Appunti di Basi di Dati

Andrea De Lorenzo, Giorgia Nadizar

May 3, 2022

1 SQL Data Definition Language

```
--Mostra i DB esistenti
SHOW DATABASES;
```

```
--Crea un nuovo DB [se non esiste]
CREATE DATABASE [IF NOT EXISTS] nomeDataBase;
```

```
--Elimina il DB [se esiste]
DROP DATABASE [IF EXISTS] nomeDataBase;
```

```
--Tutti i comandi ora saranno riferiti a questo DB
USE nomeDataBase;
```

```
--Creazione tabella
CREATE TABLE [IF NOT EXISTS] nomeTabella(
    nomeAttributo1 tipo,
    attributo2 tipo,
    ...
    attributoN tipo
)
```

1.1 Domini

- Numeri interi: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, `BIGINT`
- Numeri razionali: `float`, `double`, `numeric(i,n)`, `decimal(i,n)`
- Testo: `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, `LONGTEXT`
- Generico: `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, `LONGBLOB`
- Tempo: `YEAR`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`
- Spazio: `GEOMETRY`, `POINT`, `LINESTRING`, `POLYGON`
- Booleani: `BOOL`

1.2 Vincoli

- `PRIMARY KEY`: vicino all'attributo, o alla fine con l'attributo (gli attributi) chiave tra parentesi
- `NOT NULL`
- `UNIQUE`: vicino all'attributo o alla fine con gli attributi tra parentesi
- `CHECK` (supportato in MySQL sono da recente, non in MariaDB)
- `FOREIGN KEY (attr1) REFERENCES Tabella(attr2)`: nella versione compatta si può specificare `REFERENCES Tabella(attr2)` vicino alla definizione di `attr1`. Per specificare il comportamento in caso di cancellazione si ha `ON DELETE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT`. In caso di modifica, invece si ha: `ON UPDATE RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT`.
- `CONSTRAINT [nome] ...`: un qualsiasi tipo di vincolo può avere un nome. Si possono attivare con `SET nome = 1` e disattivare con `SET nome = 0`
- `ASSERTION`: per specificare vincoli di integrità a livello di schema

1.3 Altro

- `AUTO_INCREMENT`
- `nomeAttributo tipo DEFAULT valore`
- `nomeAttributo tipo COMMENT "commento"`

1.4 Cancellazione

```
--Elimina la tabella [se esiste]
DROP TABLE [IF EXISTS] nomeTab1, nomeTab2, ...;
```

1.5 Cambiare lo schema

```
--Modifica la tabella
ALTER TABLE nomeTabella azione1 [, azione2, ...]
```

```
--Aggiunta colonna
ALTER TABLE nomeTabella
    ADD COLUMN nome tipo
    [ FIRST | AFTER nomeColonna ]
```

```
--Eliminazione colonna
ALTER TABLE nomeTabella
    DROP COLUMN nomeColonna
```

```
--Modifica colonna
ALTER TABLE nomeTabella
    CHANGE COLUMN nomeOriginale nomeNuovo tipo
```

```
--Ridenominazione tabella
ALTER TABLE nomeTabella
    RENAME TO nuovoNome
```

```
--Aggiunta di vincoli
ALTER TABLE nomeTabella
    ADD CONSTRAINT nome
    FOREIGN KEY (...)
    REFERENCES tabella(...)
```

```
--Rimozione di vincoli
ALTER TABLE nomeTabella
    DROP FOREIGN KEY nome
```

2 SQL Data Manipulation Language

```
--Selezione
SELECT attributo1 [, attributo2, ...]
    FROM tabella1 [, tabella2, ...]
    [WHERE condizione]
    [LIMIT x, n]
```

```
--Alias sugli attributi
SELECT attributo1 AS attr,
    FROM tabella1 [, tabella2, ...]
    [WHERE condizione]
```

```
--Alias e formule
SELECT attributo1/2 AS attr1mezzi,
    FROM tabella1 [, tabella2, ...]
    [WHERE condizione]
```

```
SELECT t.attributo1,
    FROM tabella1 t
    [WHERE condizione]
```

```
--Ordinamento
SELECT ... FROM ... WHERE ...
    ORDER BY col1 [ASC|DESC], col2 [ASC|DESC], ...
```

```
--Rimozione duplicati
SELECT DISTINCT ... FROM ...
```

