Laboratorio:
Basi Di Dati

Enrico Martini

2019

Structured Query Language (SQL)

Structured Query Language (SQL) il linguaggio pi diffuso per i RDBMS. SQL stato definito negli anni 70 ed stato standardizzato negli anni 80 e 90 da ISO e IEC. SQL diviso in diverse parti:

1. **DDL** Data Definition Language
2. **DML** Data Manipulation Language
3. **DQL** Data Query Language
4. **DCL** Data Control Language

1 DDL

1.1 Creazione tabella

Sintassi:

```
CREATE [TEMP] TABLE nome_tabella(
    nome_attributo dominio [vincoli],
    nome_attributo dominio [vincoli],
    [...]
    vincolo_tabella
);
```

1.2 Lista dei domini

Caratteri:

- `CHAR`: carattere singolo
- `CHAR(N)`: stringa di lunghezza fissa N
- `VARCHAR(N)`: stringa di lunghezza variabile (max N)
- `TEXT`: stringa di lunghezza variabile illimitata

Booleani:

- `BOOL`: rappresentano valori booleani T/F, 1/0, YES/NO, TRUE/FALSE e NULL

Numerici esatti:

- `SMALLINT`: valori interi (simile a `short`)
- `INTEGER`: valori interi (simile a `int`)
- `NUMERIC(N,M)`: valori decimali, N il numero di cifre totali, M il numero di cifre dopo la virgola
- `DECIMAL(N,M)`: valori decimali, N il numero di cifre totali, M il numero di cifre dopo la virgola

Numerici approssimati:

- `REAL`: valori con precisone a 6 cifre decimali
- `DOUBLE PRECISION`: valori con precisione a 15 cifre decimali

Tempo:

- `DATE`: date nel formato ISO `'YYYY:MM:DD'`
- `TIME [(P)] [WITH TIME ZONE]`: tempo orario nel formato ISO `'hh:mm:ss'`
  - `P`: precisione delle frazioni di secondo
  - `WITH TIME ZONE`: fuso orario nel formato ISO `'hh:mm:ss:± hh'`
- `TIMESTAMP [WITH TIME ZONE]`: unione di date e time
- `INTERVAL[FIELDS][P]`: per rappresentare la durata, in formato ISO `'P1Y2M3DT4H6M9S'`
  - `FIELDS`: YEAR, MONTH, DAY, YEAR TO MONTH, DAY TO HOUR, DAY TO SECOND, MINUTE TO SECOND
  - `P`: precisione delle frazioni di secondo (se `FIELDS` comprende i secondi)

Domini definiti dall'utente:

```
CREATE DOMAIN nome_dominio AS tipo_base [valore_default]
                                        [vincolo];
```

1.3 Vincoli

Specificano proprieta' che devono essere soddifsfatte da ogni tupla.

1.3.1 Vincoli NOT NULL e DEFAULT

- `NOT NULL`: non ammesso il valore nullo
- `DEFAULT valore`: quando non specificato attribuisce `valore` all'attributo

1.3.2 Vincolo UNIQUE

- `UNIQUE(attributo/i)`: i valori dell'attributo sono superchiave

1.3.3 Vincolo PRIMARY KEY

- `PRIMARY KEY`: equivalente a `UNIQUE NOT NULL`, chiave primaria della relazione

Pu essere specificata sia come vincolo di attributo, se la chiave primaria composta da un solo attributo, sia alla fine della definizione, se la chiave composta da pi attributi.

1.3.4 Vincolo CHECK

- `CHECK(espressione)`: soddisfatto se la sua espressione vera o nulla

E' conveniente mettere sempre `NOT NULL` insieme al vincolo `CHECK`.

