SQL

6 lezione
manipolazione dei dati

Francesca Gasparini
gasparini@disco.unimib.it

Operazioni di aggiornamento

- **INSERT** inserisce nuove tuple nel DB
- **DELETE** cancella tuple dal DB
- **UPDATE** modifica tuple del DB

•Tutte le istruzioni possono operare su un insieme di tuple (set-oriented).

•In ogni caso gli aggiornamenti riguardano una sola relazione sulla base di una condizione che può coinvolgere anche altre relazioni.

Operazioni di aggiornamento

- **INSERT può usare il** risultato di una query **per eseguire inserimenti multipli**
- **DELETE e UPDATE possono fare** uso di condizioni **(where) per specificare le tuple da cancellare o modificare**

Insert

INSERT INTO Tabella [(Attributi)]
VALUES (Valori)

permette di inserire *singole* righe all'interno delle tabelle
L'argomento della clausola *values* rappresenta i valori da inserire

oppure

INSERT INTO Tabella [(Attributi)]
SELECT ...
permette di inserire gruppi di tuple selezionate da un'altra relazione

Insert-Values

È possibile inserire una nuova tupla specificandone i valori

INSERT INTO Sedi (Sede, Responsabile, Citta)
VALUES ('S04', 'Bruni', 'Firenze')

INSERT INTO Sedi (Sede, Citta) --- sede senza responsabile
VALUES ('S04','Firenze')

Insert-Values

Persone | Nome | Età | Reddito

Maternità | Madre | Figlio

Paternità | Padre | Figlio

INSERT INTO Persone VALUES ('Mario', 25, 52)

INSERT INTO Persone (Nome, Eta, Reddito)
VALUES ('Pino', 25, 52)

INSERT INTO Persone (Nome, Reddito)
VALUES ('Lino', 55)

Insert

- l'ordinamento degli attributi (se presenti) e dei valori è significativo
- le due liste debbono avere lo stesso numero di elementi
- se la lista di attributi è omessa, si fa riferimento a tutti gli attributi della relazione, secondo l'ordine con cui sono stati definiti
- se la lista di attributi non contiene tutti gli attributi della relazione, per gli altri viene inserito un valore nullo (che deve essere permesso) o un valore di default

Insert-Select

E' possibile aggiungere insiemi di righe estratti dal contenuto della base di dati ad es:

PRODOTTO (Codice, Descrizione, LuogoProd, Prezzo)
PRODOTTIMILANESI (Codice, Descrizione)

Inserire nella tabella PRODOTTIMILANESI quelli prodotti a Milano

```
INSERT INTO PRODOTTIMILANESI (Codice, Descrizione)
(SELECT Codice, Descrizione
FROM PRODOTTO
WHERE LuogoProd="Milano")
```

Insert-Select

Inserire nella tabella persone i padri dalla tabella paternità, se non sono già presenti

Persone | Nome | Età | Reddito

Maternità | Madre | Figlio

Paternità | Padre | Figlio

```
INSERT INTO Persone (Nome)
(SELECT Padre
FROM Paternita
WHERE Padre NOT IN (SELECT Nome
                    FROM Persone))
```

Insert

- si vuole creare una relazione Promozioni, che contiene alcune delle colonne della relazione Impiegati: Nome, Sipendio, Premio_P
- si vuole inserire in questa relazione tutti gli ingegneri il cui premio di produzione è superiore al 25% del loro stipendio; le informazioni sugli ingegneri devono essere estratte dalla relazione Impiegati

```
INSERT INTO Promozioni (Nome, Sipendio, Premio_P)
(SELECT Nome, Sipendio, Premio_P
FROM Impiegati WHERE Premio_P > 0.25*Stipendio AND
   Mansione = 'ingegnere')
```

Delete

DELETE: comando di eliminazione delle righe dalle tabelle.

```
DELETE FROM Tabella
      [ WHERE Condizione ]
```

se non è specificata la clausola WHERE vengono eliminate tutte le righe della tabella

rispetta la sintassi di select, quindi ci possono essere nella clausola where delle interrogazioni nidificate

Delete

Persone | Nome | Età | Reddito

Maternità | Madre | Figlio

Paternità | Padre | Figlio

```
DELETE FROM Persone
WHERE Eta < 35

DELETE FROM Paternita
WHERE Figlio NOT in (SELECT Nome
    FROM Persone)

DELETE FROM Paternita
```

Delete

- elimina le ennuple che soddisfano la condizione
- può causare (se i vincoli di integrità referenziale sono definiti con politiche di reazione cascade) eliminazioni da altre relazioni
- ricordare: se la where viene omessa, si intende where true

Delete

esempio: eliminare le sedi di Bologna dalla tabella Sedi

```
DELETE FROM Sedi
WHERE Citta = 'Bologna'
```

Che succede se la cancellazione porta a violare il vincolo di integrità referenziale? (ad es.: che accade agli impiegati delle sedi di Bologna?)

