



Il linguaggio SQL: le basi

Sistemi Informativi L-A

Home Page del corso:

<http://www-db.deis.unibo.it/courses/SIL-A/>

Versione elettronica: [SQLa-basi.pdf](#)



SQL: caratteristiche generali

- SQL (Structured Query Language) è il linguaggio *standard de facto* per DBMS relazionali, che riunisce in sé funzionalità di DDL, DML e DCL
- SQL è 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**
 - ...inoltre SQL fa molte altre cose...
- 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



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 lo standard **SQL:1999**, che rende SQL un **linguaggio computazionalmente completo** (e quindi con istruzioni di controllo!) per il supporto di oggetti persistenti...
- Allo stato attuale **ogni sistema ha ancora un suo dialetto**:
 - supporta (in larga parte) SQL-2
 - ha già elementi di SQL:1999
 - ha anche costrutti non standard
- Quello che vediamo è la parte più “diffusa”



Organizzazione del materiale

- La trattazione di SQL viene suddivisa in più parti come segue:
 - DDL di base e DML “per gli operatori dell’algebra” e per le operazioni di modifica dei dati
 - Per fare “quello che si fa anche in algebra”
 - DML per il raggruppamento dei dati
 - Per derivare informazioni di sintesi dai dati
 - DML con blocchi innestati
 - Per scrivere richieste complesse
 - DDL per la definizione di viste e vincoli generici
 - Per migliorare la qualità dei dati
 - Utilizzo di SQL da linguaggio ospite
 - Per scrivere applicazioni



Data Definition Language (DDL)

- Il DDL di SQL permette di definire **schemi di relazioni** (o “**table**”, tabelle), modificarli ed eliminarli
- Permette di 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 si trattano completamente)
- Inoltre si possono definire **viste** (“view”), ovvero tabelle virtuali, e **indici**, per accedere efficientemente ai dati (questi ultimi li vedremo in SI L-B)



Per quanto non trattato nel seguito si faccia riferimento al materiale di laboratorio



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

DROP TABLE Imp



Definizione di tabelle: esempio

```
CREATE TABLE Imp (  
  CodImp      char(4)      PRIMARY KEY,  
  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  
)
```

```
CREATE TABLE Prog (  
  CodProg     char(3),  
  Citta       varchar(40),  
  PRIMARY KEY (CodProg,Citta) )
```



Valori nulli e valori di default

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

CF **char (16)** **NOT NULL,**

- Per ogni attributo è inoltre possibile specificare un valore di default, che verrà usato se all'atto dell'inserimento di una tupla non viene fornito esplicitamente un valore per l'attributo relativo

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)

- Ovviamente, specificare

UNIQUE (Cognome) ,
UNIQUE (Nome)

sarebbe molto più restrittivo

Chiavi primarie

- La definizione della chiave primaria di una tabella avviene specificando un vincolo **PRIMARY KEY**, o in linea o come vincolo di tabella

CodImp **char (4)** **PRIMARY KEY**

PRIMARY KEY (CodProg,Citta)

- Va osservato che:
 - 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



In DB2 è necessario specificare il vincolo NOT NULL sia per definire chiavi sia per definire chiavi primarie!

Chiavi straniere (“foreign key”)

- La definizione di una foreign key avviene specificando un vincolo **FOREIGN KEY**, e indicando quale chiave viene referenziata

Sede **char (3)** **REFERENCES** **Sedi (Sede)**

- Ovvero

FOREIGN KEY (Sede) **REFERENCES** **Sedi (Sede)**

- Nell’esempio, **Imp** è detta **tabella di riferimento** e **Sedi** **tabella di destinazione** (analogia terminologia per gli attributi coinvolti)
- Le colonne di destinazione devono essere una chiave della tabella destinazione (non necessariamente la chiave primaria)
- Se si omettono gli attributi destinazione, vengono assunti quelli della chiave primaria

Sede **char (3)** **REFERENCES** **Sedi**

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>. Pertanto

Stipendio int CHECK (Stipendio > 0) ,

non permette tuple con stipendio negativo, ma **ammette valori nulli per l'attributo Stipendio**

- 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)



In DB2 il CHECK può usare solo condizioni valutabili sulla singola tupla



Vincoli con nomi

- A fini diagnostici (e di documentazione) è spesso utile sapere quale vincolo è stato violato a seguito di un'azione sul DB
- A tale scopo è possibile associare dei **nomi ai vincoli**, ad esempio:

```
Stipendio int                                CONSTRAINT StipendioPositivo
                                              CHECK (Stipendio > 0),
```

```
CONSTRAINT ForeignKeySedi
    FOREIGN KEY (Sede) REFERENCES Sedi
```



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'
```



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

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
- La forma di base è:

```
SELECT  A1 ,A2 , . . . ,Am  
FROM    R1 ,R2 , . . . ,Rn  
WHERE   <condizione>
```

ovvero:

- **SELECT** (o **TARGET**) list (cosa si vuole come risultato)
- **clausola FROM** (da dove si prende)
- **clausola WHERE** (che condizioni deve soddisfare)

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 dice di prendere la tabella IMP
 - La clausola WHERE dice di prendere solo le tuple per cui Sede='S01'
 - Infine, si estraggono i valori degli attributi (o "colonne") nella SELECT list
- Equivale a `CodImp, Nome, Ruolo (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 condizione

- Se si vogliono tutte le tuple:

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

- Quindi

```
SELECT  *  
FROM    Imp
```

restituisce tutta l'istanza di Imp

Tabelle vs Relazioni

- 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 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

- 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, ...
```

Pseudonimi

- Per chiarezza, ogni nome di colonna può essere scritto prefissandolo con il nome della tabella:

```
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'
```


Operatore LIKE

- L'operatore **LIKE**, mediante le "wildcard" **_** (un carattere arbitrario) e **%** (una stringa arbitraria), permette di esprimere dei "pattern" su stringhe

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

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

Nome
Bianchi
Gialli
Violetti

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

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 scrivendo:

```
SELECT  CodImp, Sede
FROM    Imp
WHERE   Sede = 'S02' OR Sede = 'S03'
```

Valori nulli

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

```
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 (A IS NULL) si scrive anche A 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, il default) o “discendenti” (DESC)

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

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



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

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

Ridenominazione del risultato

- Se la SELECT list contiene 2 o più colonne con lo stesso nome, è necessario operare una ridenominazione per ottenere un output con tutte le colonne intestate

```
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

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'
```

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
```

DB2 non supporta il join naturale





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

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
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



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**



Inserimento di tuple: caso singolo

- È possibile inserire una nuova tupla specificandone i valori

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

- Ci deve essere **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 risultano da una query

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

- Valgono ancora le regole viste per il caso singolo
- Gli schemi del risultato e della tabella in cui si inseriscono le tuple possono essere diversi, l'importante è che i tipi delle colonne siano compatibili



Cancellazione 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'
```

- Che succede se la cancellazione porta a violare il vincolo di integrità referenziale? (ad es.: che accade agli impiegati delle sedi di Bologna?)
- ...lo vediamo tra 2 minuti



Modifica di tuple

- 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'
```

- Anche l'**UPDATE** può portare a violare il vincolo di integrità referenziale



Politiche di “reazione”

- Anziché 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**



Riassumiamo:

- 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**)