
Il linguaggio SQL

Di Roberta Molinari



SQL

Introduzione

Il linguaggio **Structured Query Language** è il linguaggio standard per le gestioni dei DB relazionali dal 1974. Nasce come dichiarativo, ma possiede costrutti procedurali. Esiste la possibilità di scrivere il codice direttamente in modo interattivo o tramite un'interfaccia grafica.

- *SQL embedded*: si trova all'interno di linguaggi tradizionali (*ospiti*) come C o C++
- *SQL Stand-alone*: interprete SQL interattivo o tramite programmi batch.

L'SQL svolge funzione di DDL, DML, DCL e QL

SQL

Caratteristiche generali

- ▶ Ogni comando termina con ;
 - ▶ Non è case sensitive
 - ▶ I nomi delle tabelle e degli attributi sono alfanumerici di 18 caratteri, devono iniziare con una lettera e possono contenere '_'
 - ▶ Per individuare un campo `NomeTab.NomeAttr`
 - ▶ Si utilizza il punto come segno decimale.
 - ▶ Le stringhe sono comprese tra ' o "
 - ▶ Funzioni su stringhe: `LENGTH()`, `LCASE()`, `UCASE()`, `trim()`,
`MID(column_name, start[, length])`, ...
 - ▶ I commenti su una riga iniziano con -- altrimenti `/* */`
-

SQL

Caratteristiche generali

- ▶ Operatori proposizionali AND OR NOT (&& || ! in MySql)
- ▶ Operatori aritmetici + - * / ^ sqrt() MOD DIV abs() ROUND(NUM,NDECIMALI) trunc()
- ▶ Operatori relazionali <, >, =, <>, <=, >=
- ▶ NULL: valore non presente o sconosciuto, non è =, <, > a nessun altro valore, operazioni matematiche con NULL restituiscono NULL (compaiono prima negli ordinamenti)

SQL

Tipi di dato: numerici

BIT BOOLEAN	1 bit	0/1 FALSE/TRUE
INT	4 Byte	-2miliardi ... 2miliardi
SMALLINT	2 Byte	-32768 ... 32767
FLOAT	4 Byte	Precisione singola
DOUBLE	8 Byte	Precisione doppia
DECIMAL (m, d)	<17 Byte	m cifre totali di cui d decimali per difetto (per def m=10 e d =0)

- ▶ Se si aggiunge la clausola `AUTO_INCREMENT` il campo diventa un contatore, se `UNSIGNED` disabilita i valor negativi

`INT UNSIGNED`

- ▶ Se si indica un numero tra parentesi rappresenta il numero di cifre totali rappresentate `INT (5)`



SQL

Tipi di dato: carattere

CHAR CHAR (n)	1 n Byte	Carattere Stringa di n caratteri ≤ 255 a dimensione fissa
VARCHAR VARCHAR (n)	1 n Byte	Stringa di n caratteri ≤ 255 a dimensione variabile (occupa meno spazio, ma più lento il reperimento)
CLOB		fino a 65.535 caratteri
TEXT		Non indicizzabile
BLOB		Bynary Large OBject: per immagini

SQL

Tipi di dato: data

DATE		'aaaa-mm-gg' da 1000-01-01 a 9999-12-31
TIME		'hh:mm:ss'
DATETIME		'aaaa-mm-gg hh:mm:ss'

In MySQL le date e le ore vanno espresse:

- come stringhe nel formato *'aa-mm-gg'* o *'aaaa-mm-gg'* e *'hh:mm:ss'* o *'hh:mm'*
- come un numero unico *aammgg* o *aaaammgg* o *mmgg* e le ore come *hhmmss* o *mmss*

In ACCESS invece per le date si usa *#aaaa/mm/gg#* mentre per le ore *#hh:mm:ss#* o *#hh:mm#*

SQL

Tipi di dato: corrispondenze

SQL	Access	SQLServer	Oracle	MySQL
<i>BOOLEAN</i>	Yes/No*	Bit	Byte	BOOL**
<i>INTEGER</i>	Number (integer)	Int	Number	INT INTEGER
<i>FLOAT</i>	Number (single)	Float Real	Number	FLOAT
<i>DECIMAL</i>	Currency	Money	N/A	DECIMAL
<i>CHAR</i>	N/A	Char	Char	CHAR
<i>VARCHAR</i>	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	VARCHAR
<i>BINARY OBJECT</i>	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	BLOB TEXT

*è tradotto in un BIT con 0=false e -1=true

**è tradotto in un TINYINT per cui 0 è falso e un qualsiasi valore <>0 è vero

SQL

Funzioni per le date in MySQL

- ▶ `CURDATE()`, `CURTIME()`, `datetime NOW()`
- ▶ `MONTH(d)`, `YEAR(d)`, `DAY(d)`
- ▶ `HOURL(d)`, `MINUTE(d)`, `SECOND(d)`
- ▶ `DATE_ADD(data, INTERVAL n MINUTE|HOUR|DAY|MONTH|YEAR)`
- ▶ `DATE_SUB(data, INTERVAL n MINUTE|HOUR|DAY|MONTH|YEAR)`
- ▶ `DATEDIFF(d1, d2)` **restituisce il numero di giorni tra le due date**
- ▶ `TIMEDIFF(h1, h2)` **restituisce il numero di hh:mm:ss tra le due ore**
- ▶ `STR_TO_DATE(str, format)` **restituisce una data a partire da una stringa interpretata secondo il formato specificato**
`STR_TO_DATE('01,5,2013', '%d,%m,%Y');` -> `'2013-05-01'`

