Capitolo 1

SQL - PostgreSQL

Il linguaggio SQL si compone di tre parti:

- Data Definition Language (DDL), utilizzato per definire le strutture dati e la definizione dei vincoli di integrità;
- Data Manipulation Language (DML), utilizzato per inserire, aggiornare e cancellare dati. Contiene anche il Data Query Language, utilizzato per le interrogazioni;
- Data Control Languade, utilizzato per modificare la base di dati.

Un database si compone di uno o più schemi (directory), che possono contenere una o più tabelle.

1.1 Data Definition Language

Operazioni principali Le operazioni principali del DDL sono:

1. Creazione tabella:

```
CREATE TABLE nomeTabella (
        nomeColonna TIPO VINCOLO,
        ...
);
```

2. Eliminazione tabella:

```
DROP TABLE nameTabella;
```

3. Modifica tabella:
   - Inserimento nuovo attributo:

     ALTER TABLE tabellaEsistente **ADD COLUMN** attributo tipo;

   - Rimozione attributo:

     ALTER TABLE tabellaEsistente **DROP COLUMN** attributo;

   - Modifica nome attributo:

     ALTER TABLE tabellaEsistente RENAME **COLUMN** vecchioNome TO nuovoNome;

   - Aggiunta vincolo di tabella:

     ALTER TABLE tabellaEsistente **ADD** vincolo;

   - Inserimento o eliminazione del vincolo associato all'attributo:

     ALTER TABLE tabellaEsistente **ALTER COLUMN** attributo
     SET VINCOLO (valore) "oppure" **DROP** vincolo;

   - Inserimento nuovo vincolo di integrità referenziale:

     ALTER TABLE tabellaEsistente **ADD CONSTRAINT** definizioneVincolo;

- Rimozione nuovo vincolo di integrità referenziale:

  ALTER TABLE tabellaEsistente **DROP CONSTRAINT** nomeVincolo;

  Il paramentro $nomeVincolo$ è il nome dato durante la dichiarazione. Se non viene dato, il DBMS ne assegna lui uno.

Domini elementari I domini elementari sono:

CHAR	permette di rappresentare singoli caratteri o stringhe di lunghezza fissa (CHAR(20) == stringa di 20 caratteri). Se la stringa non raggiunge la dimensione fissata, il sistema aggiunge degli spezi. I caratteri devono essere dichiarati all'interno di apici singoli (' ');
VARCHAR(dim)	permette di rappresentare stringhe di dimensione variabile (VARCHAR(20) indica che la dimensione max della stringa è 20 caratteri). Le stringhe devono essere dichiarate all'interno di apici singoli (' ');
TEXT	Stringa di dimensione infinita (postgreSQL);
BOOL	permette di rappresentare i valori booleani. Può assumere tre valori: TRUE, FALSE, NULL;
SMALLINT	permette di rappresentare numeri interi compresi tra -32.768 e +32.767. Equivale al tipo short di Java;
INTEGET	permette di rappresentare numeri compresi tra -2^{31} e +2^{31}-1. Equivale al tipo int di Java;
REAL	permette di rappresentare valori in virgola mobile con una precisione di 6 cifre decimali;
DOUBLE PRECISION	Come REAL, ma la precisione è di 15 cifre decimali;
NUMERIC (o **DECIMAL**)	permette di rappresentare numeri decimali in virgola fissa. Il numero è rappresentato in modo esatto (nessun errore di approssimazione in seguito a operazioni aritmetiche). Solitamente il dominio si specifica come NUMERIC(precisione, scala), dove precisione indica il numero di cifre significative (destra e sinistra della virgola), mentre scala indica il numero di cifre dopo la virgola (NUMERIC(5, 2) mi permette di rappresentare numeri come 100.01);
DATE	permette di rappresentare date nel formato YYYY-MM-DD. Le date devono essere specificate tra apici singoli (' ');
TIME[(precisione)][WITH TIME ZONE]	permette di rappresentare un orario all'interno della giornata. La precisione permette di indicare i decimali da usare per rappresentare frazioni del secondo. WITH TIME ZONE permette di specificare un fuso orario (nomeFusoOrario o $+-$hour:minute come differenza con l'ora di Greenwich);
TIMESTAMP[(precisione)][WITH TIME ZONE]	DATE + TIME (TIMESTAMP);

