SQL
Manuale introduttivo

SOMMARIO

INTRODUZIONE

L'acronimo SQL sta per Standard Query Language, un linguaggio creato per interrogare database relazionali, reso molto noto dalla Oracle Inc. Attualmente lo standard SQL è utilizzato da quasi tutti gli RDBMS (Relational DataBase Management Systems, ovvero gli strumenti per la creazione e la gestione dei database) e rappresenta un linguaggio completo per la gestione di una base dati. L'SQL include inoltre anche funzionalità di DDL (Data Desctiption Language), di DCL (Data Control Language) e di DML (Data Management Language).
Nonostante SQL sia ormai uno standard de facto, esistono comunque delle differenze di implementazione dell'SQL a seconda dell'RDBMS utilizzato. Ad ogni modo le differenze sono lievi e facilmente individuabili attraverso l'help on line ormai presente in tutti i programmi.

TIPI DI DATI

Di seguito vedremo come si definiscono i tipi di dati gestibili da SQL. Chiaramente, trattandosi proprio di un linguaggio orientato ai dati, vedrete che ve ne sono parecchi.

STRINGHE DI CARATTERI

In SQL si possono definire due tipi di stringhe di caratteri:

- a dimensione fissa (a destra sono completate da uno spazio)
- a dimensione variabile

Per definire una stringa di dimensione fissa si possono utilizzare le seguenti sintassi:
CHARACTER | CHARACTER(<*dimensione*>)
CHAR | CHAR(<*dimensione*>)
Se tra le parentesi non si indica la lunghezza della stringa, SQL intenderà lunghezza=1.

Per definire invece una stringa di dimesione variabile, si può utilizzare una delle seguenti sintassi:
CHARACTER VARYING(<*dimensione*>)
CHAR VARYING(<*dimensione*>)
VARCHAR(<*dimensione*>)
Bisogna ricordarsi, però, di indicare la dimensione massima della stringa (il minimo è rappresentato dalla stringa nulla)

NUMERI

In SQL si possono avere valori ESATTI e valori APPROSSIMATI. Si ha il primo caso quando si conosce il massimo numero di cifre numeriche intere e decimali che si dovrà utilizzare. Il tipo approssimato viene, invece, utilizzato quando si ha a che fare con i numeri a virgola mobile. **NUMERIC** permette di definire un valore numerico composto come indicato dai parametri tra parentesi (con SCALA si intendono i valori che vogliamo riservare per le cifre dopo la virgola). Se non si specifica la scala, si avrà a che fare solo con valori interi. Se non si indica nemmeno la precisione, SQL adotterà quella di default del sistema RDBMS in uso.

NUMERIC | NUMERIC(<*precisione*>[,<*scala*>])

DECIMAL è simile a **NUMERIC**, ma le caratteristiche della precisione e della scala indicate rappresentano le esigenze minime che ha l'utente, ed il sistema potrà fornire una rappresentazione con precisione o scala maggiore.

DECIMAL | DECIMAL(<*precisione*>[,<*scala*>])

DEC | DEC(<*precisione*>[,<*scala*>])

INTEGER e **SMALLINT** sono tipi interi la cui dimensione dipende dalle caratteristiche del sistema operativo e dall'hardware in uso. Ad ogni modo, **SMALLINT** permette di rappresentare interi con precisione inferiore o uguale a **INTEGER**:

INTEGER | INT

SMALLINT

Il tipo **FLOAT** è un tipo numerico approssimato (cioè a virgola mobile) ed ha una precisione binaria pari o maggiore di quella indicata tra le parentesi (può anche non essere indicata, lasciando al sistema la scelta).

FLOAT | FLOAT(<*precisione*>)

REAL e **DOUBLE PRECISION** sono due tipi a virgola mobile con precisione predefinita. La precisione dipende dal sistema, anche se, in generale, **DOUBLE PRECISION** fornisce una precisione maggiore di REAL.

