Il linguaggio SQL costrutti DDL

foglia@iet.unipi.it

Sommario

- Costrutti per la creazione dello schema
- Costrutti per la creazione delle tabelle
- Costrutti per la modifica dello schema
  - modifica/cancellazione delle tabelle e dei vincoli
- Costrutti per la creazione delle viste

Data Definition Language (DDL)

- Insieme di istruzioni utilizzate per modificare la struttura della base di dati, tra cui istruzioni di **inserimento, cancellazione** e **modifica** di tabelle

- Il **DDL** di SQL permette:
  - di **definire schemi di relazioni** (o “table”, tabelle), modificarli ed eliminarli
  - di **specificare vincoli**, sia a livello di tupla (o “riga”) che a livello di tabella
  - di **definire nuovi domini**, oltre a quelli predefiniti
  - di **definire le viste** (“**view**”), ovvero *tabelle virtuali*

Creazione di uno schema

- SQL consente la definizione di uno schema di base di dati come collezione di tabelle attraverso la seguente sintassi:

CREATE SCHEMA [if not exists] NomeSchema [autorizzazione];

- Creato un determinato schema questo può essere selezionato per le future azioni attraverso il seguente comando:

USE NomeSchema;

- Per cancellare un intero database invece:

DROP SCHEMA [if exists] NomeSchema;

attenzione: cancella anche tutte le tabelle in esso

Creazione di una tabella

- La creazione di una tabella avviene attraverso l'enumerazione delle colonne che la compongono. Per ogni attributo va specificato il dominio, un eventuale valore di default e eventuali vincoli.
- Mediante l'istruzione **CREATE TABLE** si definisce lo schema di una tabella e se ne crea un'istanza vuota.
- I tipi di dato associati con ciascun attributo possono essere scelti tra i tipi di base già definiti in SQL oppure tra nuovi tipi definiti dall'utente

Sintassi:

CREATE TABLE [IF NOT EXISTS] Tabella

(<nome_campo1> <tipo_campo1>,

<nome_campo2> <tipo_campo2>,...)

Esempio:

```
CREATE TABLE Studenti(
    Matricola char(10),
    Nome char(20) NOT NULL,
    Cognome char(20) NOT NULL,
    AnnoDiIscrizione integer,
    CONSTRAINT pk_matricola PRIMARY KEY (Matricola)
    )
```

Valori di default

- I valori di default specificano cosa deve essere assegnato all'attributo (colonna) quando non si indica un valore esplicitamente.
- **Se non si specifica un valore di default, si assume come valore di default null**
- **Esempi:**
- AnnoDilscrizione integer **DEFAULT** 1
- NumeroPatente char(20) **DEFAULT NULL**

Vincoli d’integrità

- Durante la creazione di una tabella, possono essere definiti i vincoli d’integrità
- Questi (a seconda del tipo) possono essere associati ad un attributo singolo o a tutta la relazione
- Le tipologie di vincoli sono i seguenti:
  - Vincoli intra-relazionali
  - Vincoli inter-relazionali
  - Vincoli controllo

Vincoli intra-relazionali

Vincolo NOT NULL:

- Vieta la presenza di valori nulli in quella colonna

```
CREATE TABLE Studenti(
        Matricola char(10),
        Nome char(20) NOT NULL DEFAULT 'pippo',
        Cognome char(20) NOT NULL,
        AnnoDiIscrizione integer,
        …
        PRIMARY KEY (Matricola));
```

- Il valore di default viene specificato dopo il vincolo
- Il vincolo not null può essere specificato per una colonna, non per coppie di colonne.