SQL DDL

Per creare un database il comando è

```
CREATE DATABASE [IF NOT EXISTS] dbname;
```

Per creare le varie tabelle

```
CREATE TABLE [IF NOT EXISTS] nome_tab  
(lista_campi,  
  altre_specifiche);
```

IF NOT EXISTS non funziona in ACCESS

```
CREATE TABLE [IF NOT EXISTS] nometab  
(campo1 tipo1 [[NOT] NULL] [DEFAULT val]  
    [AUTO_INCREMENT] [UNIQUE] [, ...]  
    PRIMARY KEY (campo1 [, ...]) [,]  
    [FOREIGN KEY (campox)  
        REFERENCES tab1 (campoy)  
    [ON DELETE RESTRICT|CASCADE|SET NULL]  
    [ON UPDATE RESTRICT|CASCADE|SET  
    NULL] , ]  
    [ [CONSTRAINT chk1]  
        CHECK (campo espressione) , ]  
    [UNIQUE (campo1 [, ...]) ]  
) ;
```

i campi sono NULL di default.

AUTOINCREMENT/COUNTER in Access è un tipo intero

SQL

DDL check

► *espressione* può essere:

- `>= valore` o `<, <>, ...`
- `IN (val1, val2, ...)`
- `BETWEEN inizio AND fine`
- una combinazione con `AND` o `OR` delle precedenti
- Per le stringhe si possono usare le REGEX

`anno CHAR(4)`

`CHECK (anno REGEXP '^[0-9]{4}$')`

► Per definire domini (tipi) personalizzati (no in MySQL né Access)

```
CREATE DOMAIN (mioTipo) AS tipo  
CHECK (VALUE espressione);
```

SQL

DDL chiavi secondarie

- Per definire chiavi candidate o indici complessi

```
CREATE [UNIQUE] INDEX nomeind  
ON nomeiab (campox [,campoy]);
```

SQL DDL

CONSTRAINT in MariaDB

CONSTRAINT [*symbol*]] *constraint_expression*

constraint_expression:

| PRIMARY KEY [*index_type*] (*col_name*,
...) [*index_option*] ...

| FOREIGN KEY ...vedi dopo

| [UNIQUE] **INDEX** [*index_name*]
[*index_type*] (*col_name*, ...) [*index_option*] ...

| CHECK (*check_constraints*)

index_type: **USING** {BTREE | HASH | RTREE}



SQL DDL

FOREIGN KEY in MariaDB

```
[CONSTRAINT [symbol]]  
FOREIGN KEY [fk_name] (col_name, ...)  
REFERENCES tbl_name (col_name,...)  
[ON DELETE reference_option]  
[ON UPDATE reference_option]
```

reference_option: RESTRICT | CASCADE | SET
NULL | NO ACTION |

NO ACTION: Synonym for RESTRICT.

foreign keys are only supported by InnoDB.

SQL DDL

CHECK in MariaDB

Esempio di CHECK

```
CREATE TABLE t1 (  
  a INT CHECK (a>2),  
  b INT CHECK (b>2),  
  anno CHAR(4) CHECK (anno  
    REGEXP '^[0-9]{4}$'),  
  CONSTRAINT a_greater CHECK (a>b));
```

Se non si dà un nome al check ne darà uno in automatico

```
CREATE TABLE Persone
(Cod COUNTER NOT NULL,
 Nome varChar(20) NOT NULL ,
 Cognome varChar(20) NOT NULL ,
 CodNazione Integer NOT NULL,
 Natoil Date NOT NULL,
 Mortoil Date,
 PRIMARY KEY (Cod) ,
 FOREIGN KEY (CodNazione)
 REFERENCES NAZIONI (Codice)
) ;
```

SQL

Integrità referenziale

Nella dichiarazione delle FK si possono specificare le azioni da fare quando si cancella o modifica la PK di un padre utilizzando `ON DELETE`, `ON UPDATE` con le seguenti opzioni

- ▶ `CASCADE` si cancella/modifica i record dei figli
- ▶ `SET NULL` è attivabile solo se la FK non è `NOT NULL`. Le FK dei figli verranno impostate a `NULL`.
- ▶ `NO ACTION` o `RESTRICT` (default) impediscono la modifica o la cancellazione dei record della tabella padre. Sono sottintese per cui equivale a non impostare la clausola `ON DELETE|UPDATE`
- ▶ `SET DEFAULT` alle FK dei figli viene assegnato il valore impostato di default. Se non specificato e se è consentito sarà messo `NULL`

SQL

DDL

ALTER TABLE nomeTab

[ADD camponuovo tipo [NOT NULL] [, ...]]

[ADD PRIMARY KEY (campo [, ...])]

[ADD FOREIGN KEY (campo) REFERENCES ...]

[ADD INDEX index (campo1 [, ...]);

[ADD CONSTRAINT chk1 CHECK (campo
espressione)]

[DROP campoX|chkX [, ...]]