Vincoli interrelazionali : Il problema delle violazioni

- si possono introdurre violazioni operando sulle righe della tabella padre (tabella esterna) o sulle righe della tabella figlio (tabella interna)
- modifiche sulla tabella interna (figlio):
  - inserimento di una nuova riga
  - modifica della foreign key

  non vengono proposte reazioni, solo il rifiuto dell'operazione
- modifiche sulla tabella esterna (padre):
  - cancellazione di una riga
  - modifica dell'attributo riferito

  vengono proposte diverse reazioni

Aggiornamento

• Le violazioni possono essere introdotte
1. da aggiornamenti (update) dell'attributo cui si fa riferimento
2. da cancellazioni di tuple

• Reazioni previste:
– cascade : propaga la modifica
– set null : annulla l'attributo cui fa riferimento
– set default : assegna il valore di default all'attributo
– no action : impedisce che la modifica possa avvenire

Vincoli interrelazionali : Esempio

```
CREATE TABLE Esame (
Matr        CHAR(6),
CodCorso    CHAR(6),
Data        DATE        NOT NULL,
Voto        Voto,
PRIMARY KEY (Matr, CodCorso)
FOREIGN KEY (Matr) REFERENCES Studente
        ON DELETE CASCADE
        ON UPDATE CASCADE
FOREIGN KEY (CodCorso) REFERENCES Corso
        ON DELETE NO ACTION
        ON UPDATE CASCADE
 )
```

Delete e Drop

Per cancellare tutte le tuple da STUDENTE (mantenendo lo schema della tabella):

```
delete from Studente
```

Per cancellare completamente la tabella STUDENTE (contenuto e schema):

```
drop table Studente cascade
```

elimina anche tutte le tabelle o viste che ad essa fanno riferimento

```
drop table Studente restrict
```

opzione di default, non è eseguito in presenza di oggetti non vuoti, se appare in qualche altra relazione (es . dominio, vista..)

Modifiche degli schemi

• **drop: cancella oggetti DDL**

si applica su domini, tabelle, indici, view, asserzioni, procedure,...

es.: **drop table Ordine**
es.: **drop domain MyDomain**

• Opzioni restrict e cascade

– **restrict** : impedisce drop se gli oggetti comprendono istanze
– **cascade** : applica drop agli oggetti collegati

Update

Aggiornamento di uno o più attributi

```
UPDATE NomeTabella
SET  Attributo = < Espressione |SELECTSQL|NULL | DEFAULT >
   {,Attributo = < Espressione |SELECTSQL |NULL | DEFAULT >}
          [ WHERE Condizione ]
```

se non è presente clausola where, si suppone WHERE TRUE e quindi si opera la modifica su tutte le righe

Update

l'istruzione UPDATE può fare uso di una condizione (per specificare le tuple da modificare) e di espressioni (per determinare i nuovi valori)

```
UPDATE Sedi
SET Responsabile = 'Bruni', Citta = 'Firenze'
WHERE Sede = 'S01'
```

```
UPDATE Imp
SET Stipendio = 1.1 * Stipendio
WHERE Ruolo = 'Programmatore'
```

Anche l'UPDATE può portare a violare il vincolo di integrità referenziale

Update

esempio: si vuole promuovere Gianni a dirigente e contemporaneamente aumentare il suo stipendio del 10%

```
UPDATE Impiegati
SET Mansione = 'dirigente',
      Stipendio = 1.10*Stipendio
WHERE Nome = 'Gianni';
```

Update nota:

si vuole aumentare lo stipendio di Rossi del 10%

```
UPDATE Impiegati
SET  Stipendio = 1.10*Stipendio
WHERE Cognome = 'Rossi';
```

Operiamo su una riga

si vuole aumentare lo stipendio degli impiegati del 10%

```
UPDATE Impiegati
SET  Stipendio = 1.10*Stipendio
```

Operiamo su tutte le righe

si vuole aumentare lo stipendio degli impiegati dell'amministrazione del 10%

```
UPDATE Impiegati
SET  Stipendio = 1.10*Stipendio
WHERE Dipart = 'Amministrazione ';
```