Vincoli intra-relazionali

Vincolo UNIQUE:

- Non possono esistere due righe che hanno gli stessi valori per l'attributo o insieme di attributi specificati

```
CREATE TABLE Studenti(
    Matricola char(10),
    CodiceFiscale char(16) UNIQUE,
    Nome char(20) NOT NULL,
    Cognome char(20) NOT NULL,
    AnnoDiIscrizione integer,
    …
    PRIMARY KEY (Matricola))
```

- *Il vincolo unique impone che gli attributi formino una super-chiave*
- *Il vincolo UNIQUE non esclude la presenza di più righe con valori NULL, che si assumono tutti diversi fra loro*

Ordine dei vincoli se espressi un una colonna

- Il valore di defaultva messo fra l'eventuale not null e unique.

```
CREATE TABLE Studenti(
Matricola char(10),
 Nome char(20) not null default 'pippo' unique,
Cognome char(20) NOT NULL,
AnnoDiIscrizione integer,
PRIMARY KEY (Matricola));
```

Vincolo unique per più di una colonna

- Se il vincolo unique riguarda un insieme di attributi, la sintassi è:

```
CREATE TABLE Studenti(
Matricola char(10),
Nome char(20) NOT NULL ,
Cognome char(20) NOT NULL,
AnnoDiIscrizione integer,
UNIQUE(nome, cognome),
PRIMARY KEY (Matricola)
);
```

- In alternativa:

```
CREATE TABLE Studenti(
Matricola char(10),
Nome char(20) NOT NULL ,
Cognome char(20) NOT NULL,
AnnoDiIscrizione integer,
constraint un1 UNIQUE(nome, cognome),
PRIMARY KEY (Matricola));
```

Vincoli intra-relazionali

Vincolo PRIMARY KEY:

- Identifica la chiave primaria (**vincolo NOT NULL e UNIQUE**)
- Può esserci un solo vincolo di primary key in una tabella

```
CREATE TABLE Veicoli(
    Targa char(10),
    CodiceProprietario char(20) NOT NULL,
    …
    PRIMARY KEY (Targa,CodiceProprietario ))
```

Oppure

```
CREATE TABLE Veicoli(
    Targa char(10),
    CodiceProprietario char(20) NOT NULL,
    …
    CONSTRAINT pk PRIMARY KEY (Targa,CodiceProprietario ))
```

Oppure

```
CREATE TABLE Veicoli(
    Targa char(10) PRIMARY KEY,
    CodiceProprietario char(20) NOT NULL,
    )
```

Le stesse tre modalità possono essere usate per UNIQUE

Solo nel caso in cui la chiave primaria sia composta da un solo attributo.

Vincoli intra-relazionali

Vincolo di controllo CHECK:

- I vincoli di controllo sono utilizzati per verificare generiche condizioni sui valori di una colonna.
- Il vincolo è violato se esiste almeno una tupla che rende falsa la condizione

```
CREATE TABLE Esami(
        Matricola char(10),
        Corso char(20) UNIQUE,
        Voto integer CHECK (Voto > 18 AND Voto < 30),
        • …
        PRIMARY KEY (Corso, Matricola))
```

NOTA: Se **CHECK** viene espresso a livello di tabella (anziché nella definizione dell'attributo) è possibile fare riferimento a più attributi della tabella stessa

Es: **CHECK** (ImportoLordo = Netto + Ritenute)

In Alternativa:

CONSTRAINT Esami_ch CHECK(Voto > 18 AND Voto < 30)

Vincoli inter-relazionali

- Il più significativo è il vincolo di integrità referenziale, implementato in SQL tramite il vincolo di foreign key, o di chiave esterna
- Creano un legame tra i valori dell'attributo della tabella in cui è definito (tabella interna o referente) e i valori di un attributo di un'altra tabella (tabella esterna o riferita)

Tabella interna

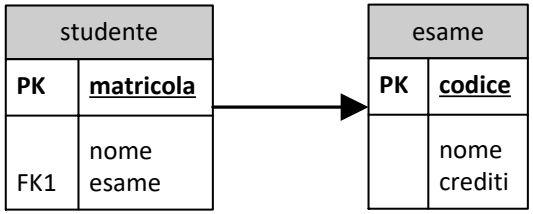

Tabella esterna

- Il vincolo impone che, per ogni tupla, il valore dell'attributo della tabella interna**, se diverso da null**, deve essere uguale ad un valore dell'attributo della tabella esterna.
- L'attributo della tabella esterna deve essere soggetto ad un **vincolo unique** (e di solito è primary key)