;

DROP TABLE nomeTab [RESTRICT|CASCADE];

DROP INDEX nomeind ON nomeTab; Access

ALTER TABLE nomeTab DROP INDEX index; MySQL

DROP DOMAIN mioTipo;

SQL

DML

INSERT INTO `nometab`

`[(campox [, ...])]`

VALUES

`(val1 [, ...]) ;`

Se non sono specificati i campi i valori sono assegnati secondo l'ordine di definizione. Per i campi di cui non si conosce il valore bisogna specificare `NULL` o non elencarli. Se la pk esiste già viene segnalato l'errore

In MySQL è possibile aggiungere più tuple con un solo comando aggiungendo più `(val2 [, ...])` separati da
" "
/

SQL DML

```
INSERT INTO Persone
```

```
(Nome,      Cognome,      NatoIl,      MortoIl,  
  CodNazione)
```

```
VALUES
```

```
('Mario',   'Rossi',   '1953-02-19',  NULL,  
  3);
```

SQL

DML

I dati si possono prelevare da una **SELECT** con la sintassi

```
INSERT INTO nometab1
```

```
SELECT campo1[, ...] [AS (nome1[, ...])]
```

```
FROM nometab2 [WHERE ...];
```

- ▶ Dove `nometab1` e `nometab2` possono anche coincidere
- ▶ È richiesto che i campi siano dello stesso tipo
- ▶ La tabella deve esistere

SQL

DML

- Per creare **copie** di una tabella (struttura+dati), per memorizzare in modo permanente i dati di una query, si crea una nuova **tabella** con

```
SELECT campo1[, ...]  
INTO nometabCopy [IN altroDB]  
FROM nometab [WHERE ...];
```

- Dove `nometab` e `nometabCopy` non possono coincidere e `altroDB` è il nome del file
- Attenzione: in questo caso le nuove tabelle vengono a fare parte del db e se vengono aggiornati dei dati nelle tabelle origine, quindi le modifiche non si ripercuotono nelle tabelle create precedentemente e viceversa con eventuali problemi di ridondanza e quindi di integrità

SQL

DML

- Per creare **copie** di una tabella (struttura+dati), per memorizzare in modo permanente i dati di una query, si crea una nuova **tabella** con **mySQL e SQLite**

```
CREATE TABLE nometabCopy  
SELECT *  
FROM nometab;
```

- Dove `nometab` e `nometabCopy` non possono coincidere

SQL

DML

Per creare una tabella vuota, ma con la stessa struttura di un'altra

```
SELECT campo1[, ...]  
INTO nometabCopy[IN altroDB]  
FROM nometab WHERE 0=1;
```

► In mySQL

```
CREATE TABLE nometabCopy  
SELECT *  
FROM nometab  
WHERE 0=1;
```

SQL

DML

DELETE FROM nomeTab

[WHERE cond] ;

- ▶ Se `cond` è verificata per un solo record si elimina una sola riga, altrimenti possono essere coinvolte più righe.
- ▶ Se non è specificata la clausola `WHERE` vengono cancellate tutte le righe
- ▶ Si può usare una subquery.

ES. Elimina tutti i giocatori dell'Inter

```
DELETE calciatori
WHERE calciatore.id_squadra=
(SELECT squadra_id
FROM squadre
WHERE squadra.nome="Inter");
```

SQL

DML

UPDATE *nometab*

SET *campox* = *espres* [, ...]

[**WHERE** *cond*];

- ▶ Se *cond* è verificata per un solo record si aggiorna una sola riga, altrimenti possono essere coinvolte più righe.
- ▶ Se non è specificata la clausola WHERE vengono modificate tutte le righe
- ▶ Si può fare anche utilizzando un'altra tabella (si possono usare subquery) tramite prodotto cartesiano di tabelle

ES. Sposta Totti all'Inter

```
UPDATE calciatori, squadre
SET calciatore.id_squadra=squadre.id_squadra
WHERE      calciatore.cognome="Totti"           AND
squadra.nome="Inter";
```

SQL

costrutto CASE

CASE

WHEN *condition1* THEN *result1*

...

WHEN *conditionN* THEN *resultN*

ELSE *result*

END;

SQL

costutto CASE in UPDATE

```
UPDATE example_table  
SET result = CASE  
    WHEN id=4 THEN 1  
    WHEN result=0 THEN 2  
    ELSE result  
END  
WHERE customer_id = 12;
```

SQL

costrutto CASE in SELECT

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END);
```

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is
greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```



SQL

QL – proiezione, alias

Proiezione

```
SELECT [DISTINCT|ALL|DISTINCTROW] campo  
        [AS nuovoNomeCampo] [, ...]
```

```
FROM nomeTab;
```

- ▶ Per default è `ALL`. `DISTINCT` restituisce righe univoche rispetto ai soli campi selezionati, mentre `DISTINCTROW` tiene conto anche degli altri campi della tabella in caso di join di tabelle (supportato solo da Access)
- ▶ Si possono creare degli alias per i campi con la clausola `AS` (consigliato per colonne calcolate). Se i nomi contengono spazi (SCONSIGLIATO) usare le `[]` in Access, `" "` in `mySQL`
- ▶ `campo` può anche essere un'espressione

```
SELECT (campo1+campo2) AS somma
```

SQL

QL - selezione, alias

Selezione

SELECT *

FROM nometab

WHERE cond;

Alias: per evitare di riscrivere per esteso il nome di una tabella in istruzioni complesse si possono usare abbreviazioni o alias.

Per es.

SELECT **A**.campo1 [, ...]

FROM nometab [**AS**] **A**

SQL

QL - condizioni di ricerca

- ▶ Contengono $<, >, =, <>, <=, >=,$
- ▶ Ordine degli operatori NOT, AND, OR
- ▶ [NOT] BETWEEN val1 AND val2 *estremi inclusi*
- ▶ [NOT] IN (val1, val2, ...)
- ▶ [NOT] LIKE '_abc%' *in Access ?≡_ *≡%*
- ▶ [NOT] IS NULL

Per confrontare date si possono usare le varie funzioni sulle date

SQL

QL - congiunzione

Congiunzione