2.1 Condizioni

- Testo esatto: `=`
- Testo incompleto: `[NOT] LIKE "\%parteditesto"` oppure `[NOT] LIKE "_parteditesto"`
- Più condizioni: `AND` oppure `OR` con le opportune parentesi
- Intervalli: `col BETWEEN x AND y` oppure `col >= x AND col <= y`
- Liste: `col IN (val1, val2, ...)`
- Null: `col IS NULL`

2.2 Funzioni

- Stringhe: `length()`, `reverse()`, `right()`, `trim()`, `concat()`
- Data e ora: `day()`, `year()`, `now()`, `month()`, `monthname()`
- Ordinamento: `FIELD(text, str1, str2, str3, ...)` ritorna la posizione della stringa `text` nella lista `str1, str2, str3, ...`

2.3 Decodificare le relazioni, unione ed intersezione

```
--Prodotto cartesiano
SELECT ... FROM tabella1, tabella2, ...
SELECT * FROM tabella1 CROSS JOIN tabella2;
```

```
--Inner join
SELECT ... FROM tabella1 t1
    INNER JOIN tabella2 t2
    ON t2.PK = t1.FK;
```

```
SELECT ... FROM tabella1
    INNER JOIN tabella2
    USING(attrComune);
```

```
--Senza specificare using (pericolose in caso di
   modifiche)
SELECT ... FROM tabella1
    NATURAL JOIN tabella2;
```

```
SELECT ... FROM tabella1
    LEFT [OUTER] JOIN tabella2
    ON PK = FK
```

```
SELECT ... FROM tabella1
    RIGHT [OUTER] JOIN tabella2
    ON PK = FK
```

```
SELECT ... FROM tabella1
    FULL [OUTER] JOIN tabella2
    ON PK = FK
```

```
SELECT ... FROM tabella1
    [INNER|LEFT|RIGHT|FULL] JOIN tabella2
    ON PK = FK
    [INNER|LEFT|RIGHT|FULL] JOIN tabella3
    ON PK = FK
```

```
SELECT ... FROM tabella1 t1
    [INNER|LEFT|RIGHT|FULL] JOIN tabella2 t2
    ON t1.a1 = t2.a2 AND t1.b1 <> t2.b2
```

```
--Unione (stessa struttura dei risultati richiesta)
SELECT ... FROM ...
    UNION [DISTINCT | ALL]
    SELECT ... FROM ...
    [UNION [DISTINCT | ALL]
    SELECT ... FROM ...]
    [ORDER BY criteri]
```

```
--Intersezione
SELECT ... FROM ...
    INTERSECT
    SELECT ... FROM ...
```

2.4 Raggruppare i dati

```
SELECT a1,a2,...,an
FROM tabella1 WHERE condizioni
GROUP BY a1,a2,...,an
```

```
SELECT a1,a2,...,an, aggregatore(ax)
FROM tabella1 WHERE condizioni
```

Esempi di aggregatori: `COUNT`, `SUM`, `AVG`, `MAX`, `MIN`

```
SELECT a1,a2,...,an, aggregatore(ax)
FROM tabella1 WHERE condizioni
GROUP BY a1,a2,...,an
```

```
SELECT a1,a2,...,an, aggregatore(ax)
    FROM tabella1 WHERE condizioni
    GROUP BY a1,a2,...,an
    HAVING condizioniAggregate
```

2.5 Subqueries

```
--Subquery indipendenti
SELECT a1,a2,...,an,(QUERY singolo val.)
    FROM (QUERY)
    WHERE a1 > (QUERY singolo val.)
    AND a2 IN (QUERY singolo attrib.)
```

```
--Subquery correlate
SELECT a1,a2,...,an
    FROM tab1 WHERE
    a1 > (SELECT c1
    FROM tab2
    WHERE tab2.c2 > tab1.a1)
```

```
SELECT a1,a2,...,an
    FROM tab
    WHERE EXISTS (QUERY singolo val.)
```

2.6 Aggiungere dati

```
--Inserimento a mano
INSERT INTO tabella(col1, col2, ...)
    VALUES (valore1, valore2, ...)
    [, (valore1, valore2, ...), ...]
```

```
--Inserimento e subquery
INSERT INTO tabella(col1, col2, ...)
    SELECT ...
```

2.7 Modificare dati

```
UPDATE tabella
SET col1 = valore1
[, col2 = val2...]
[WHERE condizione]
```

Attenzione: per MySQL 8 si richiede di disabilitare i "safe updates" (`SET SQL_SAFE_UPDATES = 0;`) per proseguire nel caso in cui nella clausola `WHERE` non venga specificata una chiave.