Vincolo REFERENCES: se l'attributo è unico

Vincolo REFERENCES:

- Permette di specificare vincoli di colonna.

```
CREATE TABLE Impiegati (
    Matricola       char(6) PRIMARY KEY,
    Cognome         varchar(50) NOT NULL,
    Nome            varchar(50) NOT NULL,
    Dipartimento    char(15) REFERENCES Dipartimenti(NomeDipartimento),
    ...
)
```

- Il campo Dipartimento può assumere solo i valori che compaiono nel campo NomeDipartimento della tabella Dipartimenti

Vincolo FOREIGN KEY: se si riferiscono più attributi

- **Vincolo FOREIGN KEY:**
- La definizione di una foreign key avviene specificando un vincolo e indicando quale chiave viene referenziata (vale anche per il singolo attributo).

```
CREATE TABLE Impiegati (
    Matricola        char(6) PRIMARY KEY,
    Cognome          varchar(50) NOT NULL,
    Nome             varchar(50) NOT NULL,
    Dipartimento     char(15),
    foreign key (nome, cognome) REFERENCES anagrafica(nome, cognome),
    ...
)
```

In alternativa

```
CONSTRAINT fk  FOREIGN KEY (nome, cognome) REFERENCES anagrafica(nome, cognome)
```

Politiche di reazione di SQL

- Per i vincoli precedenti escluso foreign key, il DBMS, quando rileva un aggiornamento (inserimento, aggiornamento, cancellazione) per cui il vincolo viene violato, **rifiuta** l'aggiornamento e segnala un errore

- Nel caso del vincolo di **foreign key**, il vincolo può essere violato operando sia sulla tabella interna che sulla tabella esterna.
  - nel caso della tabella interna (inserimento o modifica dell'attributo referente) l'operazione viene rifiutata
  - Nel caso della tabella esterna (op. di modifica o cancellazione dell'attributo riferito), è possibile definire diverse politiche.

Politiche di reazione: update e cancellazioni

- Per garantire che a fronte di cancellazioni e modifiche i vincoli di integrità referenziale siano rispettati, si possono specificare opportune **politiche di reazione** in fase di definizione degli schemi

- In particolare tramite la seguente sintassi può essere definita l'azione che il sistema intraprende quando viene richiesta la cancellazione o l'update di una tupla della tabella esterna con un attributo legato da vincolo di foreign key:

ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }
ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }

Politiche di reazione: on update

- ON UPDATE: quando si aggiorna l'attributo della tabella esterna

CASCADE
- Se si tenta di aggiornare un valore di un attributo in una riga e a tale attributo fanno riferimento (per valore) chiavi esterne in righe in altre tabelle, tali valori verranno anch'essi aggiornati al nuovo valore

SET NULL

- Se si tenta di aggiornare un valore di un attributo in una riga cui fanno riferimento chiavi esterne in righe esistenti in altre tabelle, a tali valori viene assegnato il valore NULL.

SET DEFAULT
- All'attributo referente corrispondente (tabella interna) viene assegnato il valore di Default

NO ACTION
- Se si tenta di aggiornare un valore di un attributo in una riga cui fanno riferimento chiavi esterne in righe esistenti in altre tabelle, verrà generato un errore e l'operazione non viene eseguita.

Politiche di reazione: on delete

- ON DELETE: quando si cancella una riga nella tabella esterna

CASCADE
- Se si tenta di eliminare una riga contenente un attributo a cui fanno riferimento chiavi esterne in righe di altre tabelle, tali righe verranno eliminate

SET NULL
- Se si tenta di eliminare una riga contenente un attributo cui fanno riferimento chiavi esterne in righe di altre tabelle, a tali valori viene assegnato il valore NULL..

.
SET DEFAULT
- Se si tenta di eliminare una riga contenente un attributo cui fanno riferimento chiavi esterne in righe di altre tabelle, tutti i valori che compongono la chiave esterna presenti in tali righe verranno impostati sul relativo valore predefinito.