INTERVAL[fields][(p)] permette di rappresentare una durata temporale. Il campo fields permette di limitare l'estensione dell'intervallo, mentre il campo p permette di specificate la precisione delle frazioni di secondo (se fields contiene anche i secondi). Gli intervalli possono essere specificati in più modi:

1. Formato SQL -> es: '3 4:05:06' == 3 giorni, 4 ore, 5 minuti, 6 secondi. Il massimo rappresentabile sono i giorni;
2. Formato ISO -> es: 'P1Y2M3DT4H5S6' == 1 anno, 2 mesi, 3 giorni, 4 ore, 5 minuti, 6 secondi. La lettera P denota l'inizio della data, la lettera T l'inizio dell'ora.

Domini specifici È possibile creare un dominio specifico (partendo da uno elementare) nel seguente modo:

CREATE DOMAIN nome **AS** tipoBase dominioBase
[vincolo].

Solitamente come vincolo si usa il CHECK.

Esempio:

CREATE DOMAIN giorniSettimana **AS CHAR**(3)
CHECK(**VALUE IN**('LUN', 'MAR', 'MER', 'GIO', 'VEN', 'SAB', 'DOM'));

Vincoli In base a cosa vengono associati, i vincoli si dividono in:

- Vincoli di attributi: solitamente si specifica dopo il nome dell'attributo e si riferisce solo a quell'attributo;
- Vincoli di tabella: solitamente specificati dopo tutti gli attributi, si riferiscono a gruppi di attributi.

I vincoli principali sono:

DEFAULT si utilizza per specificare un valore di base da assegnare al campo nel caso non venga specificato. Posso assegnare anche il valore DEFAULT.

Esempio:

prezzo **NUMERIC DEFAULT** 9.99;

CONSTRAIT si utilizza per assegnare un nome ad un gruppo di vincoli (es: è più semplice rimuoverli). Raggruppa tutti i vincoli che lo seguono fino alla prima virgola.

Esempio:

price **NUMERIC CONSTRAINT** pPrice **NOT NULL CHECK** (price > 0);

NOT NULL si utilizza per indicare che il campo non può essere nullo (Attenzione: una stringa vuota non è NULL);

Esempio:

nome **VARCHAR** (20) **NOT NULL**;

CHECK si utilizza per indicare che l'attributo deve rispettare la condizione indicata nel check. Il vincolo viene rispettato se la sua espressione booleana è vero o nulla. Se l'espressione è false, il sistema dà errore;

Esempio:

stipendi **NUMERIC CHECK**(stipendio >= 0.0);
CHECK(nome <> cognome);

NULL	si utilizza per indicare che il campo deve essere nullo;
UNIQUE	si utilizza per imporre che l'attributo sia superchiave, ovvero utti i valori assegnati a quell'attributo devono esser diversi tra loro (oppure NULL). Se si riferisce a più attributi, ogni gruppo deve avere sempre almeno un valore diverso; Esempio: nome **VARCHAR** (20) **UNIQUE**; **UNIQUE** (nome, cognome);
PRIMARY KEY	si utilizza per identificare una chave primaria. Si ottiene dalla composizione di UNIQUE + NOT NULL. Questo attributo può essere può essere inserito una sola volta in ogni tabella. Esempio: matricola **CHAR**(6) **PRIMARY KEY**; **PRIMARY KEY**(nome, cognome);

Valore NULL Il valore NULL di SQL è un valore speciale:

- NULL = NULL è falso. Il NULL è sempre diverso da se stesso;
- Per eseguire confronti col NULL si devono usare specifiche operazioni.

Vicoli di integrità referenziale Un vincolo di integrità referenziale crea un legame tra i valori di un attributo, o insiemi di attributi, della tabella corrente, detta interna (o slave), e i valori di un attributo, o insieme di attributi, di un'altra tabella, detta esterna (o master). Il legame impone che:

- Il valore dell'attributo di ogni riga della tabella interna sia presente tra quelli della tabella esterna. Se il valore nella tabella interna è NULL (es: non ho specificato NOT NULL e le ho assegnato NULL) allora il vincolo è soddisfatto;
- Il valore dell'attributo di ogni riga della tabella interna sia presente anche tra quelli della tabella esterna, a meno che questo non sia NULL. L'attributo, nella tabella esterna, deve essere vincolato da UNIQUE o PRIMARY KEY (basta che identifichi le righe della tabella esterna, non deve essere necessariamente la sua chiave primaria).