```
SELECT campox [, ...]  
FROM tab1,tab2  
WHERE tab1.key1=tab2.key2;
```

Dove `key1` e `key2` sono i campi appartenenti allo stesso dominio

Si può anche fare (standard SQL-2)

```
SELECT campox [, ...]  
FROM tab1 [INNER] JOIN tab2  
      ON tab1.key1=tab2.key2;
```

Al posto dell'`ON` si può usare `USING (tab1.key1);`



SQL

QL - congiunzione

Esempio con 3 tabelle

```
SELECT campox
FROM tab1, tab2, tab3
WHERE tab1.key1=tab2.key2
      AND tab2.key=tab2.key;
```

oppure

```
SELECT campox
FROM tab3 INNER JOIN
  (tab1 INNER JOIN tab2 ON key1=key2)
ON tab2.key=tab2.key;
```

oppure

```
SELECT campox
FROM tab1 JOIN tab2 ON tab1.key1=tab2.key2 JOIN
tab3 ON tab2.key=tab3.key3;
```

SQL

QL – congiunzione esterna

Congiunzione esterna o outer join

```
SELECT tab1.*, tab2.*  
FROM tab1 [LEFT|RIGHT|FULL OUTER]  
JOIN tab2  
ON tab1.key1=tab2.key2;
```

Dove key1 e key2 sono i campi appartenenti allo stesso dominio.

La full outer join equivale all'unione di una left e una right join

```
SELECT * FROM tab1 LEFT JOIN tab2 ON  
tab1.key1=tab2.key2  
UNION  
SELECT * FROM tab1 RIGHT JOIN tab2 ON  
tab1.key1=tab2.key2;
```



SQL

QL - autocongiunzione

Autocongiunzione o self join

```
SELECT campox [, ...]  
FROM tab t1, tab t2  
WHERE t1.key1=t2.key2;
```

Dove `key1` e `key2` sono i campi appartenenti allo stesso dominio

Si può anche fare

```
SELECT t1.*, t2.*  
FROM tab AS t1 [INNER] JOIN tab AS t2  
ON t1.key1=t2.key2;
```

Al posto di `INNER` **posso anche scrivere** `LEFT` **o**
`RIGHT`



SQL

QL – DISTINCT, DISTINCTROW

Customers		
Cust ID	Company	City
1	ABC, Inc.	London
2	ABC, Inc.	Paris
3	Acme, Ltd.	New York

Orders			
Order	Cust ID	Date	Product
1	1	6/1	Access Analyzer
2	1	6/2	Access Statistics
3	2	6/3	Access Detective
4	3	6/3	Access Emailer

Notare che ABC di Londra ha 2 ordini

```
SELECT DISTINCT Company FROM Customers
JOIN Orders ON Customers.CustID =
Orders.CustID;
```

Company
ABC, Inc.
Acme, Ltd.

```
SELECT DISTINCTROW Company FROM Customers
JOIN Orders ON Customers.CustID =
Orders.CustID;
```

Company
ABC, Inc.
ABC, Inc.
Acme, Ltd.

NON TUTTI I DBMS LO SUPPORTANO

SQL

QL –DISTINCT DISTINCT ROW

Supponendo di avere
le seguenti tabelle

clienti		
ID	cogn	nome
1	aa	bb
2	aa	bb
3	aa	cc
4	ab	bb

ordini		
ID	tot	cli
1	€ 13,00	1
2	€ 275,00	1
3	€ 24,00	2
4	€ 53,00	2
5	€ 6,00	3
6	€ 7,00	4

```
SELECT DISTINCT cogn,nome
FROM clienti INNER JOIN ordini
  ON ID=cli;
```

ottiene:

cogn	nome
aa	bb
aa	cc
ab	bb

```
SELECT DISTINCTROW cogn,nome
FROM clienti INNER JOIN ordini
  ON ID=cli;
```

ottiene:

cogn	nome
aa	bb
aa	bb
aa	cc
ab	bb

SQL

QL- ordinamento

```
SELECT campo1, campo2, ...
```

```
FROM tab
```

```
WHERE ...
```

```
ORDER BY campo2 [ASC|DESC], campo1  
[ASC|DESC], ...;
```

- ▶ La clausola `ORDER` è l'ultima.
- ▶ `ASC` è il valore per default

SQL

QL- operatori insiemistici

Prodotto cartesiano o cross join

```
SELECT * FROM tab1, tab2;
```

tab1 e tab2 qualunque

Si creano tutte le possibili combinazioni delle tuple di tab1 con tutte le tuple di tab2

SQL

QL- operatori insiemistici

Es. Si desidera trovare tutte le possibili coppie

uomini	
Cognome	Nome
Rossi	Mario
Verdi	Luca
Ugo	Ughi

donne	
Cognome	Nome
Bianchi	Maria
Dutto	Lucia

```
SELECT    U.Cognome,    U.Nome,    D.Cognome,
          D.Nome
```

```
FROM uomini as U, donne as D ;
```

U.Cognome	U.Nome	D.Cognome	D.Nome
Rossi	Mario	Bianchi	Maria
Verdi	Luca	Bianchi	Maria
Ugo	Ughi	Bianchi	Maria
Rossi	Mario	Dutto	Lucia
Verdi	Luca	Dutto	Lucia
Ugo	Ughi	Dutto	Lucia

SQL

QL- funzioni di aggregazione

```
SELECT  COUNT ( *      |  [DISTINCT]  campox [,  
campoy, ... ] ) [AS NomeNuovo]
```