REAL

DOUBLE PRECISION

DATA/ORA

Per indicare data/ora si possono utilizzare diverse sintassi. La prima serve per memorizzare un giorno specifico(anno-mese-giorno):
DATE
Le sintassi seguenti, invece, memorizza un'informazione data/orario completa (TIME fornisce ore-minuti-secondi e, se si desidera, frazioni di secondo; TIMESTAMP fornisce le informazioni complete e, se si indica la precisione, si specifica anche la parte frazionaria dei secondi, altrimenti questi non vengono mostrati. Se si aggiunge WITH TIME ZONE, si chiede che l'orario venga posto nel formato di una particolare area geografica):
TIME WITH TIME ZONE | TIME(<*precisione*>) WITH TIME ZONE
TIMESTAMP | TIMESTAMP(<*precisione*>)
TIMESTAMP WITH TIME ZONE | TIMESTAMP(<*precisione*>) WITH TIME ZONE
La sintassi seguente, invece, è utile per registrare un orario:
TIME | TIME(<*precisione*>)

COSTANTI STRINGA

Vengono indicate ponenedole tra apici singoli, oppure doppi, a volte a seconda dell'RDBMS. In ogni caso l'apice singolo è accettato da tutti gli RDBMS.
'esempio di stringa'
"esempio di stringa"

COSTANTI NUMERICHE

Le costanti numeriche vengono indicate semplicemente dal numero senza delimitatori. La virgola (per i decimali) si indica con il punto (**.**)

COSTANTI DATA/ORA

Vengono indicate come le stringhe e delimitate da apici. A seconda del sistema RDBMS potrebbe prevedere più forme, ma, in generale, negli esempi riportati di seguito, potete ritrovare i modi più comuni per indicare costanti data/ora.
Esempi per rappresentare il 31/12/2000:
'2000-12-31'
'2000/12/31'
'2000.12.31'
Tipi diversi di rappresentazione dell'orario:
'12:30:50.10'
'12:30:50'
'12:30'
Informazioni complete data/ora (tipo TIMESTAMP)
'1999-12-31 12:30:50.10'
'1999-12-31 12:30:50'
'1999-12-31 12:30'

NOTE SULLA NOMENCLATURA

Anche se spesso vi capiterà di trovare le istruzioni del linguaggio scritte in lettere maiuscole, questo stile di scrittura è utile solo per individuare immediatamente le parole chiave. In realtà si tratta solo di una convenzione, poiché SQL non distingue tra maiuscole e minuscole né per le istruzioni, né per i nomi di tabelle, di colonne e di altri oggetti del database. La distinzione avviene solo quando si assegna il contenuto di una variabile: in questo caso dovremmo stare attenti alle differenze!
Le istruzioni possono essere scritte su righe diverse o tutte una di seguito all'altra, sulla stessa riga. In alcuni sistemi si usa un simbolo per indicare la fine della riga: generalmente un punto e virgola.
Per inserire commenti, occore farli precedere da un doppio trattino (--).
Nei nomi assegnati agli oggetti del database si possono utilizzare lettere, numeri e anche l'underscore. Quest'ultimo può essere anche il primo carattere che, in ogni caso, deve essere SEMPRE una lettara.

OPERATORI, FUNZIONI ED ESPRESSIONI

Prima di passare a descrivere la sintassi per le istruzioni SQL, parleremo degli operatori. Si usa dire, infatti, che i "verbi" fondamentali di SQL siano 5:

- CREATE
- SELECT
- DELETE
- INSERT
- UPDATE

Prima di affrontare questo argomento, però, così come abbiamo fatto per i tipi di dati, parleremo degli operatori, in modo che, una volta arrivati a descrivere i verbi che permettono di scrivere le istruzioni di linguaggio, avremo tutti gli elementi per operare effettivamente su una base di dati.
Posta questa introduzione, dunque, vediamo quali operatori, espressioni e funzioni vengono messi a disposizione di SQL per agire sugli oggetti di un database.
pur non essendo un linguaggio di programmazione completo, mette a disposizione una serie di operatori e di funzioni utili per la realizzazione di espressioni di vario tipo.

OPERATORI ARITMETICI

Nella tabella che segue, sono riportati tutti gli operatori che agiscono su valori numerici. Tutti i tipi, siano essi esatti o approssimati, possono essere usati, anche perché, dove vi fosse necessità, è il sistema RDBMS stesso che provvede ad eseguire le eventuali necessarie conversioni di tipo.