Operiamo su alcune righe

Update

le subqueries possono essere usate per

•(a) determinare le tuple da modificare
•(b) determinare i nuovi valori da assegnare alle tuple

esempio (a): si consideri la relazione Bonus e si supponga di voler aumentare del 5% lo stipendio di tutti gli impiegati presenti in tale relazione

```
UPDATE Impiegati
SET Stipendio = 1.05*Stipendio
WHERE codice IN
(SELECT codice FROM Bonus)
```

Update

- esempio (b): si vuole assegnare ai tecnici uno stipendio pari al 110% della media degli stipendi dei tecnici

```
UPDATE Impiegati
SET Stipendio = (SELECT  1.1*AVG(Stipendio)
                 FROM Impiegati
                 WHERE Mansione = 'tecnico')
WHERE mansione='tecnico';
```

- le modifiche in SQL sono eseguite in modo set-oriented: la clausola WHERE e il valore dell'espressione SET vengono valutati un'unica volta, poi gli aggiornamenti vengono effettuati su tutte le tuple "contemporaneamente"

Update

Aumentare del 10% gli stipendi di coloro che guadagnano meno di 30 e del 15% quelli che guadagnano più di 30

```
update Impiegato
set Stipendio = Stipendio * 1.1
where Stipendio <= 30

update Impiegato
set Stipendio = Stipendio * 1.15
where Stipendio > 30
```

Se i comandi sono scritti in questo ordine, alcuni impiegati possono ottenere un aumento doppio

Update

Poiché il linguaggio è set-oriented, è molto importante l'ordine dei comandi. in questo caso semplice basta invertire.

```
update Impiegato
set Stipendio = Stipendio * 1.15
where Stipendio > 30

update Impiegato
set Stipendio = Stipendio * 1.1
where Stipendio <= 30
```

a volte più complicato e richiede aggiornamenti intermedi, oppure l'uso del costrutto CASE o l'impiego di un programma di alto livello tradizionale

SQL
caratteristiche evolute

Vincoli di integrità generici: check

La clausola check può essere usata per esprimere vincoli arbitrari **nella definizione dello schema**

check (condizione)

le condizioni che si possono usare sono quelle che si possono usare per la clausola WHERE: cioè un predicato o una combinazione booleana di predicati.

Vincoli di integrità generici: check

Mansione Char(10) CHECK (Mansione IN ('dirigente', 'ingegnere','tecnico', 'segretaria'))

tale condizione può contenere sottointerrogazioni che fanno riferimento ad altre tabelle, ma il vincolo viene controllato solo quando viene modificato il valore dell'attributo a cui è associato

Vincoli di integrità generici: check

```
Stipendio Decimal (7,2) CHECK (Stipendio <=
        (SELECT MAX(Stipendio)
        FROM Impiegato
        WHERE Mansione = 'dirigente'))
```

è un vincolo CHECK corretto, ma viene controllato solo sulla tupla che **sto aggiornando** ⇒ se diminuisco lo stipendio di un dirigente posso trovarmi in uno stato in cui un impiegato guadagna più di ogni dirigente.

i vincoli sui domini sono del tutto analoghi

```
CREATE DOMAIN DominioMansione AS Char(10) CHECK
(VALUE IN ('dirigente', 'ingegnere','tecnico', 'segretaria'))
```

Check, esempio

indicare un vincolo che obblighi l'impiegato ad avere stipendio inferiore al suo superiore

```
create table Impiegato
(
Matricola character(6),
Cognome character(20),
Nome character(20),
Dipart character(15)
Sesso character not null check (sesso in ('M','F'))
Stipendio integer,
Superiore character(6),
check (Stipendio <= (select Stipendio
        from Impiegato J
        where Superiore = J.Matricola)
)
```

Check, esempio

indicare un vincolo che obbliga l'impiegato ad avere un superiore nel proprio dipartimento a meno che la sua matricola non cominci con 1

```
create table Impiegato
(
Matricola character(6),
Cognome character(20),
Nome character(20),
Dipart character(15)
Sesso character not null check (sesso in ('M','F'))
Stipendio integer,
Superiore character(6),
check (Matricola like "1%" or Dipart  = (select Dipart
        from Impiegato I
        where Superiore = I.Matricola)
)
```

Vincoli di integrità generici: asserzioni

ASSERZIONI: sono elementi dello schema, servono per scrivere vincoli che coinvolgono più tuple o più tabelle