```
FROM tab
```

```
[WHERE ...] ;
```

- ▶ Viene restituita una sola riga Restituisce la cardinalità.
 - ▶ `COUNT (*)` conta le righe restituite dalla `SELECT`. Si usa
 - ▶ `COUNT (campox)` conta solo le righe in cui `campox` **NON** è vuoto.
 - ▶ `COUNT (DISTINCT campox)` conta le righe con valori diversi e non `NULL` di `campox`
-

SQL

QL- funzioni di aggregazione

SELECT **AVG** | **SUM** | **MIN** | **MAX** (campox | espr)

[AS NomeNuovo]

FROM tab [WHERE ...] ;

- ▶ Viene restituita una solo riga.
- ▶ Non vengono considerati i campi con valori `NULL`.
- ▶ `MIN`, `MAX`: con le stringhe si guarda il codice ASCII
- ▶ Insieme alle funzioni di aggregazione **NON** possono comparire altri campi nella `SELECT`

SQL

QL- raggruppamenti

```
SELECT campox[, campoy] [funzAggreg]  
FROM tab  
[WHERE ...]  
GROUP BY campox [, campoy]  
[HAVING cond];
```

- ▶ Tutti i campi dopo **SELECT** devono comparire dopo **GROUP** (non in MySQL). Dopo **GROUP** possono esserci altri campi (che magari rendono univoci).
- ▶ La **GROUP** comporta un ordinamento delle tuple in base all'ordine dei campi specificati (come **ORDER BY** sempre **ASC**)
- ▶ La clausola **HAVING** specifica delle condizioni sui gruppi creati da **GROUP** (normalmente controlla il valore restituito dalla funzione di aggregazione).

SQL

QL- ordine di esecuzione

SELECT...

FROM... JOIN...

WHERE...

GROUP BY...

HAVING...

Le operazioni vengono eseguite nel seguente ordine:

1. JOIN
2. WHERE
3. GROUP BY
4. proiezione in base ai campi selezionati
5. HAVING

SQL

QL- operatori insiemistici

Unione

```
SELECT * FROM tab1
```

```
UNION [ALL]
```

```
SELECT * FROM tab2;
```

- tab1 e tab2 devono avere gli stessi domini nello stesso ordine nella SELECT, la tabella risultato avrà i nomi della prima (usare alias se necessario)
- Con **UNION** si ottengono solo tuple distinte, mentre **UNION ALL** le conserva tutte
- più UNION (senza parentesi) si eseguono da sx verso dx
- Le singole SELECT non possono avere ORDER BY; dopo l'ultima SELECT può esserci un unico ORDER BY che viene applicato al risultato finale
- Le clausole GROUP BY e HAVING possono essere specificate per le singole clausole SELECT, ma non per il risultato finale

QL- operatori insiemistici

Es. Si desidera trovare tutte le date in cui è stata realizzata una transazione di vendite in negozio o via internet

StoreInfo		
StoreName	Sales	Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

InternetSales	
Date	Sales
07-Jan-1999	250
10-Jan-1999	535
11-Jan-1999	320
12-Jan-1999	750

Date	SELECT Date FROM StoreInfo
05-Jan-1999	UNION ALL
07-Jan-1999	SELECT Date FROM InternetSales;
08-Jan-1999	
08-Jan-1999	
07-Jan-1999	SELECT Date FROM StoreInfo
10-Jan-1999	UNION
11-Jan-1999	SELECT Date FROM InternetSales;
12-Jan-1999	

Date
05-Jan-1999
07-Jan-1999
08-Jan-1999
10-Jan-1999
11-Jan-1999
12-Jan-1999

SQL

QL- operatori insiemistici

Intersezione

```
SELECT * FROM tab1
```

INTERSECT

```
SELECT * FROM tab2
```

tab1 e tab2 devono avere gli stessi domini nello stesso ordine nella SELECT, la tabella risultato avrà i nomi della prima (usare alias se necessario)

Restituisce sempre tuple distinte

Non è supportato da molti DBMS perché equivalente a

```
SELECT DISTINCT a.campo1, a.campo2, ...
```

```
FROM tab1 a , tab2 b
```

```
WHERE (a.campo1=b.campo1 AND a.campo2=b.campo2, ...)
```

SQL

QL- operatori insiemistici

Es. si desidera trovare tutte le date corrispondenti sia alle vendite realizzate in negozio che quelle realizzate via Internet:

StoreInfo		
StoreName	Sales	Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

InternetSales	
Date	Sales
07-Jan-1999	250
10-Jan-1999	535
11-Jan-1999	320
12-Jan-1999	750

```
SELECT Date FROM StoreInfo
INTERSECT
SELECT Date FROM InternetSales;
```

Date
07-Jan-1999

```
SELECT DISTINCT s.Date
FROM StoreInfo s , InternetSales i
WHERE (s.Date=i.Date);
```

SQL

QL- operatori insiemistici

Differenza

```
SELECT * FROM tab1
```

MINUS | EXCEPT

```
SELECT * FROM tab2
```

tab1 e tab2 devono avere gli stessi domini nello stesso ordine nella SELECT, la tabella risultato avrà i nomi della prima (usare alias se necessario)

Restituisce sempre tuple distinte

Non è supportato da molti DBMS perché equivalente a