Operatori/operandi	Descrizione
-< *oper*>	Effettua l'inversione ell'operando (da negativo a positivo e viceversa)
< *oper1*> + < *oper2*>	Somma tra due operandi.
< *oper1*> - < *oper2*>	Sottrazione tra due operandi.
< *oper1*> * < *oper2*>	Moltiplicazione di due operandi.
< *oper1*> / < *oper2*>	Divisione tra due operandi
< *oper1*> % < *oper2*>	Modulo, ovvero l'operazione fornisce il resto della divisione tra il primo e il secondo operando.

OPERATORI DI CONFRONTO E OPERATORI LOGICI

In tutti i linguaggi gli operatori di confronto sono molto importanti: forniscono la possibilità di mettere in relazione tra loro due elementi (mediante il confronto maggiore, minore, uguale, ecc.) Il risultato del confronto è un booleano, cioè un valore VERO o FALSO.
Nella tabella a seguire è indicata la sintassi degli operatori di confronto disponibili in SQL.

Operatori/operandi	Descrizione
< *oper1*> = < *oper2*>	*Vero* quando gli operandi si equivalgono.
< *oper1*> <> < *oper2*>	*Vero* quando gli operandi sono differenti.
< *oper1*> < < *oper2*>	*Vero* quando il primo operando è minore del secondo.
< *oper1*> > < *oper2*>	*Vero* quando il primo operando è maggiore del secondo.
< *oper1*> <=< *oper2*>	*Vero* quando il primo operando è minore o uguale al secondo.
< *oper1*> >= < *oper2*>	*Vero* quando il primo operando è maggiore o uguale al secondo.

Gli operatori LOGICI, invece, si utilizzano quando si vogliono combinare tra loro più espressioni logiche. Ovviamente, con le parentesi tonde si possono raggruppare i confronti in modo da dare priorità diverse nella valutazione delle espressioni che scriviamo.
Nella tabella sono riportate le sintassi degli operatori logici:

Operatori/operandi	Descrizione
NOT <*oper*>	Inverte il risultato logico dell'operando.
< *oper1*> AND < *oper2*>	*Vero* se tutti e due operandi restituiscono il valore *Vero*.
< *oper1*> OR < *oper2*>	*Vero* se almeno uno degli operandi restituisce il valore *Vero*.

Con le stringhe, oltre agli operatori visti nella tabella precedente, si possono eseguire confronti attraverso gli operatori **IS LIKE** e **IS NOT LIKE**. Possono essere utilizzati anche caratteri jolly (il simbolo underscore (_), che rappresenta un singolo carattere qualsiasi, e il simbolo di percentuale (**%**), che rappresenta una sequenza qualsiasi di caratteri), come evidenziato nella tabella che segue:

Espressioni	Descrizione
<*stringa1*> IS LIKE < *stringa2*>	Restituisce *Vero* quando la stringa1 corrisponde alla stringa2
< *stringa1*> IS NOT LIKE < *stringa2*>	Restituisce *Vero* quando la stringa1 non corrisponde alla stringa2
_	Rappresenta un carattere singolo qualsiasi.
%	Rappresenta una sequenza indeterminata di caratteri.

Vi sono poi goi operatori che consentono il controllo delle espressioni, come quelli, il cui significato è intuitivo, riportati di seguito:

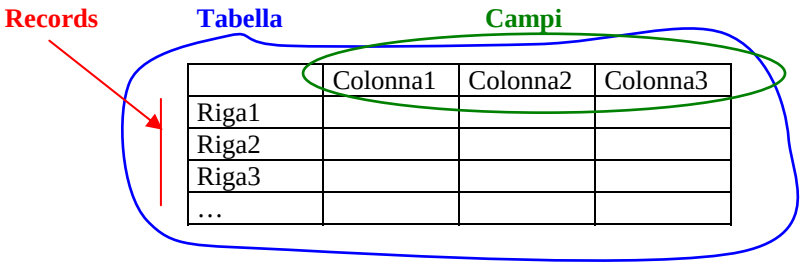
Operatori	**Descrizione**
<*espressione*> IS NULL	Restituisce *Vero* se l'espressione genera un risultato indeterminato.
<*espressione*> IS NOT NULL	Restituisce *Vero* se l'espressione non genera un risultato indeterminato.