Utili in molte situazioni (es., per esprimere vincoli interrelazionali di tipo generico)

Una asserzione associa un nome a una clausola check;

create assertion *NomeAsserzione* **check** (*Condizione*)

la condizione è un predicato o una combinazione booleana di predicati (componente WHERE di una query SQL)

Vincoli di integrità generici: asserzioni

•Specifica vincoli a livello di schema

```
create assertion NomeAss check ( Condizione )
```

Esempio: imporre che ci sia nella tabella IMPIEGATO sempre almeno una riga, cioè almeno un impiegato

```
create assertion AlmenoUnImpiegato
    check (1 <= (select count(*)
                from Impiegato ))
```

Vincoli di integrità generici: asserzioni

vincolo per cui ci sia un solo dirigente per dipartimento

```
CREATE ASSERTION UnSoloDirigentePerDipartimento
CHECK (NOT EXISTS (SELECT * FROM Impiegati
WHERE Mansione = 'dirigente'
GROUP BY dipart
HAVING COUNT(*) > 1))
```

non tutti i DBMS prevedono tutti i tipi di vincoli, in particolare le asserzioni non sono generalmente supportate

quasi tutti i DBMS si limitano a gestire i vincoli che possono essere verificati esaminando una sola tupla
motivazione: efficienza della valutazione

Politica controllo dei vincoli tramite check o asserzione

La verifica dei vincoli può essere:

a) immediate (immediata): è verificata immediatamente dopo ogni modifica della base di dati. La loro violazione annulla l'ultima modifica *(rollback parziale)*

b) deferred (differita): verificata dopo una serie di operazioni che costituiscono una transazione. la loro violazione annulla l'intera applicazione *(rollback)*

controllo dei vincoli

Ogni vincolo è definito di un tipo (normalmente "immediate")

L'applicazione può modificare il tipo iniziale dei vincoli:

- set constraints immediate
- set constraints deferred

Tutti i vincoli vengono comunque verificati, prima o poi

Politica controllo dei vincoli tramite check o asserzione

verifica differita:

introdotta per gestire situazioni in cui una singola modifica della base di dati non è ammissibile

ESEMPIO:in caso di vincoli di integrità incrociati su più tabelle, le operazioni di inserimento per modificare lo stato iniziale vuoto non sarebbero ammissibili, i caso di verifica di vincoli immediata.

controllo dei vincoli

```
CREATE TABLE Impiegato (
Matricola      CHAR(20),
Cognome        CHAR(20),
Dipart         CHAR(20),   NOT NULL,
FOREIGN KEY (Dipart) REFERENCES Dipartimento (nome)
)
CREATE TABLE Dipartimento (
Nome           CHAR(20),
Direttore      CHAR(20),   NOT NULL,
FOREIGN KEY (Direttore) REFERENCES
               Impiegato (Matricola)
)
```

controllo dei vincoli

- i vincoli NOT NULL, UNIQUE, PRIMARY KEY sono verificati immediatamente e la loro violazione porta ad un rollback parziale
- quando il controllo è differito non si sa chi ha causato la violazione, viene annullata l'intera transazione

Viste

in SQL è possibile definire viste alternative degli stessi dati.

- una vista (**view**) è una relazione virtuale attraverso cui è possibile vedere i dati memorizzati nelle relazioni reali (dette **di base**)
- una vista non contiene tuple, ma può essere usata quasi a tutti gli effetti come una relazione di base
- una vista è definita da una interrogazione su una o più relazioni di base o altre viste
- una vista è materializzata eseguendo l'interrogazione che la definisce

Viste

il meccanismo delle viste è utile per

– semplificare l'accesso ai dati
– garantire la privatezza dei dati

Si definiscono associando un nome ed una lista di attributi al risultato di un'interrogazione.

il comando di creazione di viste ha il seguente formato

create view V [(ListaAttributi)] as query
[with [local | cascaded] check option]

Viste

V è il nome della vista che viene creata; tale nome deve essere unico rispetto a tutti i nomi di relazioni e di viste definite dallo stesso utente che definisce V

Query è l'*interrogazione di definizione* della vista

una vista ha lo stesso numero di colonne pari alle colonne (di base o virtuali) specificate nella clausola di proiezione di Query

ListaAttributi è una lista di nomi da assegnare alle colonne della vista; tale specifica non è obbligatoria, tranne nel caso in cui l'interrogazione contenga nella clausola di proiezione funzioni di gruppo e/o espressioni