```
SELECT DISTINCT a.campo1, a.campo2, ...
```

```
FROM tab1 a LEFT JOIN tab2 b
```

```
ON (a.campo1=b.campo1 AND a.campo2=b.campo2, ...)
```

```
WHERE b.campox IS NULL
```

QL- operatori insiemistici

Es. Si desidera trovare tutte le date relative alle vendite realizzate in negozio, ma non quelle realizzate via Internet

StoreInfo			InternetSales	
StoreName	Sales	Date	Date	Sales
Los Angeles	1500	05-Jan-1999	07-Jan-1999	250
San Diego	250	07-Jan-1999	10-Jan-1999	535
Los Angeles	300	08-Jan-1999	11-Jan-1999	320
Boston	700	08-Jan-1999	12-Jan-1999	750

```
SELECT Date FROM StoreInfo
MINUS
SELECT Date FROM InternetSales;
```

Date
05-Jan-1999
08-Jan-1999

```
SELECT DISTINCT s.Date
FROM StoreInfo s LEFT JOIN InternetSales i
ON (s.Date=i.Date)
WHERE i.Date IS NULL;
```

SQL

QL- subquery

```
SELECT ...
```

```
FROM tab1
```

```
WHERE campox OperatoreConfronto
```

```
(SELECT campoy | funzAggreg | espres
```

```
from tab2
```

```
[WHERE ...])
```

- ▶ La subquery (in arancione) può restituire un valore, nessun valore o un insieme di valori, ma deve riferirsi ad una sola colonna o espressione
- ▶ Se la subquery restituisce valori scalari, cioè unici sia come numero di tuple che come attributi, OperatoreConfronto può essere

```
=      !=, <>
```

```
>, <
```

```
<=, =, >, !=, <
```

SQL

QL- subquery

- ▶ Se la subquery può restituire più di una tupla, OperatoreConfronto dovrà essere uno dei seguenti:
 - **ANY | SOME** : la condizione è FALSE se non viene restituito nulla dalla subquery o se il confronto è falso per tutti i valori restituiti (diventa es. `WHERE campo > ANY (SELECT ..)`)
 - **ALL** : la condizione è FALSE se il confronto è falso per almeno uno dei valori restituiti (diventa es. `WHERE campo > ALL (SELECT ..)`)
 - **[NOT] EXISTS** : la condizione è FALSE se non viene restituito nulla dalla subquery, ovvero si esegue la query esterna se quella interna restituisce almeno una tupla. Per quella interna si usa sempre `SELECT *` perché il numero di campi è irrilevante ed è più veloce (diventa `WHERE EXISTS (SELECT *..)`)
 - **IN** : equivale a `= ANY` (diventa `WHERE campo IN (SELECT ..)`)
 - **NOT IN** : equivale a `<> ALL` (diventa `WHERE campo NOT IN (SELECT ..)`)

SQL

QL- subquery CERCA IL MASSIMO

```
SELECT responsabili.* ,count(*)  
FROM Responsabili, SedeB  
WHERE codr = CodResp  
GROUP BY CodResp  
HAVING COUNT(*) >= ALL  
    (SELECT count(*)  
     FROM SedeB  
     GROUP BY CodR)
```

Non funziona in SQLite

SQL

QL- subquery CERCA IL MASSIMO

```
SELECT responsabili.* , count (*)  
FROM Responsabili, SedeB  
WHERE codr = CodResp  
GROUP BY CodResp  
HAVING COUNT (*) =  
    (SELECT COUNT (*)  
    FROM SedeB  
    GROUP BY CodR  
    ORDER BY COUNT (*) DESC  
    LIMIT 1)
```

In Access bisogna modificare la sintassi



SQL

QL- subquery CERCA IL MASSIMO

```
SELECT responsabili.* ,count(*)  
FROM Responsabili, SedeB  
WHERE codr = CodResp  
GROUP BY CodResp  
HAVING COUNT(*) =  
      (SELECT MAX(n)  
FROM (SELECT COUNT(*) as n  
        FROM SedeB  
        GROUP BY CodR)  
      )
```

SQL

QL- funzioni scalari

- ▶ Restituiscono un solo valore
- ▶ Vengono valutate per ogni riga estratta dalla query
- ▶ Non aumentano il potere espressivo, ma semplificano la scrittura delle query (altrimenti bisognerebbe fare un'unione di diverse query)
- ▶ Oltre alle funzioni sulle stringhe e a quelle matematiche, ci sono anche le seguenti, non supportati da tutti i DBMS
 - CASE (mySQL)
 - COALESCE (mySQL)
 - ISNULL(Access) o IFNULL(mySQL) non standard

SQL

SQL- CASE

```
SELECT [campo1,...,] CASE nomeColonna
      WHEN val1 THEN ris1
      WHEN val2 THEN ris2
      . . .
      [ELSE risN]
      END [AS "nuovo Nome Colonna"]
FROM nomeTabella;
```

In questo caso viene utilizzata per fornire il tipo di logica *switch-case* al linguaggio SQL. In base al valore di `nomeColonna`, vengono restituiti i vari `ris`. `val` può essere un valore statico o un'espressione. La colonna può essere rinominata usando `""` se ci sono degli spazi.

SQL

QL- CASE

Es. si desidera moltiplicare la quantità delle vendite da "Los Angeles" o "Boston" per 2 e la quantità delle vendite di "San Diego" per 1,5

```
SELECT StoreName, CASE StoreName
  WHEN 'Los Angeles' THEN Sales * 2
  WHEN 'Boston' THEN Sales * 2
  WHEN 'San Diego' THEN Sales * 1.5
  ELSE Sales
END AS NewSales,
  Date as TxnDate
FROM StoreInfo;
```

StoreInfo		
StoreName	Sales	Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
San Francisco	300	08-Jan-1999
Boston	700	08-Jan-1999

StoreName	NewSales	TxnDate
Los Angeles	3000	05-Jan-1999
San Diego	375	07-Jan-1999
San Francisco	300	08-Jan-1999
Boston	1400	08-Jan-1999



SQL

SQL- CASE