1.3.5 Vincolo FOREIGN KEY

Crea un legame tra una tabella master A e una tabella slave B. In ogni tupla di B l'attributo scelto (se diverso da `NULL`) deve essere presente nella tabella A. L'attributo scelto della tabella A deve essere `PRIMARY KEY` o `UNIQUE`.
Quando il vincolo coinvolge un solo attributo:

```
1    ...
2    attributo REFERENCES tabella_esterna(attributo_estero)
3    ...
```

Quando il vincolo su piu' attributi:

```
1    ...
2    FOREIGN KEY (attributi) REFERENCES tabella_esterna(attributi)
3    ...
```

1.4 Modificare la struttura di una tabella

Aggiunta attributo:

```
1    ALTER TABLE tabella ADD COLUMN attributo tipo;
```

Rimozione attributo:

```
1    ALTER TABLE tabella DROP COLUMN attributo;
```

Modifica valore di default:

```
1    ALTER TABLE tabella ALTER COLUMN attributo {SET DEFAULT valore | DROP
     DEFAULT};
```

Aggiunta vincolo di integrit referenziale:

```
1    ALTER TABLE tabella ADD CONSTRAINT def_vincolo;
```

Rimozione vincolo di integrit referenziale:

```
1    ALTER TABLE tabella DROP CONSTRAINT nome_vincolo;
```

1.5 Inserimento dati in una tabella

Sintassi:

```
1    INSERT INTO tabella [elenco attributi]
2        VALUES(elenco valori);
```

1.6 Aggiornamento dati in una tabella

Sintassi:

```
1        UPDATE tabella SET attributo = espressione [WHERE condizione];
```

La clausola `WHERE` un espressione per selezionare le righe da modificare.

1.7 Cancellazione dati in una tabella

Cancellare righe:

```
1        DELETE FROM tabella [WHERE condizione];
```

Cancellare intera tabella:

```
1        DROP TABLE tabella;.
```

2 DQL

In SQL esiste solo il comando `SELECT` per interrogare una base di dati.
Sintassi semplificata:

```
SELECT [DISTINCT] attributi
FROM    tabelle
WHERE   condizioni;
```

2.1 Comando WHERE

2.1.1 Operatore LIKE

- `attributo LIKE 'text'`: confronto tra stringhe case sensitive
- `attributo iLIKE 'text'`: confronto tra stringhe non case sensitive

Si possono utilizzare due caratteri speciali per il matching: _ indica che pu esserci un carattere qualsiasi, `%` indica che possono esserci 0 o pi caratteri qualsiasi.

2.1.2 Operatore SIMILAR TO

Sintassi:

```
attributo SIMILAR TO espressione_regolare
```

Accetta come pattern un sottoinsieme delle espressioni regolari POSIX:

- _: 1 carattere qualsiasi
- `%`: 0 o piu' caratteri qualsiasi
- `*`: ripetizione del match 0 o pi volte
- `+`: ripetizione del match 1 o pi volte
- `{n,m}`: ripetizione del precedente match almeno n e massimo m volte
- `[...]`: elenco di caratteri ammissibili

2.1.3 Operatore BETWEEN

Sintassi:

```
WHERE attributo [NOT] BETWEEN minimo AND massimo
```

2.2 Operatore IN

Sintassi:

```
WHERE attributo [NOT] IN (valore[,...]);
```

2.3 Operatore IS

Fondamentale perch l'unico modo di sapere se un attributo `NULL`. Sintassi:

```
WHERE attributo IS [NOT] NULL;
```

3 Comando SELECT

3.1 Operatore ORDER BY

Ordina le tuple risultato in ordine rispetto agli attributi specificati. Sintassi:

```
ORDER BY attributo [ASC | DESC] [, ...];
```

3.2 Operatori di aggregazione

Permettono di determinare un valore considerando i valori ottenuti da una SELECT. Se si usano operatori aggregati **NON** si devono inserire in SELECT altri attributi di tuple. Non possibile usare due operatori di aggregazione in cascata!

3.2.1 Operatore COUNT

Restituisce il numero di tuple significative nel risultato dell'interrogazione. Sintassi:

```
SELECT COUNT ( * | [ALL,DISTINCT] espressione )
```

3.2.2 Operatori SUM / MAX / MIN / AVG

Determinano un valore numerico o alfanumerico considerando le tuple significative. Sintassi:

```
SELECT op ( [DISTINCT] espressione )
```

3.3 Operatori di raggruppamento

Un raggruppamento un insieme di tuple che hanno stessi valori su uno o pi attributi.