Per concludere, vi sono degli operatori che permettono di verificare se un valore appartiene o meno ad un intervallo o ad un elenco di valori:

Operatori/operandi	**Descrizione**
<*oper1*> IN (<*elenco*>)	*Vero* se il primo operando è presente nell'elenco.
< *oper1*> NOT IN (<*elenco*>)	*Vero* se il primo operando non è presente nell'elenco.
< *oper1*> BETWEEN < *oper2*> AND < *oper3*>	*Vero* se il primo operando è compreso nell'intervallo indicato dal secondo e dal terzo.
< *oper1*> NOT BETWEEN < *oper2*> AND < *oper3*>	*Vero* se il primo operando non è compreso nell'intervallo indicato.

TABELLE E RELAZIONI

In questo manuale parleremo esclusivamente di SQL, dando per scontato che chi si accinge a studiare questo linguaggio, conosca le regole per modellare un database relazionale ed abbia familiarità con il modello entity-relationship stesso.
Le relazioni sono trattate da SQL attraverso il modello tabellare e di conseguenza sono le tabelle stesse il cardine della creazione di una base dati.
Una tabella è costituita dall'insieme di più righe. Una riga (record) è una sequenza non vuota di valori. Ogni riga ha la stessa cardinalità (e cioè contiene lo stesso numero di campi). Ogni colonna corrisponde ad un campo.
Per grado di una tabella si intende il numero di colonne (campi che la costituisce.
La tabella, che viene identificata da un nome, è un insieme di informazioni organizzato in righe (senza nome, ovvero i record dati) e colonne (cui viene attribuito un nome – campi).
In figura si è cercato di schematizzare quanto scritto:



CREARE UNA TABELLA

Per creare una tabella si utilizza il comando CREATE TABLE. La sintassi più semplice è la seguente:
CREATE TABLE <*nome-tabella*> (<*specifiche*>)
Possiamo però specificare direttamente tra parentesi (in genere si fa così) i nomi dei campi, e quindi la sintassi, estesa, diventa:
CREATE TABLE <*nome-tabella*> (<*nome-colonna*> <*tipo*>[,...])
Come si evince, dopo il nome della tabella si indicano i nomi dei campi seguiti dal tipo di valori che conterranno e dalla dimensione. Ecco un esempio:

```
CREATE TABLE Anagrafica (
                Identificativo      integer,
                Cognome             char(40),
                Nome                char(40),
                Indirizzo           varchar(60),
                Telefono            varchar(40)
        )
```

Un'altra opzione che si può utilizzare quando si crea una tabella è quella di inizializzare i campi, ovvero inserire valori iniziali per un dato campo, come si può vedere nell'esempio:

```
CREATE TABLE Anagrafica (
                Identificativo      integer,
                Cognome             char(40),
                Nome                char(40),
                Indirizzo           varchar(60) DEFAULT 'sconosciuto',
                Telefono            varchar(40)
        )
```

NOTA: notate la virgola dopo ogni riga che specifica il nome del campo, il tipo, le dimensioni e l'eventuale valore di default. Solo sull'ultima riga prima della parentesi, non si deve porre la virgola.

Potrebbe capitare che all'interno di una riga, alcuni valori non siano ammissibili. In questo caso si possono porre alcuni **VINCOLI INTERNI** alla tabella, come si vede nella sintassi che segue:

```
CREATE TABLE <nome-tabella> (
        <nome-colonna> <tipo>
                [NOT NULL]
        [,...]
    )
```

Eccone l'esempio corrispondente:

```
CREATE TABLE Anagrafica(
        Identificativo      integer         NOT NULL,
        Cognome             char(40)        NOT NULL,
        Nome                char(40)        NOT NULL,
        Indirizzo           varchar(60)     DEFAULT 'sconosciuto',
        Telefono            varchar(40)     NOT NULL
    )
```

Se invece si vuole indicare che in uno (o più) campi della tabella non siano inseriti dati ripetuti, ovvero che i dati siano presenti in modo univoco, si dovrà utilizzare la specifica UNIQUE, accanto al quale, tra parentesi, si indica il campo/i campi di interesse):

```
CREATE TABLE <nome-tabella> (
        <nome-colonna> <tipo>
        [,...],
        UNIQUE ( <nome-colonna>[,...] )
        [,...]
    )
```

Ecco l'esempio corrispondente:

```
CREATE TABLE Anagrafica (
        Identificativo      integer         NOT NULL,
        Cognome             char(40)        NOT NULL,
        Nome                char(40)        NOT NULL,
        Indirizzo           varchar(60)     DEFAULT 'sconosciuto',
        Telefono            varchar(40)     NOT NULL,
        UNIQUE (Identificativo)
    )
```

Se una colonna (cioè un campo) costituisce un valore univoco attraverso il quale si identifica un record, si utilizza il vincolo **PRIMARY KEY.** In una tabella può essere presente UNA SOLA primary key. Il vincolo stabilisce che i dati contenuti, oltre a non poter essere ripetuti, non possono essere indefiniti.

