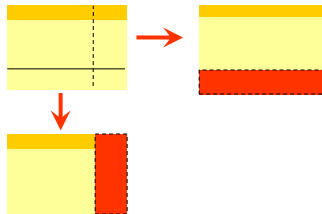


Il linguaggio SQL: le basi



Dario Maio
<http://bias.csr.unibo.it/maio/>

Annalisa Franco

Il linguaggio SQL: le basi 1

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).
- SQL adotta la **logica a 3** valori introdotta con l'Algebra Relazionale.

Il linguaggio SQL: le basi 2



SQL: standard e dialetti

- Il processo di standardizzazione di SQL è iniziato nel 1986.
- Nel 1992 è stato definito lo standard **SQL-2** (o SQL-92) da parte dell'**ISO** (International Standards Organization), e dell'**ANSI** (American National Standards Institute), rispettivamente descritti nei documenti ISO/IEC 9075:1992 e ANSI X3.135-1992 (identici!).
- Del 1999 è attivo lo standard **SQL:1999** che rende SQL un **linguaggio computazionalmente completo** (e quindi con istruzioni di controllo!) per il supporto di oggetti persistenti.
- Sono stati rilasciati altri standard nel 2003, 2006 e 2008. A seguito delle novità introdotte negli ultimi anni, SQL si sta trasformando in un linguaggio sempre più **procedurale**.
- Allo stato attuale **ogni sistema ha ancora un suo dialetto che:**
 - » è compatibile (in larga parte) con SQL-2;
 - » ha già elementi degli standard successivi;
 - » ha anche costrutti non standard.

Il linguaggio SQL: le basi



3



Gli standard SQL

Year	Name	Comments
1986	SQL-86	First formalized by ANSI.
1989	SQL-89	Minor revision, adopted as FIPS 127-1.
1992	SQL-92	Major revision (ISO 9075), Entry Level SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	Added regular expression matching, recursive queries, triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features.
2003	SQL:2003	Introduced XML-related features, window functions, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006	ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it enables applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.
2008	SQL:2008	Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers. Adds the TRUNCATE statement.

<http://en.wikipedia.org/wiki/SQL>

Il linguaggio SQL: le basi



4



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.



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



Modifica di tabelle

- Mediante l'istruzione **ALTER TABLE** è possibile modificare lo schema di una tabella, in particolare:

- aggiungendo 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'



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)



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`
- o 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)`



Politiche di "reazione"

- Aniché lasciare al programmatore il compito di garantire che a fronte di cancellazioni e modifiche i vincoli di integrità referenziale siano 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**.



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



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.*



SELECT su singola tabella

Codice, nome e ruolo dei dipendenti della sede S01

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

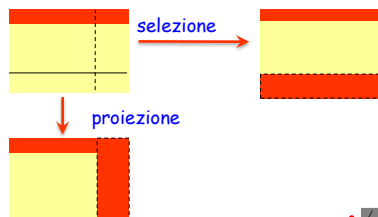
CodImp	Nome	Ruolo
E001	Rossi	Analista
E003	Bianchi	Programmatore
E006	Grigi	Sistemista
E007	Violetti	Programmatore

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



Selezione e proiezione in SQL

- Selezione e proiezione sono due operazioni "ortogonali":
 - **Selezione:** realizza una decomposizione orizzontale includendo nel risultato solo le ennuple che soddisfano i requisiti; produce un risultato che:
 - » Contiene tutti gli attributi dell'operando;
 - » Contiene solo alcune ennuple dell'operando.
 - **Proiezione:** realizza una decomposizione verticale includendo nel risultato solo gli attributi richiesti; produce un risultato che:
 - » ha solo una parte degli attributi dell'operando;
 - » contiene un numero di ennuple pari quelle dell'operando.
 - » **ATTENZIONE:** il risultato di una proiezione in SQL non è in generale una relazione poiché può contenere duplicati.