3.3.1 Clausola GROUP BY

Permette di determinare tutti i raggruppamenti delle tuple in funzione degli attributi. Sintassi:

```
GROUP BY attributo [,...];
```

3.3.2 Clausola HAVING

Mentre la clausola WHERE permette di selezionare le righe che devono far parte del risultato, la clausola HAVING permette di selezionare i raggruppamenti che devono far parte del risultato. L'espressione booleana deve usare attributi del GROUP BY o operatori aggregati. Sintassi:

```
HAVING espressioneBooleana;
```

3.4 Interrogazioni nidificate

Permette il confronto di un valore con il risultato dellesecuzione di una interrogazione SQL. Gli operatori di confronto tradizionali (¡,¿,¡¿,=,...) NON non possono essere usati. Lo schema deve essere uguale tra la query nidificata e la clausola del `WHERE`. Sintassi:

```
WHERE espressione OP (subquery)
```

Nuovi operatori da utilizzare:

1. `[NOT] EXISTS`: ritorna falso se (subquery) non contiene righe. Si utilizzano anche i valori contenuti nella `SELECT` principale(*data binding*). **NON EFFICIENTE!**
2. `[NOT] IN`: la (subquery) deve restituire un numero di colonne pari al numero di espressioni. I valori dellespressioni vengono confrontati con i valori di ciascuna riga del risultato. ritorna vero se i valori sono uguali ai valori di almeno una riga della subquery.
3. `ANY/SOME`: (subquery) una SELECT che deve restituire UNA sola colonna. operator un operatore di confronto, come = o >= . Ritorna vero se expression operator rispetto al valore di una qualsiasi riga del risultato di (subquery) .
4. `ALL`: (subquery) una SELECT che deve restituire UNA sola colonna. operator un operatore di confronto, come = o >= . Ritorna vero se expression operator rispetto al valore di ciascuna riga del risultato di (subquery) .

4 Operatori di tipo insiemistico

Sintassi:

```
    query { UNION | INTERSECT | EXCEPT } [ ALL ] query
```

Gli operatori si possono applicare solo quando le due query producono risultati con lo stesso numero di colonne e di tipo compatibile fra loro.

5 Viste

Le viste sono tabelle virtuali il cui contenuto dipende dal contenuto delle altre tabelle della base di dati. Sintassi:

```
CREATE [ TEMP ] VIEW nome [( col_name [ , ...]) ] AS query
```

5.1 Comando FROM

5.1.1 Operatore JOIN

- `CROSS JOIN`: prodotto cartesiano
- `INNER JOIN`: join classico
- `LEFT OUTER JOIN`: mantiene anche le tuple della tabella di sinistra che non matchano
- `RIGHT OUTER JOIN`: mantiene anche le tuple della tabella di destra che non matchano
- `FULL OUTER JOIN`: mantiene anche le tuple che non matchano

Sintassi:

```
FROM tabella [AS] nome JOIN tabella ON condizione
```

6 Indici

Gli indici sono delle strutture dati ausiliare che permettono di accedere ai dati di una tabella in maniera pi efficiente. Il costo di aggiornamento pu essere significativo quando ci sono molti indici definiti sulla medesima tabella. Sintassi:

```
CREATE INDEX [ nome ] ON nomeTabella [ USING method ]
    ({ nomeAttr |( expression ) } [ ASC | DESC ] [ , ...])
```

Per modificare o eliminare un indice:

```
ALTER INDEX [IF EXISTS] indice;
DROP INDEX  [IF EXISTS] indice;
```

Per applicare o aggiornare un indice/indici:

```
ANALYZE [nome_tabella];
```

Il DBMS crea di default un indice sulla chiave primaria. Un indice pu velocizzare anche i comandi `UPDATE / DELETE` quando nella clausola `WHERE` ci sono attributi indicizzati.

6.0.1 Metodi

PostgreSQL ammette diversi tipi di indici: `B-tree`, `hash`, `GiST`, `SP-GiST`, `GIN` e `BRIN`. Gli indici dichiarati su pi attributi potrebbero avere efficienza maggiore se nelle query vengono utilizzati entrambi, a meno che non siano usati in `OR`.