```
SELECT [campo1,...,] CASE  
    WHEN cond1 THEN ris1  
    WHEN cond2 THEN ris2  
  
    ...  
    [ELSE risN]  
    END [AS "nuovo Nome Colonna"]  
FROM nomeTabella;
```

In questo caso viene utilizzata per fornire il tipo di logica *if-then-else* al linguaggio SQL, in cui `cond` è un confronto (`<`, `>`, `=`, `...`) di una colonna con un valore statico o un'espressione o una `IN`, `IS NULL`.

Anche in questo caso la colonna può essere rinominata usando `""` se ci sono degli spazi.

SQL

QL- CASE

Es. si desidera moltiplicare la quantità delle vendite da "Los Angeles" o "Boston" per 2 e la quantità delle vendite di "San Diego" per 1,5

```
SELECT StoreName, CASE
    WHEN StoreName IN('Los Angeles', 'Boston') THEN
        Sales * 2
    WHEN StoreName = 'San Diego' THEN Sales * 1.5
    ELSE Sales
END AS NewSales, TxnDate FROM StoreInfo;
```

oppure

```
SELECT StoreName, CASE
    WHEN (StoreName = 'Los Angeles' OR StoreName =
        'Boston') THEN Sales * 2
    WHEN StoreName = 'San Diego' THEN Sales * 1.5
    ELSE Sales
END AS NewSales, TxnDate FROM StoreInfo;
```

SQL

SQL- COALESCE

La funzione `COALESCE` in SQL restituisce la prima espressione non-NULL presente tra i suoi argomenti.

```
SELECT COALESCE ( expression1 [ , ...n ] )  
      [AS nuovoNome]  
FROM nomeTabella;
```

che corrisponde alla seguente CASE

```
SELECT CASE  
      WHEN expression1 is not NULL THEN  
        expression1  
      WHEN expression2 is not NULL THEN  
        expression2  
      ...  
      [ELSE NULL]  
END [AS nuovoNome]  
FROM nomeTabella;
```

SQL

QL- COALESCE

Es.

si desidera trovare il modo migliore per contattare ogni persona in base alle seguenti regole:

1. telefono aziendale
2. telefono mobile
3. telefono del domicilio

si può fare:

ContactInfo			
Name	BusinessPhone	CellPhone	HomePhone
Jeff	531-2531	622-7813	565-9901
Laura	NULL	772-5588	312-4088
Peter	NULL	NULL	594-7477

```
SELECT Name, COALESCE (BusinessPhone, CellPhone,  
    HomePhone, 'nessun numero') AS ContactPhone  
FROM ContactInfo;
```

Name	ContactPhone
Jeff	531-2531
Laura	772-5588
Peter	594-7477

SQL- IFNULL o ISNULL

Restituisce il valore `value` se l'espressione `input` è nulla, altrimenti restituisce `input` (`value` avrà il tipo di `input`, se è un `char(3)` e `value = 'ciao'` sarà restituito 'cia')

`IFNULL (input, value)` `ISNULL()` in Access

Esempio:

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

```
SELECT ProductName, IFNULL(UnitsOnOrder, 'Nessun ordine') FROM Products;
```

In espressioni, per non avere NULL per il "Mascarpone"

```
SELECT ProductName, UnitPrice*(UnitsInStock+  
  IFNULL(UnitsOnOrder, 0))  
FROM Products;
```

SQL

QL- FORMAT

Restituisce il valore della colonna formattato come specificato dal secondo parametro

`FORMAT (campo, formato)`

Es.

```
SELECT Format(1222.4, '##,##0.00') -- "1,222.40"
SELECT Format(345.9, '###0.00') -- "345.90"
SELECT Format(15, '0.00%') -- "1500.00%"
SELECT Format(Now(), 'h:m:s') -- "1:10:47"
SELECT Format(Now(), 'hh:mm:ss tt') -- "01:10:47 AM"
SELECT Format(@date, 'dddd, MMM d yyyy') --
"Monday, Dec 22 2014"
```

SQL

QL- TOP o LIMIT

È possibile visualizzare solo un numero prestabilito di righe (normalmente ordinate con la clausola ORDER BY) usando

- ▶ in Access (si specifica PERCENT se si vuole l'n%)

```
SELECT TOP n [PERCENT] campo1[, campo2,...]  
FROM table;
```

- ▶ in MySQL

```
SELECT campo1[, campo2,...]  
FROM table[WHERE...] [GROUP BY...[HAVING ...]]  
LIMIT n;
```

oppure

LIMIT da, quanti;

la prima riga è 0

SQL-DCL

Creare viste

Una **vista** è una "tabella virtuale" le cui tuple sono il risultato di una query che viene valutata dinamicamente ogni volta che si fa riferimento alla vista. Le viste mettono a disposizione degli utenti rappresentazioni diverse degli stessi dati (livello esterno)

Sono:

- ▶ relazioni (tabelle) definite per mezzo di interrogazioni
- ▶ dotate di schema, ma prive di istanza (non occupano spazio)
- ▶ utilizzabili nelle interrogazioni come le tabelle di base

SQL-DCL

Creare viste

Una vista deve avere le seguenti caratteristiche:

- ▶ per crearla si può solo usare l'istruzione SELECT
- ▶ NON si può usare l'ORDER BY
- ▶ NON si possono utilizzare parametri di ingresso
- ▶ Sono memorizzate sul server DB al momento dell'utilizzo
- ▶ Possono essere usate come sorgenti dati di altre viste o query
- ▶ È possibile modificare i dati di una vista e queste si rifletteranno sulle tabelle reali

SQL-DCL

Creare viste

Le viste servono per:

- ▶ semplificare la scrittura di query molto complesse, con sottoespressioni ripetute
- ▶ formulare interrogazioni altrimenti non esprimibili
- ▶ introdurre meccanismi di personalizzazione e protezione delle tabelle (autorizzazioni di accesso rispetto alle viste)
- ▶ far fronte a modifiche dello schema logico che comporterebbero una ricompilazione dei programmi applicativi creando viste con il nome e la struttura delle vecchie tabelle ricavabili dalle nuove

SQL-DCL

Creare viste

Per creare una **vista**