Esistono due modi per esprimere questo vincolo:

1. REFERENCES -> si utilizza quando il vincolo è usato solo su un attributo della tabella interna. Si dichiara come segue:

```
CREATE TABLE tabellaInterna(
        ...
        attributo tipo REFERENCES
                tabellaEsterna(attributoChiave);
);
```

2. FOREIGN KEY -> si utilizza quando il vincolo è usato su più attributi della tabella interna. Si dichiara come segue:

```
CREATE TABLE tabellaInterna(
        ...
        attributo1 tipo,
        ...

        FOREIGN KEY(attributo1, ...) REFERENCES tabellaEsterna (nomeAttributo1Esterna, ...);
);
```

Attenzione:

- La tabella master deve essere creata prima della tabella slave;
- Il dominio dei due attributi collegato deve essere compatibile (meglio uguale);
- Senza l'uso di comandi specifici, un valore dell'attributo collegato può essere eliminato dalla tabella master solo se non compare nell'attributo della tabella slave.

  I voncoli di integrità referenziali possono essere rimossi e agginti anche dopo la creazione della tabella (vedi sopra).

1.2 Data Management Language

Operazioni principali Le operazioni principali del DML sono:

1. Popolazione delle tabelle:

   INSERT INTO tabellaEsistente (elencoAttributi)
   VALUES (elencoValori),
   ...
   ;

   Posso aggiungere una o più righe con ogni comando VALUES.

2. Modifica dei valori di una riga:

   UPDATE tabellaEsistente
   SET attributo = espressione,
   ...
   [**WHERE** condizione];

   Se non specifico il WHERE, la modifica verrà applicata a tutte le righe. Devo ripetere UPDATE per ogni modifica differente dalla precedente.

3. Eliminazione righe:

   DELETE FROM tabellaEsistente
   [**WHERE** condizione];

Operatori SQL Alcuni operatori particolari sono:

Concatenzazione stringhe	||
Confronto e assegnazione	=

1.2.1 Data Query Language

Il processo per recuperare dati dal database è chiamato query/interrogazione.
Il comando SQL per effettuare una query è:

SELECT [**DISTINCT**]
[* | expression [[**AS**] output_name] [, ...]]
[**FROM** from_item [, ...]]
[**WHERE** condition]
[**GROUP BY** grouping_element [, ...]]
[**HAVING** condition [, ...]]
[{ **UNION** | **INTERSECT** | **EXCEPT** } [**DISTINCT**]
other_select]
[**ORDER BY** expression [**ASC** | **DESC** | **USING** operator]]
...

Nello specifico:

*	ritorna tutti gli attributi delle tabelle indicate.

DISTINCT elimina le tuple risultato duplicate.

AS permette di assegnare un nome a expression (assegna un nome per la colonna risultato nella tabella risultato temporanea). I nuovi nome possono essere del tipo:

- NOME -> viene interpretato come nome. Non è case sensitive;
- "Nome" -> viene interpretato come Nome. È case sensitive.

FROM specifica la tabella da cui prendere le tuple. Il campo from_item può realizzare uno dei seguenti valori:

- Prodotto cartesiano:

  nomeTabella [[**AS**] alias [(alisaColonna [, ...])]]

  Se sono presenti due o più tabelle, si esegue il prodotto cartesiano tra tutte le tabelle. Se ci sono attributi con lo stesso nome, nel SELECT questi vengono identificati con nomeTabella.nomeAttributo;

- Query Innstata:

  (other_select) [**AS**] nomeRisultato [(aliasColonna [, ...])]

  Con questa forma creo una query innestata. Il risultato della SELECT interna è la tabella con nome nomeTisultato su cui fare la query corrente;

- Join:

  from_item [**NATURAL**] join_type from_item [**ON** condition]

  Questa forma permette di selezionare un sottoinsieme del prodotto cartesiano di due o più tabelle. Il campo join_type specifica il tipo di join. I tipi principali sono riassunti nelle slide seguenti.

- Altre opzioni più complesse.

