applica la funzione aggregata.

Sede	NumProg
S01	2
S03	1
S02	1

GROUP BY: esempi

1) Per ogni ruolo, lo stipendio medio nelle sedi di Milano

```
SELECT   I.Ruolo, AVG(I.Stipendio) AS AvgStip
FROM     Imp I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE    S.Citta = 'Milano'
GROUP BY I.Ruolo
```

Ruolo	AvgStip
Analista	2000
Sistemista	1100
Programmatore	1000

2) Per ogni sede di Milano, lo stipendio medio

```
SELECT   I.Sede, AVG(I.Stipendio) AS AvgStip
FROM     Imp I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE    S.Citta = 'Milano'
GROUP BY I.Sede
```

Sede	AvgStip
S01	1275
S03	1000

3) Per ogni ruolo e sede di Milano, lo stipendio medio

```
SELECT   I.Sede, I.Ruolo, AVG(I.Stipendio)
FROM     Imp I JOIN Sedi S ON (I.Sede = S.Sede)
WHERE    S.Citta = 'Milano'
GROUP BY I.Sede, I.Ruolo
```

Ruolo	Sede	
Analista	S01	2000
Sistemista	S01	1100
Programmatore	S01	1000
Programmatore	S03	1000

Raggruppamento e proiezione

- Quando la SELECT list include solo le colonne di raggruppamento, il risultato è equivalente a ciò che si otterrebbe omettendo il GROUP BY e rimuovendo i duplicati con l‟opzione DISTINCT.

```
SELECT    Sede
FROM      Imp
GROUP BY Sede
```

equivale pertanto a:

Sede
S01
S02
S03

```
SELECT DISTINCT Sede
FROM Imp
```

Condizioni sui gruppi

- Oltre a poter formare dei gruppi, è anche possibile **selezionare alcuni gruppi sulla base di loro proprietà "complessive"**:

```
SELECT    Sede, COUNT(*)  AS NumImp
FROM      Imp
GROUP BY Sede
HAVING COUNT(*) > 2
```

Sede	NumImp
S01	4
S02	3

- La clausola **HAVING ha per i gruppi una funzione simile a quella che ha la** clausola WHERE ha per le tuple (attenzione a non confonderle!).

Tipi di condizioni sui gruppi

- Nella clausola HAVING si possono avere due tipi di predicati:
  - predicati che fanno uso di **funzioni aggregate** **(es. COUNT(*) > 2);**
  - predicati che si riferiscono alle **colonne di raggruppamento; questi** ultimi si possono anche inserire nella clausola WHERE.

```
SELECT    Sede, COUNT(*) AS NumImp
FROM      Imp
GROUP BY  Sede
HAVING    Sede <> 'S01'
```

Sede	NumImp
S02	3
S03	1

equivale a:

```
SELECT    Sede, COUNT(*) AS NumImp
FROM      Imp
WHERE     Sede <> 'S01'
GROUP BY  Sede
```

Un esempio completo

Per ogni sede di Bologna in cui il numero di impiegati è almeno 3, si vuole conoscere il valor medio degli stipendi, ordinando il risultato per valori decrescenti di stipendio medio e quindi per sede

```
SELECT    I.Sede, AVG(Stipendio) AS AvgStipendio
FROM      Imp I, Sedi S
WHERE     I.Sede = S.Sede
AND       S.Citta = 'Bologna'
GROUP BY  I.Sede
HAVING    COUNT(*) >= 3
ORDER BY  AvgStipendio DESC, Sede
```

L'ordine delle clausole è sempre come nell'esempio.

Si ricorda che il GROUP BY non implica alcun ordinamento del risultato.

Subquery

- Oltre alla forma "**flat**" **vista sinora, in SQL è anche possibile** esprimere **condizioni che si basano sul risultato di altre interrogazioni** (**subquery**, **o query innestate o query nidificate**):

```
SELECT   CodImp    -- impiegati delle sedi di Milano
FROM     Imp
WHERE    Sede IN (SELECT   Sede
                  FROM     Sedi
                  WHERE    Citta = 'Milano')
```

Sede
S01
S03

- La subquery restituisce l'"insieme di sedi („S01",'"S03"), e quindi il predicato nella clausola WHERE esterna equivale a:

WHERE Sede IN ('S01','S03')

Subquery scalari

- Gli operatori di confronto =, <,… si possono usare solo se la subquery restituisce non più di una tupla (**subquery "scalare"**):

```
SELECT CodImp      -- impiegati con stipendio minimo
FROM Imp
WHERE Stipendio = (SELECT MIN(Stipendio)
                   FROM Imp)
```

MinStip
1000

- La presenza di vincoli può essere sfruttata a tale scopo:

```
SELECT Responsabile
FROM Sedi
WHERE Sede = (SELECT Sede -- al massimo una sede
              FROM Imp
              WHERE CodImp = 'E001')
```

Sede
S01

Subquery: caso generale

- Se la subquery può restituire più di un valore si devono usare le forme:
  - **<op> ANY: la relazione <op> vale per almeno uno dei valori;**
  - **<op> ALL: la relazione <op> vale per tutti i valori.**

```
SELECT Responsabile
FROM Sedi
WHERE Sede = ANY (SELECT Sede
                  FROM   Imp
                  WHERE Stipendio > 1500)
```

```
SELECT CodImp -- impiegati con stipendio minimo
FROM Imp
WHERE Stipendio <= ALL (SELECT Stipendio
                        FROM   Imp)
```

- La forma = ANY equivale a IN.