- `B-tree`: ogni volta che lattributo indicizzato coinvolto in un confronto usando uno degli operatori: $<, <=, =, >=, >$ e pu essere considerato anche quando ci sono `BETWEEN`, `IN`, `IS NULL`, `IS NOT NULL` e `LIKE` .
- `hash`: solo quando ci sono confronti di uguaglianza. PostgreSQL ne sconsiglia l'uso.
- `GiST`: quando si ha un attributo di tipo bidimensionale/geometrico.

Se non si specifica il tipo di indice nel `CREATE INDEX` , il sistema crea un indice di tipo `B-tree`. Gli indici non vanno usati quando ci sono modifiche sull'attributo sul quale sono dichiarati (esempio: `LOWER(stringa)`). A quel punto conviene creare l'indice sull'espressione:

```
CREATE INDEX ins_lower_nonins ON
    Insegn (LOWER(nomeins) varchar_pattern_ops);
```

7 Explain

Il comando `EXPLAIN query` permette di vedere il piano di esecuzione di una query che l'ottimizzatore determina senza eseguire la query. Un piano di esecuzione di una query un albero di nodi di esecuzione. Le foglie sono nodi di scansione: l'esecuzione di questi nodi restituiscono indirizzi di righe della tabella. Esistono differenti tipi di

scansioni: sequenziali, indicizzate e mappate su bit. Se una query contiene `JOIN` , `GROUP BY` o `ORDER BY` o altre operazioni sulle righe, allora ci saranno altri nodi di esecuzione sopra le foglie nell'albero. L'output di `EXPLAIN` ha una riga per ciascun nodo nell'albero, dove viene indicato il tipo di operazione e una stima del costo di esecuzione. La prima riga dell'output ha la stima del costo totale di esecuzione.
Operazioni che potrebbero essere in `EXPLAIN`:

- `Bitmap Head Scan`: il nodo padre analizza il B+ tree controllando la condizione
- `BitmapAnd`: il nodo padre delle foglie fa l'intersezione tra due B+ tree
- `BitmapOr`: esegue l'unione tra due B+ tree
- `Hash`: prepara un hashtable per una possibile scansione sequenziale
- `Hash Join`: cerca per ogni riga del primo figlio la riga della hashtable da unire
- `Nested Loop`: unisce ogni riga prodotta dal nodo con ogni riga prodotta da un altro nodo. il valore `loops` indica quante volte viene ripetuto il nodo
- `Merge Join`: esegue il join ordinando prima le due tabelle

`EXPLAIN ANALYZE` mostra il piano di esecuzione, esegue la query senza registrare eventuali modifiche e mostra, infine, una stima verosimile dei tempi di esecuzione.

8 Transazioni

Sequenza di istruzioni SQL che vengono eseguite come fossero ununica istruzione. Se una transazione termina senza errori, tutte le modifiche eseguite dalle istruzioni interne diventano visibili, senn vengono tutte annullate. Una transazione inizia con `BEGIN` e termina con `COMMIT` per confermare o con `ROLLBACK` per annullare. Si parla di **concorrenza** quando due o pi transazioni accedono agli stessi dati nello stesso momento e il DBMS deve garantire che i dati siano sempre in uno stato consistente.In PostrgreSQL ciascuna transazione vede un'istantanea della base di dati. Una scrittura sospesa quando, in parallelo, un'altra transazione (non ancora chiusa) ha modificato la sorgente dei dati che si vuole aggiornare. Al `COMMIT` il sistema registra le modifiche.Il cambio di livello di isolamento si effettua con `SET TRANSACTION` appena dopo il `BEGIN` della transazione.
Ci sono 3 livelli di isolamento:

1. `Serializable`: ordine sequenziale rispetto ad altre transazioni. Completo isolamento. Il codice di errore in caso di fallimento `SQLSTATE = 40001`.
2. `Repeatable read`: rifacendo la lettura dei medesimi dati, si ottengono sempre gli stessi. Possono verificarsi casi di **phantom read**.
3. `Read committent`(default): garantisce che qualsiasi `SELECT` di una transazione vede solo i dati confermati (`COMMITTED`) prima che la `SELECT` inizi. Possono verificarsi casi di **non-repeatable read** o **phantom read**.