WHERE è seguita da una condizione booleana. La condizione può contenere:

- Espressioni relative alle colonne richieste nella prima riga;
- **AND**, **OR**, **NOT**;
- =, <>, <, >, <=, >=;
- Costanti appartenenti ai domini di base;
- **BETWEEN** val1 **AND** val2 -> ritorna true se il valore è compreso tra val1 e val2 (entrambi compresi);
- **IN**(val1, val2, ...) -> ritorna true se il valore è uguale a val2 o val2 o ...;
- **IS** [**NOT**] **NULL**;

Nella WHERE non è possibile utilizzare i nomi dell'AS assegnati nella prima riga. Possono anche comparire i seguenti operatori:

- LIKE -> operatore di pattern matching. Le stringhe di controllo del LIKE si compongono con due caratteri speciali:
  - "_" == 1 carattere qualsiasi;
  - "%" == 0 o più caratteri qualsiasi;

Esempio:
Tutti gli studenti di una città che inizia con 'V' e finisce con 'a':

SELECT cognome, nome, citta **FROM** Studente
WHERE citta **LIKE** 'V%a';

Per ignorare maiuscole/minuscole si utilizza ILIKE.

- SIMILAT TO -> è una versione più estesa del LIKE che accetta un sottoinsieme delle espressioni POSIX (versione SQL). Le stringhe di controllo compongono con:
  - "_" == 1 carattere qualsiasi;
  - "%" == 0 o più caratteri qualsiasi;
  - "{n, m}" == ripetizione del precedene match almeno n e max m volte;
  - "[...]" == elenco di caratteri ammissibili.

  Esempio:
  Studenti con cognome che inizia con 'A', 'B', 'D' o 'N' e finisce con 'a':

  SELECT cognome, nome, citta **FROM** Studente
  WHERE cognome SIMILAR TO '[ABDN]{1}%a';

ORDER BY ordina le tuple risultato in ordine rispetto agli attributi specificati. Si specifica come ultima clausola. La struttura è:

ORDER BY attributo [**ASC** | **DESC**], attributo2 [**ASC** | **DESC**], ...;

Le tuple vengono ordinate secondo il primo attributo, se vi sono valori uguali sul primo, si ordina basandosi sul secondo, e così via.

Operatori Aggregati permettono di determinare un valore considerando i valori ottenuti da una SELECT. Gli operatori aggregati possono ritornare un:

- Conteggio:

  COUNT({ $*$ | expr | **ALL** expr | **DISTINCT** expr }])

  Il parametro expr è un'espressione che usa attributi e funzioni di attributi (non può usare operatori di aggregazione). Nello specifico:

  - COUNT(*) -> ritorna il numero di tuple che risultano dallla query;
  - COUNT(expr) -> ritorna il numero di tuple che rispettano expr (il valore di expr non è null);
  - COUNT(ALL expr) -> alias a COUNT(expr);
  - COUNT(DISTINCT expr) -> ritorna il numero di tuple che rispettano expr eliminando i doppioni.

  In genere COUNT ritorna una tabella composta da una sola colonna e una sola riga.

- Valore numerico/alfanumerico:

  SUM | **AVG** | **MAX** | **MIN** ({expr | **DISTINCT** expr }])

Il risultato della SELECT:

- Ha come schema tutti gli attributi richiesti nella SELECT;
- Ha come contenuto tutte le tuple t ottenute proiettando sugli attributi dopo SELECT le tuple t_0 appartenenti al prodotto cartesiano delle tabelle ottenute dopo il FROM che soddisfano l'eventuale condizione nella clausola WHERE / HAVING / GROUP BY.

Funzioni usabili in expression Il campo expression della SELECT permette anche di applicare delle funzioni alle colonne richieste. Alcune di queste sono:

- UPPER(attributo) -> ritorna il contenuto tutto in maiuscolo;
- LOWER(attributo) -> ritorna il contenuto tutto in minuscolo;
- CHAR_LENGHT(attributo) -> ritorna la lunghezza in caratteri dell'attributo;
- CURRENT_DATE -> ritorna la data corrente in formato DATE;

Conversioni nelle query SQL permette di effettuare cast nei valori ritornati da una query. Il comando SQL per effettuarlo è:

> **CAST** (expr **AS** newType)

PostgreSQl permette di abbreviarlo come segue:

> expr::nuovoTipo

Esempio:
Calcola la media delle medie distinte degli studenti.

```
SELECT AVG(DISTINCT media)::DECIMAL(5,2)
FROM Studente;
```