Sintassi:

```
CREATE TABLE <nome-tabella> (
        <nome-colonna> <tipo>
        [,...],
        PRIMARY KEY ( <nome-colonna>[,...] )
    )
```

Esempio:

```
CREATE TABLE Anagrafica (
        Identificativo      integer,
        Cognome             char(40)        NOT NULL,
        Nome                char(40)        NOT NULL,
        Indirizzo           varchar(60)     DEFAULT 'sconosciuto',
        Telefono            varchar(40)     NOT NULL,
        PRIMARY KEY (Identificativo)
    )
```

Vi sono poi altri vincoli che è possibile imporre: si tratta dei **VINCOLI ESTERNI** che riguardano le relazioni con altre tabelle e la validità dei riferimenti ad esse. La definizione di tali vincoli è complessa e, visto che viene utilizzata in casi molto particolari, vi invitiamo a consultare lo stesso manuale del sistema RDBMS che utilizzate per maggiori informazioni e sintassi specifiche. A tale scopo, consultate le parti relative alle opzioni **FOREIGN KEY** e **REFERENCES**. Tali opzioni danno la possibilità di cambiare i valori di un campo di una tabella conseguentemente al cambiamento di quelli di un'altra, ecc..

MODIFICARE LA STRUTTURA DI UNA TABELLA

Se si desidera aggiungere nove colonne ad una tabella, si dovrà utilizzare la seeguente sintassi:

ALTER TABLE <*nome-tabella*> (
ADD COLUMN <*nome-colonna*> <*tipo*> [<*altre caratteristiche*>]
)

Se invece si desidera cancellare un campo (colonna) si dovrà utilizzare la sintassi:

ALTER TABLE <*nome-tabella*> (
DROP COLUMN <*nome-colonna*>
)

Con ADD, quindi, si aggiunge una colonna, mentre con DROP la si elimina. Si possono anche specificare dei vincoli, ovvero si può anche eliminare o attribuire un valore predefinito, come si evince dalla seguente sintassi:

ALTER TABLE <*nome-tabella*> (
ALTER COLUMN <*nome-colonna*> DROP DEFAULT
)

ALTER TABLE <*nome-tabella*> (
ALTER COLUMN <*nome-colonna*> SET DEFAULT <*valore-predefinito*>
)

ELIMINARE DI UNA TABELLA

Il comando DROP consente anche di eliminare l'intera tabella ed il suo utilizzo è molto semplice:

DROP TABLE <*nome-tabella*>

ATTENZIONE, però: l'eliminazione di una tabella dovrebbe essere consentita solo all'utente che l'ha creata!

INSERIRE, ELIMINARE E MODIFICARE I DATI

Queste operazioni possone essere effettuate sempre a livello di record (riga). Vediamo nel seguito le relative sintassi.

INSERIRE UNA RIGA DI VALORI (RECORD)

Per inserire una nuova riga in una tabella, occorre utilizzare l'istruzione INSERT.
Occorre indicare IN MODO ORDINATO i valori da inserire che andranno, così, nell'ordine, nelle colonne (campi) corrispondenti. La sintassi è:

INSERT INTO <*nome-tabella*> VALUES (<*espressione-1*>[,...<*espressione-N*>])

Esempio:

```
INSERT INTO Anagrafica
        VALUES (
                01,
                'Pippo',
                'De Pippis',
                'Via Topolinia 13',
                '0111,111111'
        )
```

Per non rischiare di dimenticare il valore da inserire in qualche colonna (cui verrebbe attribuito il valore di default o il nULL, sempre se il valore è in coda a quelli inseriti), conviene elencare anche il nome dei campi cui attribuire il valore, anche perché così non si dipende più dall'ordine dei campi stessi:

INSERT INTO <*nome-tabella*> (<*colonna-1*>[,...<*colonna-N*>])]
VALUES (<*espressione-1*>[,...<*espressione-N*>]);

