IL LINGUAGGIO SQL

DISPENSE DIDATTICHE

GENERALITA' SUL LINGUAGGIO SQL

Il linguaggio SQL (Structured Query Language) è un linguaggio divenuto lo standard per creare e manipolare basi di dati relazionali. Questo linguaggio assolve alle funzioni di:

- DDL (Data Definition Language) che prevede le istruzioni per definire la struttura della base di dati. Serve quindi, a creare tabelle, vincoli, ecc.

- DML (Data Manipulation Language) che prevede le istruzioni per manipolare i dati contenuti nelle tabelle. In particolare permette di inserire, modificare, cancellare record e di effettuare interrogazioni.

- DCL (Data Control Language) che prevede istruzioni per controllare il modo in cui le operazioni vengono eseguite. Consentono di gestire il controllo degli accessi da parte di più utenti mediante permessi e autorizzazioni.

Il linguaggio SQL può essere usato in due diverse modalità:

- Modalità STAND ALONE: in questo caso i comandi possono essere inviati direttamente al sistema operativo che li esegue dopo averli tradotti utilizzando l'interprete SQL.

- Modalità ENBEDDED: in questo caso i comandi vengono inseriti all'interno di un linguaggio ospite (ad esempio Visual Basic, java). In questo caso il programma che ospita le istruzioni SQL subirà un primo processo di precompilazione, seguito dalla compilazione vera e propria.

IDENTIFICATORI E TIPI DI DATI

SQL non è un linguaggio Case-sensitive, per cui le istruzioni possono essere scritte utilizzando indifferentemente caratteri maiuscoli o minuscoli. Tuttavia, per una maggiore leggibilità, utilizzeremo le seguenti convenzioni:

- parole chiave: tutte in maiuscolo
- nomi di tabelle e campi: con iniziali maiuscole
- nomi dei campi chiave primaria e chiave esterna: tutti in maiuscolo

Gli identificatori utilizzati per i nomi di tabelle e campi devono sottostare alle seguenti regole:

- avere una lunghezza massima di 18 caratteri
- iniziare con un carattere alfabetico
- non contenere vocali accentate e caratteri speciali; l'unico carattere speciale consentito è l'underscore "_"

Per riferirsi al campo di una tabella, si utilizza la seguente sintassi: NomeTabella.NomeCampo

Le costanti di tipo stringa devono essere racchiuse tra apici ('……') o doppi apici ("…..").

Nelle espressioni possono essere utilizzati i seguenti operatori:

- aritmetici: + - * /
- relazionali: = < > <= >= <>
- logici: AND OR NOT

I principali tipi di dati sono i seguenti:

TIPO	DESCRIZIONE	Range di variabilità
CHAR	Singolo carattere	Tutti i caratteri del codice ASCII
CHAR(n)	Stringa di caratteri di lunghezza n	N varia da 1 a 15000
BIT	Singolo bit; corrisponde al tipo booleano	0 ≡ Falso, 1 ≡ Vero
INT	Numero intero	Dipende dall'implementazione, solitamente 4 Byte
SMALLINT	Numero intero inferiore a INT	Dipende dall'implementazione, solitamente 2 Byte
REAL	Numero reale	Dipende dall'implementazione, solitamente 7 bit di mantissa
FLOAT	Numero reale	Dipende dall'implementazione, solitamente 15 bit di mantissa
DOUBLE PRECISION	Numero reale	Bit di mantissa doppi rispetto a FLOAT
DATE	Data nel formato "AAAA/MM/GG"	
TIME	Ora nel formato "hh:mm:ss:msms"	

ISTRUZIONI DDL

Le istruzioni DDL servono definire la struttura della base di dati. Le principali sono:

- Creazione di un database:

```
CREATE  DATABASE   NomeDatabase
```

Esempio: **CREATE DATABASE** DBAzienda

- Creazione di una tabella

```
CREATE  TABLE   NomeTabella
(NomeCampo1        Tipo        [Vincolo],
……..........       ……        ……

NomeCampoN         Tipo        [Vincolo],

[VincoloTabella1],
……..

[VincoloTabellaM] );
```

Osserviamo che l'istruzione CREATE TABLE specifica

- ✓ Il nome della tabella
- ✓ I nomi dei campi
- ✓ I tipi dei campi
- ✓ Eventuali vincoli sui campi: questi possono essere:
  - Un valore di **default** che serve a specificare il valore da assegnare a quel campo nel caso in cui tale valore non sia stato assegnato.
  - Se il valore è obbligatoriamente richiesto (**Not Null**)

- ✓ Eventuali vincoli di tabella: questi possono essere
  - Vincoli sui valori assunti da uno o più campi
  - Definizione della chiave primaria
  - Definizione di eventuali chiavi esterne con riferimento alle tabelle collegate;
  - Il comportamento da tenere nel caso di modifica dei valori delle chiavi con conseguente violazione dell'integrità referenziale.