Il linguaggio SQL: le basi



15



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'
```

Il linguaggio SQL: le basi



16

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.

Il linguaggio SQL: le basi  17

Tabelle vs Relazioni: 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

Il linguaggio SQL: le basi  18



La clausola TOP

- La clausola TOP specifica **quante righe** deve restituire, come risultato, la query.
- L'insieme di righe da restituire può essere specificato come **numero** o come **valore percentuale**.
- È possibile mantenere più record se hanno lo stesso valore per uno o più attributi (**WITH TIES**).

```

SELECT TOP (numero|percentuale) [PERCENT] [WITH TIES] A1,A2,...,Am
FROM R1,R2,...,Rn
...

```

SQLServer, Access

- N.B. Non tutti i DBMS supportano la clausola TOP.
- Ciascun DBMS usa una propria sintassi:

MySQL

```

SELECT A1,A2,...,Am
FROM R1,R2,...,Rn
...
LIMIT numero

```

Oracle


```

SELECT A1,A2,...,Am
FROM R1,R2,...,Rn
...
AND ROWNUM <= numero

```

Il linguaggio SQL: le basi

19



La clausola TOP: esempi

```

SELECT CodImp, Nome
FROM Imp
WHERE Sede = 'S01'

```

CodImp	Nome
E001	Rossi
E003	Bianchi
E006	Grigi
E007	Violetti

```

SELECT TOP (2) CodImp, Nome
FROM Imp
WHERE Sede = 'S01'

```

CodImp	Nome
E001	Rossi
E003	Bianchi

```

SELECT TOP (20) PERCENT CodImp, Nome
FROM Imp
WHERE Sede = 'S01'

```

CodImp	Nome
E001	Rossi

Il linguaggio SQL: le basi

20



Espressioni complesse

- All'interno di un comando select è possibile inserire espressioni booleane con operatori AND e 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 le "wildcard":
- **_** (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

- A 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, ...
```



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'
```

...e 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'
```

Il linguaggio SQL: le basi



27



Valori nulli

- Il trattamento dei valori nulli si basa su quanto già visto in algebra relazionale, 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

Il linguaggio SQL: le basi



28



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 (A **IS NULL**) si scrive anche A **IS NOT NULL**.

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

CodImp
E004
E006

Il linguaggio SQL: le basi



29



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**, il 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

Il linguaggio SQL: le basi



30



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.

Il linguaggio SQL: le basi

31



Interrogazioni su più tabelle

- L'interrogazione

```
SELECT I.Nome, I.Sede, S.Citta
FROM Imp I, Sedi 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.

Il linguaggio SQL: le basi

32



Interrogazioni su più tabelle: risultato

- Dopo avere applicato il predicato **I.Sede = S.Sede**:

I.CodImp	I.Nome	I.Sede	I.Ruolo	I.Stipendio	S.Sede	S.Responsabile	S.Citta
E001	Rossi	S01	Analista	2000	S01	Biondi	Milano
E002	Verdi	S02	Sistemista	1500	S02	Mori	Bologna
E003	Bianchi	S01	Programmatore	1000	S01	Biondi	Milano
E004	Gialli	S03	Programmatore	1000	S03	Fulvi	Milano
E005	Neri	S02	Analista	2500	S02	Mori	Bologna
E006	Grigi	S01	Sistemista	1100	S01	Biondi	Milano
E007	Violetti	S01	Programmatore	1000	S01	Biondi	Milano
E008	Aranci	S02	Programmatore	1200	S02	Mori	Bologna

- **celle in blu**: dopo avere applicato il predicato **I.Ruolo = 'Programmatore'** e la proiezione.

Il linguaggio SQL: le basi



33



Ridenominazione del risultato

- Se la SELECT list contiene 2 o più colonne con lo stesso nome, è necessario operare una **ridenominazione** per ottenere un risultato in output con tutte le colonne dotate di intestazione:

```
SELECT  I.Sede AS SedeE001, S.Sede AS AltraSede
FROM    Imp I, Sedi S
WHERE   I.Sede <> S.Sede
AND     I.CodImp = 'E001'
```

SedeE001	AltraSede
S01	S02
S01	S03

Il linguaggio SQL: le basi



34



Self Join

- L'uso di alias è forzato quando si deve eseguire un self-join:

Chi sono i nonni di Anna?

Genitori G1

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

Genitori G2

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

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

Il linguaggio SQL: le basi



35



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

Il linguaggio SQL: le basi



36



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 due istruzioni 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

Il linguaggio SQL: le basi



37



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 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!

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

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

Il linguaggio SQL: le basi



38

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
c

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

B
a
b



Il linguaggio SQL: le basi **39**


Istruzioni di aggiornamento dei dati

- Le istruzioni che permettono di aggiornare il DB sono:

INSERT inserisce nuove tuple nel DB;

DELETE cancella tuple dal DB;

UPDATE modifica tuple del DB.
- **INSERT** può usare il risultato di una query per eseguire inserimenti multipli.
- **DELETE** e **UPDATE** possono fare uso di condizioni per specificare le tuple da cancellare o modificare.
- In ogni caso gli aggiornamenti riguardano una sola relazione.



Il linguaggio SQL: le basi **40**



Inserimento di tuple: caso singolo

- È possibile inserire una nuova tupla specificandone i valori:

```
INSERT INTO Sedi (Sede, Responsabile, Città)
VALUES          ('S04', 'Bruni', 'Firenze')
```

- Deve esservi **corrispondenza** tra attributi e valori.
- La lista degli attributi si può omettere, nel qual caso vale l'ordine con cui sono stati definiti.
- Se la lista non include tutti gli attributi, i restanti assumono valore **NULL** (se ammesso) o il valore di **default** (se specificato):

```
INSERT INTO Sedi (Sede, Città) -- sede senza responsabile
VALUES          ('S04', 'Firenze')
```



Inserimento di tuple: caso multiplo

- È possibile anche inserire le tuple che rappresentano il risultato di una query:

```
INSERT INTO SediBologna (SedeBO, Resp)
SELECT Sede, Responsabile
FROM   Sedi
WHERE  Città = 'Bologna'
```

- Gli schemi del risultato e della tabella in cui si inseriscono le tuple possono essere diversi, ma è necessario rispettare che i tipi delle colonne siano compatibili.



Cancellazione e modifica di tuple

- L'istruzione **DELETE** può fare uso di una condizione per specificare le tuple da cancellare:

```
DELETE FROM Sedi      -- elimina le sedi di Bologna
WHERE  Citta = 'Bologna'
```

- Anche l'istruzione **UPDATE** può fare uso di una condizione, per specificare le tuple da modificare, e di espressioni per determinare i nuovi valori:

```
UPDATE Sedi
SET    Responsabile = 'Bruni',
       Citta = 'Firenze'
WHERE  Sede = 'S01'

UPDATE Imp
SET    Stipendio = 1.1*Stipendio
WHERE  Ruolo = 'Programmatore'
```



Data Manipulation Language (DML)

- Le istruzioni principali del DML di SQL sono:

SELECT	esegue interrogazioni (query) sul DB;
INSERT	inserisce nuove tuple nel DB;
DELETE	cancella tuple dal DB;
UPDATE	modifica tuple del DB.

- **INSERT** può usare il risultato di una query per eseguire inserimenti multipli.
- **DELETE** e **UPDATE** possono fare uso di condizioni per specificare le tuple da cancellare o modificare.



SQL in sintesi

- Il **linguaggio SQL** è lo standard de facto per interagire con DB relazionali.
- Si discosta dal modello relazionale in quanto permette la presenza di **tuple duplicate** (**tabelle anziché relazioni**).
- La definizione delle tabelle permette di esprimere **vincoli** e anche di specificare politiche di reazione a fronte di violazioni dell'integrità referenziale.
- L'istruzione **SELECT** consiste nella sua forma base di 3 parti: **SELECT**, **FROM** e **WHERE**.
- A queste si aggiunge **ORDER BY**, per ordinare il risultato (e altre che vedremo).
- Per trattare i valori nulli, SQL ricorre a una **logica a 3 valori** (**vero, falso e sconosciuto**).