Ed ecco l'esempio:.

```
INSERT INTO Anagrafica (
                Identificativo,
                Cognome,
                Nome,
                Indirizzo,
                Telefono
        )
        VALUES (
                01,
                'Pippo',
                'De Pippis',
                'Via Topolinia 13',
                '0111,111111'
        )
```

AGGIORNARE UNA RIGA

La modifica di una riga avviene scandendo la tabella, dalla prima riga all'ultima e verificando il verificarsi delle condizioni che scatenano la modifica stessa. La sintassi semplice è:

UPDATE <*tabella*>
SET
<*colonna-1*>=<*espressione-1*>[,...<*colonna-N*>=<*espressione-N*>]
[WHERE <*condizione*>]

UPDATE esegue tutte le sostituzioni indicate da <*colonna*>=<*espressione*>, nelle righe in cui la condizione posta dopo la parola chiave WHERE si avvera. ATTENZIONE: se non mettete la condizione, la modifica viene effettuata su TUTTE le righe della tabella!
Esempio:

```
UPDATE Anagrafica
        SET Nome='Pico'
        WHERE Cognome = 'De Paperis' AND Indirizzo = 'Via Galileo Galilei 12'
```

ELIMINARE UNA RIGA

Per cancellare una riga bisogna utilizzare la seguente sintassi, in viene indicato il nome della tabella **dalla quale** (FROM) cancellare la riga e, come per l'update, la condizione che deve verificarsi per effettuare la cancellazione.

DELETE FROM <*tabella*> [WHERE <*condizione*>]

ATTENZIONE: se non indicate una condizione, cancellerete TUTTE le righe!

INTERROGAZIONI (SELECT)

Con “interrogazione di una tabella” si intende il recupero dei dati in essa contenuti. Si possono interrogare in modo combinato anche più tabelle. Per indicare un’interrogazione si utilizza generalmente il termine QUERY. Vediamo quali sono le sintassi per compiere queste operazioni.

INTERROGAZIONI SEMPLICI

La sintassi più semplice per interrogare UNA SOLA tabella è la seguente:

SELECT <*espress-col-1*>[,...<*espress-col-N*>]
from <*tabella*>
[where <*condizione*>]

Esempio:

SELECT Cognome, Nome FROM Anagrafica

Se si vogliono ottenere, nell’ordine, tutti i valori di tutti i campi di una tabella, invece, si utilizza la sintassi:

SELECT * FROM Anagrafica

Non abbiamo però ancora utilizzato la condizione WHERE e quindi, come risultato, si sono ottenuti tutti i valori contenuti nella tabella. Se introduciamo anche la parola AS, come nella sintassi che segue, possiamo anche mostrare il risultato attribuendo un altro nome ad esso (un alias):

SELECT <*specifica-della-colonna-1*>[,...<*specifica-della-colonna-N*>]
FROM <*tabella*> AS <*alias*>
[WHERE <*condizione*>]

Anche alla tabella si può assegnare un alias, nell’effettuare una select.

Se si vuole fare riferimento al nome di una colonna che, per qualche motivo, risulta ambiguo, si può aggiungere anteriormente il nome della tabella a cui appartiene, separandolo attraverso l'operatore punto (**.**):

SELECT Anagrafica.Cognome, Anagrafica.Nome FROM Anagrafica

Se poi al nome della tabella viene abbinato un alias, si può scrivere la stessa espressione indicando il nome dell'alias al posto di quello della tabella, come nell'esempio:

SELECT Anag.Cognome, Anag.Nome FROM Anagrafica AS Anag

INTERROGAZIONI DI PIÙ TABELLE

Se dopo la parola chiave **FROM** si indicano più tabelle (si può anche indicare più volte la stessa tabella), si effettua la query su più tabelle contemporaneamente. Ecco la sintassi:

SELECT <*specifica-della-colonna-1*>[,...<*specifica-della-colonna-N*>]
FROM <*specifica-della-tabella-1*>[,...<*specifica-della-tabella-N*>]
[WHERE <*condizione*>]

L'interrogazione simultanea di più tabelle si presta anche per elaborare più volte la stessa tabella. In questo caso è obbligatorio utilizzare gli alias, come è evidente nell’esempio:

```
SELECT Anag1.Cognome, Anag1.Nome
        FROM Anagrafica AS Anag1, Indirizzi AS Anag2
        WHERE
                Anag1.Cognome = Anag2.Cognome
        AND     Anag1.Nome <> Anag2.Nome
```

Si ottiene così l'elenco delle persone che hanno lo stesso cognome, ma nome diverso.