Vediamo un esempio:

```
CREATE TABLE Dipendenti
(ID              INT          NOT NULL,
Nominativo       CHAR(50)     NOT NULL,
DataNascita      DATE         NOT NULL,
Sesso            BIT          NOT NULL,
DataAssunzione   DATE
Livello          INT          DEFAULT  5,
Stipendio        REAL
ID_REPARTO       INT          NOT NULL,
CHECK (Stipendio>0),
CHECK (Livello IN (1,2,3,4,5)),
CHECK (DataAssunzione > DataNascita),
CHECK (Stipendio BETWEEN 1500 AND 2000),
UNIQUE (Nominativo, DataNascita),
PRIMARY KEY (ID),
FOREIGN KEY (ID_REPARTO) REFERENCES Reparti (ID) ON DELETE SET NULL );
```

La clausola CHECK è utilizzata per esprimere vincoli sui valori dei campi; oltre agli operatori di confronto si possono utilizzare i seguenti operatori:

- **IN**: per specificare che il valore deve essere compreso in un insieme finito.
- **BETWEEN**: per specificare che un valore deve essere compreso tra un minimo e un massimo
- **LIKE**: per specificare che il valore del campo deve avere un certo formato; ad esempio: CHECK (CodiceArticolo LIKE “COD%”).

La parola chiave **UNIQUE** serve a specificare che i valori di una sequenza di campi devono essere diversi per ogni record.
Come già detto, nella definizione di una tabella è possibile specificare la chiave primaria ed eventuali chiavi esterne. In questo caso occorre indicare la tabella a cui la chiave esterna fa riferimento e le politiche da adottare in caso di violazione dell’integrità referenziale. In particolare, può accadere che il valore della chiave primaria a cui una chiave esterna fa riferimento venga:

- Modificato: **ON UPDATE**
- Eliminato: **ON DELETE**

I comportamenti che possono essere adottati sono i seguenti:

- **CASCADE**: i valori della chiave esterna vengono automaticamente aggiornati con il nuovo valore della chiave primaria;
- **SET NULL**: i valori delle chiavi esterne vengono impostati a NULL;
- **SET DEFAULT**: i valori delle chiavi esterne vengono impostati al valore di DEFAULT

- **NO ACTION**: Non viene fatto niente; questa è l'opzione di default che viene adottata nel caso in cui non sia stato specificato nulla.

Il DML prevede anche istruzioni per modificare la struttura di una tabella, aggiungendo o eliminando un campo:

```
ALTER  TABLE   NomeTabella
       ADD NomeCampo1          Tipo
       [BEFORE NomeColonna2];

ALTER  TABLE   NomeTabella
       DROP COLUMN NomeCampo;
```

E' possibile, inoltre, eliminare una tabella

```
DROP  TABLE   NomeTabella
```

in questo caso occorre specificare il comportamento da adottare nel caso vi siano tabelle collegate a quella che si vuole eliminare. Le possibili opzioni sono:

- **CASCADE**: cancella in cascata tutte le tabelle collegate
- **SET NULL**: imposta a NULL tutti i valori delle chiavi esterne collegate alla tabella da eliminare
- **RESTRICT**: non cancella la tabella se essa è collegata ad altre tabelle

Esempio

DROP TABLE Dipendenti **RESTRICT**;

ISTRUZIONI DML

Questo gruppo di istruzioni consente di inserire, modificare, cancellare record da una tabella ed interrogare il database:

INSERIMENTO DI UN RECORD

```
INSERT INTO NomeTabella(NomeCampo1, ….NomeCampoN)
VALUES ( Valore1,….ValoreN );
```

Esempio:

```
INSERT INTO Calciatori (Nominativo, Livello, Gol)
VALUES ("Ronaldo",1, 8);
```

MODIFICA DI UNO O PIU' RECORD

```
UPDATE  NomeTabella  SET
        Campo1= Espressione1,
        Campo2= Espressione2,
        ……
        CampoN = EspressioneN
[WHERE condizione];
```

Esempi:

```
UPDATE Calciatori SET
Gol = Gol +1 ;

UPDATE Calciatori SET
Gol = 10
WHERE Nominativo="Ronaldo";
```

CANCELLAZIONE DI UNO O PIU' RECORD

```
DELETE FROM NomeTabella
[WHERE Condizione];
```