Sintassi:

```
SET TRANSACTION ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ
                COMMITTED | READ UNCOMMITTED } | READ WRITE | READ ONLY
```

Un metodo pratico per decidere se necessario il livello serializable : verificare se la selezione delle righe di un UPDATE/INSERT/DELETE in una transizione pu essere modificata in una transizione concorrente (che pu aggiungere/togliere o modificare le righe di interesse).

9 Python & Database

9.1 Connection

Viene utilizzata l'implelentazione di DB-API v2.0 `psycopg2`. Accesso al database:

```
1 connector= psycopg2.connect(host="dbserver.scienze.univr.it", \
2 database="db0", user="user0", password="xxx" )
```

Metodi della classe Connection (oggetto connector):

- `cursor()`: invia comandi SQL e accede al risultato del comando
- `commit()`: registra la transazione. Se non si esegue un commit tutte le modifiche andranno perse
- `rollback()`: abortisce la transazione
- `close()`: chiude la connessione ed effettua un rollback automatico
- `autocommit`: se a True non si deve mettere il comando `commit()`
- `readonly`: se a True non si possono modificare i dati.
- `isolation_level`: modifica il livello di isolamento con: `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` , `SERIALIZABLE`, `DEFAULT`.

9.2 Cursor

Lista dei metodi:

- `execute(comando, parametri)`: esegue un comando SQL usando i parametri che devono essere passati come tupla o dictionary formattando il comando con %s.
- `executemany(comando, parametri)`: esegue un comando SQL per ciascun valore presente nella lista parametri. Accetta una lista di tuple.
- `fetchone()`: ritorna una tupla della tabella risultato.
- `fetchmany(<n>)`: ritorna una lista di n tuple.
- `rowcount`: conta il numero di righe prodotte dall'ultimo comando.
- `statusmessage`: ultimo messaggio ritornato

9.3 Schema tipico di comunicazione DBMS

1. Apro la connessione (`conn = psycopg2.connect(...)`)
2. Eventuale modifica proprieta'
3. Instazio il cursore (`cur = conn.cursor()`)
4. Eseguo le operazioni
5. Se la sessione non ha autocommit eseguo un `conn.commit()`
6. Chiudo il cursore `cur.close()`
7. Chiudo la connessione `conn.close()`

10 Java & Database

Java DataBase Connectivity (JDBC) la Application Program Interface (API) ufficiale.

10.1 Schema tipico di comunicazione con DBMS

1. Carico il driver `Class.forName(...)`
2. Creare l'oggetto Connection `DriverManager.getConnection(<uri>,<user>,<pw>)`

10.2 Metodi della classe Connection

- `createStatement()`: crea oggetto `Statement` per inviare query statiche (senza parametri).
- `prepareStatement()`: rappresenta la query per poterla inviare pi volte con parametri diversi.
- `commit()` registra la transazione. Normalmente sono di default in autocommit.
- `rollback()`: abortisce la transazione.
- `close()`: chiude la connessione.

10.3 Metodi della classe PreparedStatement

- `executeUpdate()`: esegue l'update.
- `executeQuery()`: esegue la query

Esempio di `PrepareStatement`:

```
Prep aredSt ateme nt pst = con . prepareStatement ( " INSERT INTO Spese
( data , voce , importo ) VALUES (? ,? ,?) , (? ,? ,?) , (? ,? ,?) ,
(? ,? ,?) " )

pst.setDate (1 , new Date (2016 ,02 ,24) ) ;
pst.setString (2 , " Stipendio1 " ) ;
pst.setBigDecimal (3 , new BigDecimal ( " 0.1 " ) ) ;
```

Ritorna un `ResultSet`, i quali metodi sono:

- `next()`: posiziona il cursore alla prossima riga non letta.
- `get<tipo>(<index>)`: recupera il valore della colonna ¡index¿ o ¡nome¿ della riga corrente.