CONDIZIONI (WHERE)

La condizione di selezione delle righe può essere composta a piacimento, purché il risultato sia di logico e i dati a cui si fa riferimento provengano dalle tabelle di partenza. Si possono usare anche altri operatori di confronto, funzioni, e operatori booleani.

AGGREGAZIONI

Per ottenere risultati riepilogativi da una tabella, si usa l'aggregazione, ovvero alcune funzioni speciali che restituiscono un solo valore, e quindi concorrono a creare un'unica riga. Le funzioni di aggregazione sono: **COUNT()**, **SUM()**, **MAX()**, **MIN()**, **AVG()**.
Per esempio:
SELECT COUNT(*) FROM Prodotti WHERE ...
In questo caso, quello che si ottiene è solo il numero di righe della tabella **Prodotti** che soddisfano la condizione posta. L'asterisco posto come parametro della funzione **COUNT()** rappresenta l'elenco di tutti i nomi delle colonne della tabella **Prodotti**.
La sintassi della funzione **COUNT()** è:
COUNT(*)
COUNT([DISTINCT|ALL] <*lista-colonne*>)
Il DISTINCT permette di contare solo le righe che contengono effettivamente valori diversi.
Quando non si usa DISTINCT è ovviamente implicito ALL.
Le altre funzioni di aggregazione non prevedono l'asterisco, perché generano un risultato per ogni riga della selezione.

SUM([DISTINCT|ALL] <*espressione*>)	somma
MAX([DISTINCT|ALL] <*espressione*>)	valore massimo
MIN([DISTINCT|ALL] <*espressione*>)	valore minimo
AVG([DISTINCT|ALL] <*espressione*>)	media

Provate ad utilizzare questi comandi, che seguono le sintassi viste fino ad ora.

RAGGRUPPAMENTI

I raggruppamenti si ottengono quando alle sintassi viste fino ad ora si aggiunge la clausola **GROUP BY**.
SELECT <*specifica-della-colonna-1*>[,...<*specifica-della-colonna-N*>]
FROM <*specifica-della-tabella-1*>[,...<*specifica-della-tabella-N*>]
[WHERE <*condizione*>]
GROUP BY <*colonna-1*>[,...]
La sintassi fa sì che la tabella ottenuta dall'istruzione **SELECT...FROM** sia filtrata dalla condizione **WHERE**. Fatto ciò, la tabella risultato viene ordinata in modo da raggruppare le righe in cui i contenuti delle colonne poste dopo la clausola **GROUP BY** sono uguali. Su questi gruppi di righe vengono poi valutate le funzioni di aggregazione.
Se si vuole porre un ulteriore filtro sul risultato di un raggruppamento, si può utilizzare **HAVING**.
SELECT <*specifica-della-colonna-1*>[,...<*specifica-della-colonna-N*>]
FROM <*specifica-della-tabella-1*>[,...<*specifica-della-tabella-N*>]
[WHERE <*condizione*>]
GROUP BY <*colonna-1*>[,...]
HAVING <*condizione*>

TRASFERIRE DATI IN UN'ALTRA TABELLA

Se vi è necessità, si possono trasferire dati da una tabella all'altra, vediamo come.

CREARE UNA TABELLA DA ALTRE PREESISTENTI

Si utilizza l'istruzione select, con la sintassi:

SELECT <*specifica-della-colonna-1*>[,...<*specifica-della-colonna-N*>]
INTO TABLE <*tabella-da-generare*>
FROM <*specifica-della-tabella-1*>[,...<*specifica-della-tabella-N*>]
[WHERE <*condizione*>]

L'esempio crea la tabella **NuovaTab** fondendo la tabella **Anagrafica** e **Parenti**.

```
SELECT
        Parenti.Figlio,
        Parenti.Coniuge,
        Anagrafica.Cognome,
        Anagrafica.Nome
        INTO TABLE NuovaTab
        FROM Parenti, Anagrafica
        WHERE Parenti.Identificativo = Anagrafica.Codice;
```

INSERIMENTO DI DATI IN UNA TABELLA ESISTENTE

L'inserimento di dati in una tabella prelevando da altre, può essere effettuata attraverso l'istruzione **INSERT** sostituendo la clausola **VALUES** con una **SELECT**.