Esempi: `DELETE FROM Calciatori;`

```
DELETE FROM Calciatori
WHERE Gol < 3;
```

INTERROGAZIONE DEL DATABASE

Il comando che si utilizza per interrogare il database è il SELECT. Si tratta di un comando dalla sintassi molto complessa, quindi ci limiteremo a fornire solo degli esempi.
Questo comando consente tra l'altro, di effettuare le principali operazioni che già conosciamo:

PROIEZIONE

```
SELECT Campo1,…CampoN
FROM NomeTabella;
```

Esempio:

```
SELECT Nominativo, Livello
FROM Calciatori;
```

RESTRIZIONE

```
SELECT *
FROM NomeTabella
WHERE condizione;
```

Esempio:

```
SELECT *
FROM Calciatori
WHERE Gol > 10;
```

INNER JOIN

```
SELECT *
FROM NomeTabella1 INNER JOIN NomeTabella2
ON condizione;
```

Esempio:

```
SELECT *
FROM Calciatori INNER JOIN Squadre
ON Calciatori.ID_SQUADRA= Squadra.ID;
```

Il comando SELECT consente di effettuare contemporaneamente una restrizione e/o una proiezione e/o un join. Vediamo alcuni esempi:

```
SELECT Nominativo, Livello
FROM Calciatori
WHERE Gol > 10;
```

```
SELECT Calciatori.Nominativo, Squadre.Nome
FROM Calciatori INNER JOIN Squadre ON Calciatori.ID_SQUADRA= Squadra.ID;

SELECT Calciatori.Nominativo, Calciatori.Gol, Squadre.Nome
FROM Calciatori INNER JOIN Squadre ON Calciatori.ID_SQUADRA= Squadra.ID
WHERE Calciatori.Gol > 10
```

Quando si effettua una interrogazione, L' SQL permette di rinominare una tabella o un campo, in modo che nella tabella risultato le colonne siano intestate con altri nomi; per fare ciò sui utilizza la clausola AS:

```
SELECT Calciatori.Nominativo AS Calciatore, Calciatori.Gol AS GolFatti, Squadre.Nome AS Squadra
FROM Calciatori INNER JOIN Squadre ON Calciatori.ID_SQUADRA= Squadra.ID
WHERE Calciatori.Gol > 10
```

Per utilizzare l'OUTER JOIN destro o sinistro, il comando è lo stesso ma al posto di INNER JOIN si scrive RIGHT JOIN o LEFT JOIN.
Per realizzare il SELF JOIN su una stessa tabella si utilizza la ridenominazione delle tabelle:

```
SELECT *
FROM Persone AS Tab1 INNER JOIN Persone AS Tab2 ON Tab1.ID_PADRE= Tab2.ID;
```

Per realizzare il CROSS JOIN, si utilizza la seguente sintassi:

```
SELECT *
FROM tabella1, tabella2
```

È possibile effettuare contemporaneamente anche il CROSS JOIN tra più di due tabelle

```
SELECT *
FROM tabella1, tabella2, ….,tabellaN
```

Il comando SELECT normalmente non elimina eventuali righe uguali dalla tabella risultato; se si vuole invece avere la tabella risultato senza ripetizioni occorre utilizzare la clausola DISTINCT:

```
SELECT DISTINCT *
FROM ……..
```

Nell' espressioni delle condizioni, semplici o composte, si possono utilizzare gli operatori logici AND, OR, NOT e gli operatori IN, BETWEEN, LIKE. Per quanto riguarda il valore NULL, occorre precisare che la sintassi corretta è la seguente:

WHERE Impiegati.DataAssunzione **IS NULL**

Oppure

WHERE Impiegati.DataAssunzione **IS NOT NULL**

Infine, per tradurre le oprazioni di UNIONE, INTERSEZIONE, DIFFERENZA, SQL utilizza gli operatori UNION, INTERSECT, EXCEPT, che si applicano ai risultati delle interrogazioni:

Esempi:

per determinare tutti gli attori o registi:

(**SELECT** Cognome, Nome **FROM** Registi) **UNION** (**SELECT** Cognome, Nome **FROM** Attori)

per determinare gli attori che sono anche registi:

(**SELECT** Cognome, Nome **FROM** Registi) **INTERSECT** (**SELECT** Cognome, Nome **FROM** Attori)

per determinare i registi che non sono anche attori:

(**SELECT** Cognome, Nome **FROM** Registi) **EXCEPT** (**SELECT** Cognome, Nome **FROM** Attori)

LE FUNZIONI DI AGGREGAZIONE

L' SQL possiede alcune funzioni predefinite, utili nei casi in cui occorre effettuare conteggi, somme ecc. sulla tabella risultato di una query. Tali funzioni si applicano a un campo di una tabella. Le principali funzioni sono:

- **COUNT**: conta il numero di elementi presenti nel campo specificato, diversi da NULL.
  Se invece di specificare il campo si usa l'asterisco, vengono contati tutti i record della tabella;

- **MIN, MAX**: restituiscono il valore minimo o massimo del campo specificato;

- **SUM**: restituisce la somma degli elementi specificati;

- **AVG**: restituisce la media aritmetica degli elementi del campo specificato.