Viste

esempio: si vuole creare una vista costituita da un sottoinsieme delle tuple della relazione Impiegati; più precisamente la vista deve elencare le colonne Codice, Nome e Mansione degli impiegati del dipartimento 'produzione'

```
CREATE VIEW Imp10 AS
SELECT Codice, Nome, Mansione
FROM Impiegati WHERE dipart='produzione';
```

i nomi delle colonne della vista Imp10 sono rispettivamente: Codice, Nome, Mansione

su una vista si possono eseguire (con alcune importanti restrizioni) sia interrogazioni che modifiche

Viste

Uso di join

può essere facile per alcuni utenti lavorare con una sola relazione piuttosto che eseguire join tra relazioni diverse

esempio: si vuole creare una vista Personale che contiene le colonne Nome e Mansione della relazione Impiegati e il nome del progetto a cui ogni impiegato lavora

```
CREATE VIEW Personale AS
SELECT Nome, Mansione, Pnome
FROM Impiegati, Progetti
WHERE Impiegati.ProgNo = Progetti.ProgNo;
```

Viste

Uso di espressioni e funzioni

è possibile definire viste tramite interrogazioni che contengono espressioni o funzioni

queste espressioni appaiono del tutto simili alle altre colonne della vista, ma il loro valore è calcolato dalle relazioni di base ogni volta che la vista è materializzata

tali colonne sono spesso chiamate colonne virtuali; è obbligatorio specificare un nome nella vista per tali colonne

Viste

Uso di espressioni e funzioni

esempio: si vuole definire una vista che contenga lo stipendio annuale degli impiegati

```
CREATE VIEW V1
(Nome,Stipendio_Mensile,Stipendio_Annuale, Dip#) AS
SELECT Nome, Stipendio, Stipendio*12, Dip#
FROM Impiegati;
```

Viste

quando si specifica un'interrogazione su una vista, il sistema sostituisce nell'interrogazione la vista con la sua definizione

esempio: si consideri la vista Personale e la query

```
SELECT Nome, Pnome FROM Personale
WHERE Mansione = 'dirigente';
```

la query che si ottiene dopo la composizione è la seguente

```
SELECT Nome, Pnome
FROM Impiegati, Progetti
WHERE Impiegati.ProgNo = Progetti.ProgNo
AND Mansione = 'dirigente';
```

Viste: aggiornamento

Data la complessità del problema, di fatto ogni DBMS pone dei **limiti** su quelle che sono le viste aggiornabili

Le più comuni restrizioni riguardano la non aggiornabilità di viste in cui il blocco più esterno della query di definizione contiene:

- GROUP BY
- Funzioni aggregate
- DISTINCT
- join (espliciti o impliciti)

Viste: aggiornamento

La precisazione che è il blocco più esterno della query di definizione che non deve contenere, ad es., dei join ha importanti conseguenze. Ad esempio, la seguente vista non è aggiornabile

```
CREATE VIEW ImpBO (CodImp, Nome, Sede, Ruolo, Stipendio)
        AS SELECT I.*
        FROM Imp I JOIN Sedi S ON (I.Sede = S.Sede)
        WHERE S.Citta = 'Bologna'
```

mentre lo è questa, di fatto equivalente alla prima

```
CREATE VIEW ImpBO (CodImp, Nome, Sede, Ruolo, Stipendio)
        AS SELECT I.*
        FROM Imp I
        WHERE I.Sede IN (SELECT S.Sede FROM Sedi S
                WHERE S.Citta = 'Bologna')
```

Viste: aggiornamento

Check option:

I sistemi commerciali considerano aggiornabile una vista solo se e' definita da una **sola** tabella.

qualche sistema richiede che fra gli attributi vi sia una chiave primaria della tabella.

La check option specifica che **possono essere ammessi aggiornamenti solo sulle righe della vista** e dopo gli aggiornamenti le righe **devono** continuare ad appartenere alla vista.

può accadere quando si aggiorna con un valore che rende falso uno dei predicati di selezione

Viste: aggiornamento

Si consideri il seguente inserimento nella vista ImpBO

```
INSERT INTO ImpBO (CodImp, Nome, Sede, Ruolo, Stipendio)
VALUES ('E009','Azzurri','S03','Analista',1800)
```

in cui il valore di Sede ('S03'--> Milano) non rispetta la specifica della vista. Ciò comporta che una successiva query su ImpBO non restituirebbe la tupla appena inserita