11 HTML

Usato per la formattazione e impaginazione di documenti ipertestuali. Scopo principale del linguaggio gestire i contenuti (sia strutturalmente, sia graficamente) all'interno della pagina web mediante luso di marcatori (tag) ,la sua estensione tipicamente `.html` .
Struttura generale:

```
<!DOCTYPE html>
<html>
    <head>
    ...
    </head>
    <body>
    ...
    </body>
</html>
```

11.1 Tag principali

- `<link>`: definisce un collegamento tra un documento e una risorsa esterna
- `<style>`: definisce una proprieta' CSS
- `<article>`: definisce un elemento indipendente
- `<section>`: definisce una sezione
- `<p>`: definisce un paragrafo
- `<ul>` : elenco non ordinato (con `<li>`)
- `<ol>` : elenco ordinato (con `<li>`)
- `<table>` : crea una tabella
- `<form>` : definisce da dove prendere gli input
- `<input>` : definisce il listener (es. `<input type="text">`, `<input type="button">`, `<input type="password">` ...)
- `<br>` : vai a capo
- `<a>` : definisce un *hyperlink* tra una pagina web e l'altra
- `<span>` : inserisce elementi inline

12 CSS

E´ un linguaggio per definire la formattazione di documenti HTML, XML ... Permete di definire gli aspetti visivi/sonori di ciascun tipo di tag, di classi di tag o di tag specifici. Un tag puo´ ereditare specifiche. Un foglio di stile si associa ad un documento `html` usando il tag `<style>` o il tag `<link>` nello `<head>`.

12.1 Struttura di un foglio CSS

```
selettore[,...]{
    proprieta' : valore[;...]
}
```

- `selettore`: un tag o identificatore di classe (inizia con "."), o un identificatore di un tag specifico (inizia con "#").
- `proprieta'`: il nome di un aspetto grafico da controllare.
- `valore`: valore da assegnare

13 Flask

E´ un micro framework per web application in Python.

```
1  from flask import Flask
2
3  app = Flask (__name__) # app    un' applicazione web
4
5  #utilizzo del pattern decorator
6  @app.route('/') # helloWorld    associato all' url '/'
7  def helloWorld() :
8  return 'Hello World! ' #    testo , no html !
9
10 @app.route ("/it") # ciaoMondo    associato all' url '/it'
11 def ciaoMondo () :
12 return '''<html> <head> Ciao </head > </html>
13  '''
14
15 if __name__ == '__main__': # se il modulo    invocato direttamente
16 app.run (debug = True) # attiva il web server con questa app
```

- L'istruzione `app = Flask(__name__)` crea l'applicazione (vuota) e chiede sempre un nome. `__name__` una variabile definita dallinterprete. il nome del modulo se il file importato, o la stringa `__main__` se il file direttamente eseguito.
- Il *decorator* `route(roule, options)` della classe Flask modifica lapplicazione in modo all'URL `rule` venga associato il metodo subito sotto la specifica del decorator. `Options` una lista di parametri opzionali che permettono di specificare altre condizioni per lassociazione.
- Il metodo `run(host, port, debug, options)` di Flask esegue lapplicazione nel web server locale. I parametri sono opzionali. I valori di default sono: `host = 127.0.0.1` (`0.0.0.0` permette di far raggiungere il server da Internet), `port=5000`, `debug=false`, `options=None`.

13.1 Webapp in Python basata su Flask

14 Altre skills

14.1 Elaborazione di stringhe

- `string || string`: concatena due espressioni
- `CHAR_LENGTH(attributo)`: lunghezza della stringa `attributo`
- `UPPER()`: torna la stringa in maiuscolo
- `LOWER()`: torna la stringa in minuscolo
- `varchar_pattern_ops`: parola chiave per **indici** su stringhe per utilizzare pattern come `LIKE`.

14.2 Casting

Ci sono due modi per fare casting in SQL:

1. `CAST(espressione AS TIPO)`
2. `espressione::TIPO`

14.3 PostreSQL

- Il comando `\timing` attiva/disattiva la visualizzazione del tempo di pianificazione + calcolo di una query in ms.
- Il comando `\di` visualizza l'elenco di indici sulla base di dati.
- Il comando `\d nome_tabella` visualizza la definizione di tabella e gli indici associati ad essa.