NO ACTION
- Se si tenta di eliminare una riga contenente un attributo cui fanno riferimento chiavi esterne in righe esistenti in altre tabelle, verrà generato un errore e l'operazione non sarà consentita.

Politiche di reazione e MYSQL

- Se non si specifica alcuna politica di reazione, si assume la politica NO ACTION (o RESTRICT)
- RESTRICT è un sinonimo di NO ACTION

- La politica SET DEFAULT non è ammessa

Vincoli inter-relazionali update e cancellazioni

Esempio:

```
CREATE TABLE Iscrizioni(
        CodIscrizione char(20),
        Matricola char(10),
        ...
        CONSTRAINT "PK_Iscrizioni" PRIMARY KEY (CodIscrizione, Matricola),
        CONSTRAINT “FK_Studenti" FOREIGN KEY (Matricola) REFERENCES Studenti(Matricola)
        ON DELETE CASCADE                    -- cancellazione in cascata
        ON UPDATE NO ACTION      -- modifiche non permesse
)
```

- Se una riga di Studenti e’ cancellata verranno cancellate tutte le righe di Iscrizioni che la referenziano
- Se la colonna Matricola di Studenti e’ modificata, l’aggiornamento deve essere rifiutato se una riga di Iscrizioni punta alla riga modificata

- **Si possono definire differenti politiche su ON DELETE e ON UPDATE**

Modifica dello schema

- Istruzione **DROP TABLE**:
  - Rimuove la tabella e eventualmente le tabelle dipendenti
- **Esistono due modalità di drop (anche se oggi sono in disuso):**
  - **CASCADE**: Elimina la tabella, il suo contenuto e le viste connesse, ed i vincoli che fanno riferimento ad essa.
  - **DROP TABLE** Studenti **CASCADE**
  - **RESTRICT**: Elimina la tabella solo se è vuota e non ci sono oggetti connessi (ad esempio da relazioni di chiave esterna)
  - **DROP TABLE** Studenti **RESTRICT**

- Sintassi:
  - **DROP table [if exists] NomeElemento [RESTRICT | CASCADE]**
  - **La stessa sintassi può essere usata per effettuare il drop di schemi, domini, o viste usando le rispettive parole chiave schema, domain, view**
  - In MYSQL **[RESTRICT | CASCADE]** non fanno niente – si cancella la tabella e tutti i sui dati
  - **Attenzione al drop di tabelle esterne (set foreign_key_check = 0)**

Modifica dello schema

- Istruzione **ALTER TABLE**:
  - permette di inserire/rimuovere colonne dalle tabelle
  - inserire/rimuovere vincoli
- La sintassi è la seguente:
  - **ALTER TABLE NomeTabella {ADD [COLUMN] NomeAttributo def_attributo | DROP [COLUMN] NomeAttributo}**

Esempi:

- Inserimento della colonna Sesso nella Tabella Studenti:
  - **ALTER TABLE** Studenti **ADD COLUMN** Sesso char(1)),
- Cancellazione colonna AnnoIscrizione nella Tabella Studenti:
  - **ALTER TABLE** Studenti **DROP COLUMN** AnnoIscrizione;

Esempi: aggiungere/rimuovere una colonna

- Data la tabella:

```
CREATE TABLE `base_for` (
  `mat` varchar(45) NOT NULL,
`nome` varchar(45) DEFAULT NULL,
 PRIMARY KEY (`mat`),
 UNIQUE KEY `nome_UNIQUE` (`nome`)
);
```

- alter table base_for add column miacol varchar(10);
  - Aggiunge la riga miacol
- alter table base_for drop column miacol;
  - Rimuove la riga miacol
- **alter table base_for drop column nome;**
  - **Fallisce se nome è riferito da qualche altra tabella (anche la stessa)**

Modifica dello schema – gestione dei vincoli/I

- Data la tabella

```
CREATE TABLE provafor6 (
chiave varchar(45) NOT NULL,
 nome varchar(45)  ,
 PRIMARY KEY (chiave),
constraint fk foreign key (nome) REFERENCES base_for (nome)
);
```

Modifica dello schema - gestione dei vincoli/II

- Rimuovere il vincolo di foreign key

alter table provafor6 drop foreign key fk;

- Aggiungere un nuovo vincolo di foreingn key

alter table provafor6 add constraint fkk3 foreign key (nome) references base_for(nome);

Conviene usare la notazione constraint etc per rimuovere successivamente un vincolo.