Esempio:
Consideriamo la seguente tabella:

Dipendenti (ID, Nominativo, Livello, Stipendio)

Per calcolare il numero di dipendenti con stipendio maggiore di 2000:

SELECT COUNT (Stipendio)
FROM Dipendenti
WHERE Stipendio >2000;

Per calcolare lo stipendio medio di tutti i dipendenti

SELECT AVG (Stipendio)
FROM Dipendenti;

Per calcolare lo stipendio massimo di tutti i dipendenti del 5° livello

SELECT MAX (Stipendio)
FROM Dipendenti
WHERE Livello= 5;

Per calcolare il numero di record presenti nella tabella Dipendenti:

SELECT COUNT (*)
FROM Dipendenti;

ORDINAMENTI

In SQL è possibile ordinare le righe di una tabella risultato in base a certi criteri.
E' possibile specificare uno o più campi, in base a cui ordinare e per ogni campo si può specificare il tipo di ordinamento crescente (**ASC**) o decrescente (**DESC**). Per fare ciò basta utilizzare la clausola **ORDER BY**:

Esempio:

Per avere tutti i dipendenti ordinati per nome in ordine crescente

```
SELECT *
FROM Dipendenti
ORDER BY Nominativo ASC;
```

Per avere tutti i dipendenti del 5° livello ordinati per stipendio decrescente

```
SELECT *
FROM Dipendenti
WHERE Livello= 5
ORDER BY Stipendio DESC;
```

Per avere tutti i dipendenti ordinati per livello decrescente e poi per nominativo crescente:

```
SELECT *
FROM Dipendenti
ORDER BY Livello DESC, Nominativo;
```

RAGGRUPPAMENTI

L' SQL consente di raggruppare logicamente le righe della tabella risultato di una query, in base al valore assunto da uno o più campi.
In particolare tutte le righe aventi lo stesso valore (o gli stessi valori) di un determinato campo (campi) vengono fuse in un'unica riga.
Per effettuare un raggruppamento si usa la clausola **GROUP BY**:

Esempio:

La seguente query raggruppa tutti i dipendenti per livello e calcola la media dello stipendio per ciascun livello:

SELECT Livello, **AVG** (Stipendio)
FROM Dipendenti
GROUP BY Livello;

Livello	AVG(Stipendio)
1	1500
2	1400
3	1300
4	1200
5	1100

La seguente query raggruppa tutti i dipendenti per livello e calcola il numero di dipendenti per ciascun livello:

SELECT Livello, **COUNT** (Livello) **AS** NumDipendenti
FROM Dipendenti
GROUP BY Livello;

Livello	NumDipendenti
1	15
2	5
3	7
4	10
5	12

Si può aggiungere una condizione sul gruppo, utilizzando la clausola **HAVING**:

SELECT Livello, **AVG**(Stipendio) **AS** StipendioMedio
FROM Dipendenti
GROUP BY Livello
HAVING Livello >=4;

Livello	StipendioMedio
4	1200
5	1100

QUERY INTERMEDIE E QUERY ANNIDATE

In alcuni casi, la scrittura di una interrogazione può richiedere più passaggi: occorre, cioè, creare prima una o più tabelle intermedie e infine scrivere la query che produce il risultato finale.

Esempio1

Supponiamo di voler conoscere i nomi dei dipendenti aventi stipendio maggiore dello stipendio medio di tutti i dipendenti. Procediamo nel seguente modo:

- scriviamo una query intermedia che calcola lo stipendio medio e salva il risultato in una tabella Temp.

```
CREATE TABLE Temp
SELECT AVG (Stipendio)
FROM Dipendenti;
```

Osserviamo che la tabella Temp conterrà un singolo valore, cioè una sola riga ed un solo campo

AVG(Stipendio)
1300

- scriviamo la query principale che seleziona i nomi dei dipendenti aventi stipendio maggiore della media

```
SELECT Dipendenti.Nome
FROM Dipendenti
WHERE Dipendenti.Stipendio > Temp;
```