INSERT INTO <*nome-tabella*> [(<*colonna-1*>...<*colonna-N*>)]
SELECT <*espressione-1*>, ... <*espressione-N*>
FROM <*tabelle-di-origine*>
[WHERE <*condizione*>]

LE VISTE (VIEW)

Le viste (View) sono delle tabelle virtuali ottenute da tabelle o da altre viste. Si tratta di rendere una select sottoforma di una tabella "normale": la vista.

CREATE VIEW <*nome-vista*> [(<*colonna-1*>[,...<*colonna-N*>)]
AS <*richiesta*>

Dopo la parola chiave **AS** deve essere indicato ciò che compone un'istruzione **SELECT**.

Esempio:

```
CREATE VIEW Strada_Topolinia
        AS SELECT Identificativo, Nome, Cognome, Indirizzo
                FROM Anagrafica
                WHERE Indirizzo IS LIKE 'Via Topolinia'
```

CONTROLLO DEGLI ACCESSI AD UN DATABASE

Per amministrare al meglio una base di dati, è necessario che vi sia un DBA (*Data Base Administrator*) che in quanto amministratore ha sempre tutti i privilegi necessari a intervenire sul RDBMS. Per default il nome di questo speciale utente è, in SQL, **_SYSTEM**.

IL CREATORE

L'utente che crea una tabella, o un'altra risorsa, ne è il creatore. In quanto tale è l'unico utente che può modificarne la struttura e che può eliminarla (oltre al DBA). Solo lui può usare **DROP** e **ALTER**. Inoltre, il creatore di una tabella può anche concedere o revocare privilegi ad altri utenti (sempre relativamente alla tabella stessa)

TIPI DI PRIVILEGI (GRANT)

Vi sono vari tipi di privilegi, che consentono di effettuare le operazioni:

- **SELECT** -- lettura del valore di un oggetto, per esempio una riga da una tabella;
- **INSERT** - - inserimento di un nuovo oggetto, ad esempio l'inserimento di una riga in una tabella;
- **UPDATE** -- aggiornamento del valore di un oggetto, ad esempio la modifica del contenutodi una riga di una tabella;
- **DELETE** -- eliminazione di un oggetto, come per esempio la cancellazione di una riga da una tabella;
- **ALL PRIVILEGES** – quando un utente ha tutti i privilegi elencati precedentemente.

COME CONCEDERE PRIVILEGI

L’istruzione da utilizzare è **GRANT**, la cui sintassi è:

GRANT <*privilegi*>
ON <*risorsa*>[,...]
TO <*utenti*>
[WITH GRANT OPTION]

Per esempio:

GRANT SELECT ON Anagrafica TO Pippo

Pippo potrà così leggere i dati della tabella Anagrafica.
Perché Pippo e Pluto possano invece godere di tutti i privilegi, occorrerà l’istruzione:

GRANT ALL PRIVILEGES ON Anagrafica TO Pippo, Pluto

Se si utilizza l’'opzione **WITH GRANT OPTION,** gli utenti avranno anche la facoltà di concedere a loro volta tali privilegi ad altri utenti. Ad esempio:

GRANT SELECT ON Anagrafica TO Pippo WITH GRANT OPTION

COME REVOCARE PRIVILEGI

Si utilizza l’istruzione **REVOKE**, di cui riportiamo la sintassi e, di seguito, l’esempio che revoca tutti i privilegi su Anagrafica a Pippo e Pluto.

REVOKE <*privilegi*>
ON <*risorsa*>[,...]
FROM <*utenti*>

Esempio: **REVOKE ALL PRIVILEGES ON Anagrafica FROM Pippo, Pluto.**

CONCLUSIONI

Ovviamente vi sono molti modi di combinare tutte le istruzioni che abbiamo visto.
Se siete interessati alla creazione di basi di dati, vi consigliamo di consultare il manuale “modellazione di basi di dati” contenuto nella sezione “Database”.
Se avete già letto quel manuale, potrete approfondire le vostre conoscenze leggendo il manuale “Le stored procedures” che troverete sempre nella sezione “Database”.
Buon lavoro!