NOTA: quando si aggiunge un vincolo alla tabella, questo deve essere soddisfatto dall'instanza altrimenti l'inserimento falisce

Modifica dello schema gestione dei vincoli/III

- Rimuovere il vincolo di primary key

```
alter table provafor6 drop primary key ;
```

- Aggiungere un nuovo vincolo di primary key

```
alter table provafor6 add primary key (chiave);
```

- Per la rimozione di altri vincoli (unique, check) usare

```
alter table nome_tabella drop key nome_vincolo;
```

o in alternativa

```
alter table nome_tabella drop index nome_vincolo;
```

Viste

- Sono **tabelle virtuali** il cui contenuto dipende dal contenuto di altre tabelle.
- In SQL si ottengono assegnando una lista di attributi ed un nome ad una interrogazione con select secondo la seguente sintassi:

CREATE [OR REPLACE] VIEW NomeVista [(ListaAttributi)] AS SelectSQL

- La select deve restituire un insieme di attributi compatibile con ListaAttributi. L'ordine di tali attributi deve coincidere con l'ordine degli attributi della vista

Esempio:

- Creare una vista impiegati costosi che hanno uno stipendio annuale maggiore di 30000 all'interno del database azienda:

```
CREATE VIEW ImpiegatiCostosi (Matricola,Nome,Cognome)
AS SELECT Matricola,Nome,Cognome
    FROM impiegati
    WHERE StipendioAnnuale > 30000
```

NOTA: la lista degli attributi può essere omessa. In tal caso lo schema è quello della select

Viste: utilizzo

- Una volta creata la vista questa può essere interrogata come una tabella normale:

  SELECT * FROM ImpiegatiCostosi;

- Altro esempio, creare una vista ImpiegatiRecenti per recuperare gli impiegati assunti dopo il 2005:

CREATE VIEW ImpiegatiRecenti
(Matricola,Nome,Cognome)

AS SELECT Matricola,Nome,Cognome

FROM impiegati

WHERE DataAssunzione > ‘2005-01-01’

esempi

- Creare la vista imp_milanesi, in cui sono contenuti matricola, nome, cognome e stipendio annuale dei dipendenti che lavorano a milano.

```
create view imp_milanesi (matricola, nome, cognome, stipendio_annuale)
as
 select emp.matricola, emp.nome, emp.cognome, emp.StipendioAnnuale
from impiegati1 as emp, reparti as rep
where emp.IdReparto=rep.IdReparto and rep.Citta='Milano';
```

esempi

- Trovare la media degli stipendi degli impiegati milanesi

```
select avg (stipendio_annuale)from imp_milanesi;
```

- Stampare il cognome degli impiegati milanesi di nome ‘mario’

```
select cognome
from imp_milanesi
where nome='mario';
```

Tipici utilizzi delle viste

- Sicurezza: impedire ad alcuni utenti di avere accesso completo ai dati di una tabella

- Convenienza:
  - Semplificare la scrittura delle query (vedi la precedente)
  - Quando una query compare più volte (impiegati milanesi)

- Indipendenza logica delle applicazioni rispetto all'organizzazione dei dati
  - Tramite le viste, se si cambia lo schema del DB, è possibile non dover cambiare le applicazioni