2.8 Eliminare dati

```
DELETE FROM tabella
[WHERE condizioni]
```

Vale lo stesso discorso sui "safe updates" visto per l'aggiornamento dei dati.

2.9 Prepared Statement

```
--Precompilo le query che uso spesso
PREPARE nomeStatement FROM "query come stringa";
EXECUTE nomeStatement USING p1,p2,...;
DEALLOCATE PREPARE nomeStatement;
```

```
--Creo lo statement
--Query passata come stringa
--Parametri indicati con ?
PREPARE nomeStatement FROM
"SELECT a1,a2,...
FROM tabella
WHERE a1 = ? AND a2 = ?";
```

```
--Eseguo lo statement
--I parametri sono opzionali
EXECUTE nomeStatement
[USING @var1,@var2,...];
```

```
--Elimino lo statement
DEALLOCATE PREPARE nomeStatement;
```

2.10 Viste

- MERGE
- TEMPTABLE (materialized)

```
--Creazione
CREATE VIEW nomeVista AS
SELECT ...
```

```
--Cosa fa?
SHOW CREATE VIEW nomeVista;
```

```
--Eliminazione
DROP VIEW nomeVista;
```

```
--Modifica vista
ALTER VIEW nomeVista AS nuovaSELECT;
```

3 Controllo accessi

```
--Aggiunta utente
CREATE USER nome@host
IDENTIFIED BY "password";
```

```
--Cambio password
SET PASSWORD FOR user@host
PASSWORD("password");
```

```
--Eliminare un utente
DROP user@host;
```

```
--Assegnare i permessi
GRANT privilegio (colonne)
ON risorsa
TO account
[WITH GRANT OPTION]
```

Privilegi: `ALL`, `ALTER`, `CREATE`, `DELETE`, `SELECT`, `UPDATE`, ...

```
--Visualizzare i permessi
SHOW GRANTS FOR utente;
```

```
--Revocare i permessi
REVOKE privilege_type [(column_list)]
[, priv_type [(column_list)]]...
ON [object_type] privilege_level
FROM user [, user]...
```

4 Transazioni

- Atomicità → operazioni eseguite per intero o per niente
- Consistenza → vincoli rispettati alla fine (non necessariamente durante)
- Isolamento → coerenza in caso di concorrenza
- Durabilità → si tiene traccia del risultato

```
--Inizio della transazione
START TRANSACTION
```

```
--Esecuzione delle operazioni
COMMIT
```

```
--Annullamento esecuzione
ROLLBACK
```

- Write Ahead Logging → blocchi modificati annotati sul log, copiati sul file del DB al commit
- Command Logging → log contiene storico istruzioni, eseguite realmente solo al commit

5 SQL Programming Language

5.1 Stored Procedures (SP)

- Subroutine che contengono tutto il codice necessario per effettuare un'operazione
- Incapsulare logica di accesso alle tabelle e manipolazione dati
- Fornire un ulteriore livello di astrazione

```
--Creazione
CREATE PROCEDURE nome()
BEGIN
... codice;
END
```

```
--Cambio il delimitatore per interpretare tutto
   insieme
DELIMITER $$
CREATE PROCEDURE nome()
BEGIN
... codice1;
... codice2;
END $$
DELIMITER ;
```

```
--Esecuzione
CALL nome()
```

```
--Vedere tutte le SP
SHOW PROCEDURE STATUS
[WHERE condizioni su db o nome]
```

```
--Vedere il codice (la definizione) di una SP
SHOW CREATE PROCEDURE nome
```

```
--Eliminare una SP
DROP PROCEDURE nome
```

```
--Modificare una SP
DROP + CREATE
```

5.1.1 SP con parametri

```
--Definizione
CREATE PROCEDURE nome(
    nomePar1 tipoPar1,
    nomePar2 tipoPar2, ...
)
BEGIN
... codice
END
```

```
--Chiamata
CALL nome(par1, par2, ...)
```

- `IN` → parametri in sola lettura (default)
- `OUT` → parametri in sola scrittura
- `INOUT` → leggibili e scrivibili

```
--Scrivere il risultato di una query in una variabile
SELECT ...
INTO var
...
```

5.1.2 Variabili

- Globali `@@` → tutti le vedono
- Connessione `@` → ogni connessione vede le proprie
- Locali → interne alla SP (vanno dichiarate con `DECLARE`)