```
CREATE VIEW nomevista [ (Attributi) ]  
AS SELECT...;
```

Nella definizione di una vista è possibile referenziare altre viste

Per usarla

```
SELECT ... FROM nomevista, ...;
```

Per modificarla

```
ALTER VIEW nomevista ...;
```

Per eliminarla

```
DROP VIEW nomevista;
```

SQL-DCL

Creare viste

- ▶ Le viste sono aggiornabili, ma ogni DBMS pone dei vincoli. I più comuni riguardano la non aggiornabilità di viste in cui il blocco più esterno della query di definizione contiene:
 - GROUP BY
 - Funzioni aggregate
 - DISTINCT
 - join (espliciti o impliciti)

In SQLite le view non sono modificabili con istruzioni DELETE, INSERT, or UPDATE

SQL-DCL

Gestione degli utenti

Per gestire gli utenti si possono impartire i comandi usando l'interfaccia grafica di phpMyAdmin → Privilegi (Account utenti)

Server: 127.0.0.1

Database SQL Stato Account utenti Più

Vista d'insieme degli account utente Gruppi utenti

Vista d'insieme degli account utente

Nome utente	Nome host	Password	Privilegi globali	Gruppo utenti	Grant	Azione
<input type="checkbox"/> Qualsiasi	%	No	USAGE		No	Modifica privilegi Esporta
<input type="checkbox"/> pma	localhost	No	USAGE		No	Modifica privilegi Esporta
<input type="checkbox"/> root	127.0.0.1	No	ALL PRIVILEGES	Si		Modifica privilegi Esporta
<input type="checkbox"/> root	::1	No	ALL PRIVILEGES	Si		Modifica privilegi Esporta
<input type="checkbox"/> root	localhost	No	ALL PRIVILEGES	Si		Modifica privilegi Esporta

Seleziona tutto Se selezionati: Esporta

Nuovo

Aggiungi account utente

Rimuove gli account utente selezionati

Revoca tutti i privilegi attivi agli utenti e dopo li cancella.

☐ Elimina i database che hanno gli stessi nomi degli utenti.

Esegui

SQL-DCL

Gestione degli utenti

Per modificare i loro permessi sulle varie tabelle/viste, si deve selezionare il db e quindi si possono impartire i comandi usando l'interfaccia grafica di phpMyAdmin → Privilegi

Server: 127.0.0.1 » Database: test

Struttura SQL Cerca Query da esempio Esporta Importa Operazioni Privilegi Routine Più

Utenti che hanno accesso a "test"

Nome utente	Nome host	Tipo	Privilegi	Grant	Azione
<input type="checkbox"/> Qualsiasi	%	specifico del database	SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, REFERENCES, INDEX, ALTER, CREATE TEMPORARY TABLES, LOCK TABLES, CREATE VIEW, EVENT, TRIGGER, SHOW VIEW, CREATE ROUTINE	No	Modifica privilegi Esporta
<input type="checkbox"/> root	127.0.0.1	globale	ALL PRIVILEGES	Sì	Modifica privilegi Esporta
<input type="checkbox"/> root	::1	globale	ALL PRIVILEGES	Sì	Modifica privilegi Esporta
<input type="checkbox"/> root	localhost	globale	ALL PRIVILEGES	Sì	Modifica privilegi Esporta

↑ ☐ Seleziona tutto Se selezionati: Esporta

Nuovo

Aggiungi account utente

SQL-DCL

Gestione degli utenti

Per aggiungere un utente

```
GRANT CONNECT TO utente1  
[IDENTIFIED BY password]
```

In MySQL

```
CREATE USER username1  
[IDENTIFIED BY password]
```

username1 **è nella forma** 'nome'@'localhost'
e password **è una stringa tipo** 'miapwd'

Per eliminare un utente

```
DROP USER user [, user] ...
```

SQL-DCL

Gestione dei permessi

GRANT permesso

ON tabella //o view

TO utente1 [,utente2,...]

[WITH GRANT OPTION]

REVOKE permesso

ON tabella //o view

TO utente1 [,utente2,...]

SQL-DCL

Gestione dei permessi

Il permesso può essere `ALL PRIVILEGES` (per poter fare tutto) o una lista separata da `,` di:

- `SELECT [(col1 [, col2, ...])]` per selezionare righe (non specifico niente per tutte)
- `INSERT` per inserire nuove righe
- `UPDATE [(col1 [, col2, ...])]` per modificare righe
- `DELETE` per eliminare righe
- `ALTER` per aggiungere o eliminare colonne o modificare i tipi di dati
- `INDEX` per creare indici
- `CREATE, DROP` per creare o eliminare DB

`WITH GRANT OPTION` permette all'utente a cui sono concesse le operazioni di concederle a sua volta ad altri utenti