Per cancellare una view

- *DROP VIEW [IF EXISTS] viewname*

Il linguaggio SQL
costrutti DML per la modifica del contenuto della base di dati

foglia@iet.unipi.it

Data Manipulation Language

- Il **Data Manipulation Language** è l'insieme di istruzioni utilizzate per interrogare/modificare il contenuto della base di dati.
- Ne fanno parte le istruzioni di inserimento, cancellazione e modifica dei record (INSERT, DELETE, UPDATE):
  - **SELECT** esegue interrogazioni (query) sul DB
  - **INSERT INTO** permette di inserire nuove tuple nel DB
  - **DELETE** permette di cancellare tuple dal DB
  - **UPDATE** permette di modificare tuple del DB
- **INSERT INTO** può usare il risultato di una query per eseguire inserimenti multipli
- **DELETE** e **UPDATE** possono fare uso di condizioni per specificare le tuple da cancellare o modificare
- **In ogni caso gli aggiornamenti riguardano una sola tabella**

Inserimento

- È possibile inserire una nuova tupla specificandone i valori usando la seguente sintassi:
- **INSERT INTO Tabella [(Campo1, Campo2....)]**
- **VALUES (Val1, Val2,...)**

Esempio:

```
INSERT INTO impiegati2
(Matricola,Cognome,Nome,Mansione,IdReparto,StipendioAnnuale,PremioProduzione,DataAssunzione)
VALUES
( 10,'Celesti','Amelia','Analista',2,30000,0,'2018:02:22');
```

Inserimento

- Ci deve essere **corrispondenza** tra attributi e valori
  - A campo1 corrisponde val1
- La lista degli attributi si può omettere, nel qual caso per gli attributi vale l'ordine con cui sono stati definiti
  - La lista è costituita da tutti gli attributi

```
INSERT INTO impiegati2 VALUES
( 10,'Celesti','Amelia','Analista',2,30000,0,'2018-02-22');
```

- Se la lista degli attributi non include tutti gli attributi, i restanti assumono valore NULL (se ammesso) o il valore di default (se specificato)
- Se l'inserimento viola vincoli di chiave primaria, not null, foreign key etc, l'inserimento viene rifiutato.

Inserimento con select

- È possibile anche inserire le tuple che risultano da una query:

```
INSERT INTO ResidenzaPisa(ResidenzaPI,Matricola)
        SELECT CittaResidenza, Matricola
        FROM Studenti
        WHERE CittaResidenza = 'Pisa'
```

- Valgono le regole viste per il caso singolo

- Gli schemi del risultato e della tabella in cui si inseriscono le tuple possono essere diversi (avere nome di attributi diversi), l'importante è che i tipi delle colonne siano compatibili

Altro esempio

- Data la tebella

```
CREATE TABLE `impiegati_milanesi` (
 `matricola` int(11) NOT NULL,
 `nome` varchar(45) DEFAULT NULL,
 `cognome` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`matricola`));
```

- Popolare la tabella con nome, cognome e matricola degli impiegati che lavorano a milano

soluzione

```
insert into impiegati_milanesi(nome, cognome, matricola)
(select imp.nome, imp.cognome, imp.matricola
from impiegati2 as imp, reparti as rep
where imp.IdReparto = rep.idreparto
    and rep.Citta = 'milano');
```

Cancellazione

- L'istruzione **DELETE** cancella una tupla esistente
- Può fare uso di una condizione per specificare le tuple da cancellare attraverso la seguente sintassi:

DELETE FROM Tabella WHERE condizione

Esempio:

- Cancella uno studente con una matricola specifica:

DELETE FROM Studenti **WHERE** Matricola = 'A002038'

- Cancella gli studenti che non hanno fatto esami:

DELETE FROM Studenti

WHERE Matricola **NOT IN** (**SELECT** Matricola

FROM Esami

GROUP BY Matricola)

- NOTA: **Se la clausola WHERE non e' presente si eliminano tutti i record della tabella**
- **L'istruzione DELETE può portare a violare il vincolo di integrità referenziale**

Altri esempi

- Cancellare da impiegati milanesi gli impiegati di matricola 2

```
delete from impiegati_milanesi
where matricola = 2;
```