Quindi non e' inseribile se si usa check option

```
CREATE VIEW ImpBO (CodImp, Nome, Sede, Ruolo, Stipendio)
        AS SELECT I.*
        FROM Imp I
        WHERE I.Sede IN (SELECT S.Sede FROM Sedi S
                WHERE S.Citta = 'Bologna')
```

Viste: aggiornamento

Nel caso in cui la vista sia definita in termini di altre viste, check option puo' essere:

cascaded (default): la violazione viene controllata a tutti i livelli

local: la violazione viene controllata solo sulla vista piu' esterna

Viste: aggiornamento

Se la vista V1 è definita in termini di un'altra vista V2, e si specifica la clausola **WITH CHECK OPTION**, il DBMS verifica che la nuova tupla t inserita soddisfi sia la definizione di V1 che quella di V2, indipendentemente dal fatto che V2 sia stata a sua volta definita **WITH CHECK OPTION**

Questo comportamento di default, che è equivalente a definire V1
WITH CASCADED CHECK OPTION

si può alterare definendo V1

WITH LOCAL CHECK OPTION
In modalità **LOCAL**, il DBMS verifica solo che t soddisfi la specifica di V1 e quelle di tutte e sole le viste da cui V1 dipende per cui è stata specificata la clausola **WITH CHECK OPTION**

Viste - Esempio

```
create view NomeVista [ ( ListaAttributi ) ] as SelectSQL
[ with [ local | cascaded ] check option ]

create view ImpiegatiAmmin
      (Matricola, Nome, Cognome, Stipendio) as
    select Matricola, Nome, Cognome, Stipendio
    from Impiegato
    where Dipart = 'Amministrazione' and
        Stipendio > 10
```

Viste - Esempio

```
create view ImpiegatiAmminPoveri as
    select *
    from ImpiegatiAmmin
    where Stipendio < 50
    with check option
```

Un assegnazione dello stipendio a 8 milioni non e' accettato, ma sarebbe accettato con local check option.

Uno stipendio a 60 milioni non sarebbe accettato, neanche con local check option

Ancora sulle viste

•Estrarre numero medio di uffici distinti per ogni dipartimento

```
select avg (count (distinct Ufficio))
from Impiegato
group by Dipart
```

E' un'interrogazione scorretta perché SQL non ammette in cascata più operatori aggregati. infatti la valutazione dei due diversi operatori avviene a diversi livelli di aggregazione, ma è ammessa una sola occorrenza della clausola group by.

Ancora sulle viste

- Interrogazione scorretta

```
select avg (count (distinct Ufficio))
from Impiegato
group by Dipart
```

- Con una vista

```
create view DipartUffici(NomeDip,NroUffici) as
        select Dipart, count(distinct Ufficio)
        from Impiegato
        group by Dipart;

select avg(NroUffici)
from DipartUffici
```

Controllo degli accessi

- SQL presuppone che l'utente sia identificato in modo univoco dal sistema
- Per l'identificazione il DBMS può sfruttare il sistema operativo sottostante o fornire un proprio meccanismo di identificazione
- Il DBMS permette di proteggere tutte le risorse: tabelle, viste, procedure
- Ad ogni risorsa sono associati una lista di privilegi
- Il creatore della risorsa possiede tutti i privilegi

Risorse e privilegi

- I privilegi principali sono:
  - insert
    inserimento di dati (tabelle e attributi)
  - update
    modifica di dati (tabelle e attributi)
  - delete
    cancellazione dati (tabelle e viste)
  - select
    leggere la risorsa (solo tabelle)

Il comando grant

- Il comando grant permette di concedere i privilegi

Dare il privilegio di inserimento nella tabelle studenti a pippo

grant insert on studenti **to** pippo

Concedere a pippo e pluto tutti i privilegi sulla tabella studenti

grant all privileges on studenti **to** pippo, pluto

grant option

- Con with grant option si concede anche il diritto di amministrare il privilegio

Dare a pippo la possibilità di decidere chi può inserire dati nella tabella studenti

grant insert on studenti **to** pippo **with grant option**

Il comando revoke

- I privilegi possono essere revocati solo da chi li ha concessi
- Si può causare una cascata di rimozioni
- Non è necessario rimuovere tutti i privilegi concessi

Togliere a pippo il privilegio di selezione e tutti i privilegi connessi

revoke select on studenti **from** pippo **cascade**

Concedere a pippo e pluto tutti i privilegi sulla tabella studenti

Grant all privileges on studenti **to** pippo, pluto

revoke insert on studenti **from** pluto