È possibile arrivare allo stesso risultato anche scrivendo una sola query che “annida” dentro di sé la query parziale:

```
SELECT Dipendenti.Nome
FROM Dipendenti
WHERE Dipendenti.Stipendio > (SELECT AVG (Stipendio) FROM Dipendenti);
```

Esempio2

Supponiamo di avere le seguenti tabelle relative ai noleggi di film presso una videoteca:

Clienti (ID, Nominativo)

Film (ID, Titolo)

Noleggi (ID, ID_CLIENTE, ID_FILM, Data)

E di voler conoscere il titolo del film che è stato noleggiato meno volte:

contiamo il numero di noleggi per ogni film e salviamo il risultato in una tabella Temp

```
CREATE TABLE Temp
SELECT Film.Titolo, COUNT(Film.Titolo) AS NumNoleggi
FROM Noleggi INNER JOIN Film ON Noleggi.ID_FILM=Film.ID
GROUP BY Film.Titolo;
```

Selezioniamo da Temp il titolo del film con meno noleggi

```
SELECT Titolo
FROM Temp
WHERE Temp.NumNoleggi = (SELECT MIN(NumNoleggi) FROM Temp);
```

Oppure, scriviamo un'unica query:

```
SELECT Titolo
FROM (SELECT Film.Titolo, COUNT(Film.Titolo) AS NumNoleggi
      FROM Noleggi INNER JOIN Film ON Noleggi.ID_FILM=Film.ID
      GROUP BY Film.Titolo) AS Temp
WHERE Temp.NumNoleggi = (SELECT MIN(NumNoleggi) FROM Temp);
```

QUERY PARAMETRICHE

Per generalizzare una query, è possibile utilizzare uno o più parametri: essi possono essere visti come "dati di input" della query. Ad esempio, supponiamo di voler conoscere I nominativi di tutti I dipendenti di un certo livello x e con stipendio maggiore di un valore y, scriveremo:

```
PARAMETERS LivX: intero, StipY: reale
SELECT Nominativo
FROM Dipendenti
WHERE Livello=[LivX]  AND Stipendio > [StipY];
```

LE VISTE

In SQL è possibile definire un altro tipo di tabella chiamata VISTA.
A differenza di una normale tabella, una vista non è fisicamente memorizzata nel database ma può essere definita solo logicamente.
Per creare una VISTA si usa la seguente sintassi:

CREATE VIEW NomeVista AS Query

Una vista quindi non è altro che il risultato di una query a cui viene assegnato un nome.
Le viste vengono utilizzate per fornire una specie di "interfaccia controllata" tra il database e l'utente nel caso in cui si vogliono rendere visibili a determinati utenti solo determinati dati.

Esempio:Supponiamo di avere la seguente tabella

Prodotti (ID, Descrizione, Prezzo, Quantita)

Supponiamo che la precedente tabella faccia parte di un database di un'azienda.
In tale azienda ci sono diversi impiegati che svolgono determinate mansioni: il dirigente, l'impiegato d'ufficio, il magazziniere ecc..
Ognuno di essi nell'ambito del suo lavoro ha accesso al database.
Ciascuno però è interessato solo ad una parte di tutti i dati memorizzati nel database: quindi può essere utile definire diverse viste sul database destinate a ciascuna categoria.
Ad esempio una vista utile al magazziniere è quella che gli consente di visualizzare i prodotti che devono essere ordinati poiché sono terminati:

```
CREATE VIEW DaOrdinare AS
SELECT * FROM Prodotti WHERE Quantita = 0;
```

ISTRUZIONI DCL

Le istruzioni DCL permettono di impostare le politiche di sicurezza riguardo all'accesso ai dati da parte degli utenti del database.
Per concedere ad un utente o gruppo di utenti i diritti di accesso ad una tabella (o ad una vista), si utilizza il comando **GRANT**, la cui sintassi è:

GRANT ElencoPermessi ON NomeTabella TO (ElencoUtenti);

L'elenco dei permessi contiene l'insieme delle azioni che sono consentite sulla tabella e sui campi della tabella: SELECT, INSERT, UPDATE, DELETE.

Esempio:
Il seguente comando consente agli utenti Rossi e Bianchi di poter interrogare la tabella *Dipendenti* e di poter modificare il campo *Stipendio*

GRANT SELECT, UPDATE (Stipendio) **ON** Dipendenti **TO** (Rossi, Bianchi);

Analogamente si possono assegnare i permessi su una vista: in questo caso l'unico permesso che abbia senso è l'interrogazione:

GRANT SELECT ON DaOrdinare **TO** Neri;