- Cancellare i reparti che non hanno dipendenti:

```
delete from reparti
where idreparto not in (select idreparto
                        from impiegati1) limit 10;
```

osservazioni

- Differenze fra
  - Delete from reparti
  - Drop table reparti

- La prima elimina tutte le righe da reparti, ed eventualmente righe con vincolo di foreign key in altre tabelle. Lo schema del DB resta immutato

- La seconda altera lo schema. Cancella i dati della tabella reparti, ma cancella anche la tabella ed eventuali viste che fanno riferimento ad essa

Modifica

- L'istruzione **UPDATE** modifica i valori delle colonne in una riga esistente
- Può fare uso di una condizione per specificare le tuple da modificare

UPDATE Tabella

SET <campo1>=<valore1>, <campo2>=<valore2> ....

WHERE condizione

Esempio:

Studenti(Matricola,Cognome,Nome,CittaResidenza,Media)

Esami(Matricola,Corso,Voto)

UPDATE Esami **SET** Voto = 30

WHERE Matricola = ' M0004' **AND** corso = 'Analisi I'

- L'istruzione UPDATE può portare a violare il vincolo di integrità referenziale, in tal caso l'inserimento fallisce.
- L'operatore = nel campo set ha il significato di assegnamento

Altri esempi

- Incrementare di 2 il premio di produzione di tutti gli impiegati

```
update impiegati2
set PremioProduzione = PremioProduzione +2 limit 20;
```

- Aumentare del 10% il premio di produzione degli impiegati del reparto sviluppo

```
update impiegati2
set PremioProduzione = PremioProduzione *1.1
 where idreparto in (select idreparto
                     from reparti
                     where nome= 'sviluppo' ) limit 20;
```

Data Control Language (DCL)

- Il **Data Contro Language** è un linguaggio che permette di fornire o revocare agli utenti i permessi necessari per poter utilizzare i comandi di DML e DDL oltre agli stessi comandi DCL.

- All'atto della creazione di un database l'utente creatore ne diventa il *proprietario*. Gli altri utenti possono leggere le informazioni ma non modificare il DB.

- Per modificare i diritti di accesso è possibile utilizzare i comandi:
  - **GRANT**
  - **REVOKE**

Data Control Language (DCL)

- Creazione di nuovi utenti

CREATE USER ‘Nome'@‘%' IDENTIFIED BY ‘password';

- Cancellazione di utenti

DROP USER ‘Nome’

Data Control Language (DCL)

Concessione di privilegi:

- Il comando Grant fornisce uno o più permessi ad un determinato utente su un determinato oggetto del database. Il comando ha la seguente sintassi:

GRANT ALL PRIVILEGES ON Impiegati

TO User1 [**WITH GRANT OPTION**]

- **WITH GRANT OPTION** specifica se il privilegio può essere trasmesso ad altri utenti

Revoca di privilegi:

- Il comando Revoke revoca uno o più permessi ad un determinato utente su un determinato tipo di oggetti.
- Un utente può revocare solo privilegi che lui ha concesso

REVOKE ALL PRIVILEGES ON Impiegati **FROM** User1

Da refs pag. 726

- In SQL statements such as CREATE USER, GRANT, and SET PASSWORD, account names follow these rules:
- • Account name syntax is '*user_name*'@'*host_name*'.
- • An account name consisting only of a user name is equivalent to '*user_name*'@'%'. For example, 'me' is equivalent to 'me'@'%'.
- • The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes are necessary to specify a *user_name* string containing special characters (such as space or -), or a *host_name* string containing special characters or wildcard characters (such as . or %); for example, 'test-user'@'%.com'.
- The user name and host name parts, if quoted, must be quoted separately. That is, write 'me'@'localhost', not 'me@localhost'; the latter is actually equivalent to 'me@localhost'@'%'.

Da refs pag. 1514

- CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY '*password*';
- The host name part of the account name, if omitted, defaults to '%'.
- An account when first created has no privileges. To assign privileges, use the GRANT statement.