5.1.3 Condizioni

```
IF espressione THEN
    comandi
ELSEIF espressione THEN
    comandi
ELSE
    comandi
END IF;
```

5.1.4 Cicli

```
WHILE espressione DO
comandi
END WHILE;
```

```
REPEAT
comandi
UNTIL espressione
END REPEAT;
```

- `LEAVE` → esce dal ciclo
- `ITERATE` → procede con l'iterazione successiva

5.1.5 Gestione errori

```
DECLARE azione HANDLER FOR
    condizione [BEGIN] codice [END]
```

- Condizione → cosa vogliamo intercettare
- Codice → cosa fare
- Azione → come comportarsi dopo aver eseguito il codice
  - `CONTINUE` → continua con il resto
  - `EXIT` → termine l'esecuzione
- `SQLWARNING` → SQLSTATE che iniziano con 01
- `NOT FOUND` → SQLSTATE che iniziano con 02
- `SQLEXCEPTION` → SQLSTATE che non iniziano con 00, 01, o 02

```
--Segnalare errori
SIGNAL SQLSTATE "codice"
SET MESSAGE_TEXT = "testo"
```

5.2 Cursori

- Read-only → non è possibile aggiornare i dati tramite cursore
- Non-scrollable → si può scorrere il dataset senza cambiare ordine
- Asensitive → puntano ai dati reali

```
--Definizione cursore e query che lo usa
DECLARE nomeCursore CURSOR FOR
SELECT ...

--Apro il cursore, eseguendo la query
OPEN nomeCursore

--Uso il cursore in un ciclo
FETCH nomeCursore INTO var1, var2, ...

--Chiudo il cursore (libero memoria)
CLOSE nomeCursore

--Handler per smettere di usare il cursore
DECLARE CONTINUE HANDLER FOR NOT FOUND SET over = 1;
--Poi continuo ad iterare fino a che over diventa 1
```

5.3 Used Defined Functions (UDF)

- Scalar Functions
  - Simile ad una built-in function
  - Ritorna un singolo valore costruito con una serie di statements
- Multi-Statement Table-valued Functions
  - Contenuto simile ad una stored procedure
  - Referenziata come una Vista
- In-line Table-valued Functions
  - Simile ad una Vista con parametri
  - Ritorna una tabella come risultato di uno statement `SELECT` singolo

```
--Definizione
CREATE FUNCTION function_name
(param1 tipo1,param2 tipo2,...)
RETURNS tipo
[NOT] DETERMINISTIC
BEGIN
    statements
END
```

5.3.1 UDF vs. SP

	SP	UDF
Risultato	0 o N valori	sempre 1 valore
Parametri	input/output	solo input
Modifiche	no modifiche al DB	solo SELECT
Invocazione	può richiamare UDF	non può richiamare SP
SELECT	non usabile in una SELECT	usabile in una SELECT
RecordSet	tabella restituita non utilizzabile	usabile come tabella

5.4 Trigger

- Operazioni da eseguire quando si verifica un certo evento (es. si effettua un'operazione su una certa tabella)
- Non viene richiamata direttamente
- Possono essere legati ad eventi temporali
- Non possono usare UDF, SP, e prepared statements (in MySQL)
- Utili per controlli dell'integrità dei dati

```
--Creazione
CREATE TRIGGER nome quando
ON nomeTabella
[FOR EACH ROW]
BEGIN
    codice
END
```

5.4.1 Quando?

- `BEFORE` → i dati sono corretti? Posso segnalare errori come visto per le SP
- `AFTER` → registro le modifiche, ricalcolo valori, ...

5.4.2 Granularità

- `STATEMENT LEVEL` → eseguito una volta sola per ogni comando che lo ha attivato
- `ROW LEVEL` → eseguito una volta per ciascuna tupla che è stata modificata dal comando

5.4.3 `OLD` e `NEW`

- `OLD` → valore precedente
  - usabile nel `DELETE` e nel `BEFORE UPDATE`
- `NEW` → valore dopo le modifiche
  - usabile nell'`INSERT` e nel `BEFORE UPDATE`

5.4.4 Conflitti tra trigger

1. `BEFORE STATEMENT LEVEL`
2. `BEFORE ROW LEVEL`
3. Applicazione della modifica e verifica dei vincoli di integrità
4. `AFTER ROW LEVEL`
5. `AFTER STATEMENT LEVEL`