Basi di Dati – Antonio Vivace

Linguaggio SQL

Descrizione

Useremo questa notazione per definire la sintassi di SQL:

`< x >`	Usate per isolare il termine *x*
`[ x ]`	Indicano che termine *x* è opzionale
`{ x }`	Indicano che termine *x* può essere ripetuto 0 volte od un numero arbitrario di volte
`|`	Separa opzioni alternative Es: *x* | *y* indica che i termini *x* ed *y* sono alternativi

Le parentesi tonde ( ) appartengono al linguaggio SQL e non alla notazione sopra descritta.

Domini

`BIT [varying][(lunghezza)]` `VARBIT (lunghezza)`	Se la lunghezza non è specificata corrisponde ad 1 singolo valore. Corrisponde ad attributi che possono assumere solo due valori (0,1).	

`CHARACTER [varying] [(lunghezza)]`
`CHARACTER o CHAR`
`VARCHAR(lunghezza)`
	Rappresenta singoli caratteri alfanumerici oppure stringhe di lunghezza fissa o variabile.	

`INTEGER` `SMALLINT`	Rappresentano valori interi	La precisione (numero totale di cifre) varia a seconda della specifica implementazione di SQL, non è specificata nello standard. SMALLINT richiede minore spazio di memorizzazione.

`NUMERIC`
`NUMERIC (precisione)`
`NUMERIC (precisione,scala)`
	Rappresentano i valori decimali	La differenza tra NUMERIC e DECIMAL è che il primo deve essere implementato esattamente con la precisione richiesta, mentre il secondo può avere una precisione maggiore, la precisione segnalata è requisito minimo. Se la precisione non è specificata si usa il valore caratteristico dell'implementazione. Per la scala si usa valore 0.

`DECIMAL`
`DECIMAL(precisione)`
`DECIMAL(precisione,scala)`		

`FLOAT` `FLOAT (precisione)`	Rappresentano valori reali	Permette di richiedere la precisione che si desidera (lunghezza della mantissa). L'esponente dipende dall'implementazione.

`DOUBLE PRECISION` `REAL`	approssimati	Rappresentano valori a singola/doppia precisione in virgola mobile. La precisione (lunghezza della mantissa) dipende dalla specifica implementazione di SQL.
`DATE`	Rappresenta le date espresse come anno (4 cifre), mese (2 cifre), giorno (2 cifre). **`DATE 'yyyy-mm-dd'`**	

`TIME`
`TIME(`*`precisione`*`)`
`TIMESTAMP`
`TIMESTAMP(`*`precisione`*`)`
	Rappresenta i tempi espressi come ora (2 cifre), minuto (2 cifre) e secondo (2 cifre), è possibile con precision stabilire il numero di cifre decimali per le frazioni di secondo .
	`TIME 'hh:mm:ss[.[nnnnnn]] [TimeZone]'`
	Dove TimeZone è espresso nel formato {+|-} hh:mm e rappresenta la differenza di fuso fra ora locale e ora universale (UTC).
	Es: `TIME 21:03:04 +1` e **`TIME 20:03:04 +0`**	

`INTERVAL` *`PrimaUnitàDiTempo`* `INTERVAL` *`PrimaUnitàDiTempo`* `TO` *`UltimaUnitàDitempo`*		Esistono interval anni e mesi, oppure giorni e ore, ma non mesi e giorni. E' possibile specificare per la prima unità qualunque essa sia una precisione. **`Interval year(3) to month`** *Intervalli fino a 999 yr e 11 months* Nel caso la seconda unità siano secondi, si può specificare il num di cifre decimali dopo la virgola. **`Interval day(4) to second(3)`** *Intervalli fino a 9999 gg 23 hr 59 min e 59,999 sec*
`BLOB [ ( length [{K |M | G}] ) ]` `CLOB [ ( length [{K |M | G}] ) ]`	Binary Large Object (BLOB) e Character Large Object (CLOB) Permettono di includere direttamente nel database oggetti molto grandi. Un CLOB ed un BLOB senza la lunghezza specificata è di default a due giga characters/bytes. **`fotografia BLOB(10M)`** **`descrizione CLOB(100k)`** NOTA: Il sistema garantisce solo di memorizzarne il valore. Non possono essere usati come criterio di selezione per le interrogazioni.	

Creazione di Domini

Partendo dai domini predefiniti è possibile costrure nuovi domini tramite la primitiva create domain.

La definizione di *nuovi domini* è utile perché permette di associare dei vincoli a un nome di dominio: questo è importante quando si deve ripetere la stessa definizione di attributo su diverse tabelle.

```
CREATE DOMAIN NomeDominio AS DominioElementare
        [ ValoreDefault ] [ Vincoli ]

CREATE DOMAIN Voto AS SMALLINT
        DEFAULT 0
        NOT NULL

CREATE DOMAIN Voto AS SMALLINT
        DEFAULT 0
        CHECK (Voto>=18 AND Voto<=30)
```

Vincoli (Constraits)

Un vincolo è una regola che specifica delle condizioni sui valori di un elemento dello schema del database. Un vincolo può essere associato ad una tabella, ad un attributo, ad un dominio.
I vincoli possono essere di due tipi:
Intrarelazionali: si applicano all'interno di una relazione:

- `NOT NULL` Il valore deve essere non nullo
- `UNIQUE` I valori devono essere non ripetuti
- `PRIMARY KEY` Chiave primaria
- `CHECK` Condizioni complesse

In `UNIQUE` il valore null può comparire su diverse righe perché si assume che i valori null siano tutti diversi fra loro
Il vincolo `PRIMARY KEY` può essere definito una sola volta all'interno della relazione.

Interrelazionali: si applicano tra relazioni diverse :

- `REFERENCES` Permettono di definire vincoli di integrità
- `FOREIGN KEY` Referenziale
- `CHECK` Vincoli complessi

Alcuni attributi della tabella figlio sono definiti `FOREIGN KEY` e si devono riferire (`REFERENCES`) ad alcuni attributi della tabella padre che costituiscono una chiave (devono essere `UNIQUE` e `NOT NULL` oppure `PRIMARY KEY`). Se si omettono gli attributi destinazione, vengono assunti quelli della chiave primaria.

Vincoli interrelazionali : Reazioni alle violazioni

Le reazioni operano sulla tabella figlio (es Esami), in seguito a modifiche alla tabella padre (es Studente).

Reazioni previste:

`CASCADE`	propaga la modifica;
`SET NULL`	annulla l'attributo che fa riferimento;
`SET DEFAULT`	assegna il valore di default all'attributo;
`NO ACTION`	impedisce che la modifica possa avvenire.

Le violazioni possono essere introdotte :

- Da modifica (UPDATE) dell'attributo cui si fa riferimento;
- Da cancellazioni di tuple.

Creare uno SCHEMA

```
CREATE SCHEMA [ NomeSchema ]
        [[ authorization ] Autorizzazione ]
```

```
        { DefinizioneElementoSchema }
```

Creare una TABLE

```
CREATE TABLE NomeTabella (
        NomeAttributo Dominio [ ValoreDidefault ] [ Vincoli ]
        {,NomeAttributo Dominio [ ValoreDiDefault ] [ Vincoli ] }
        [ AltriVincoli ] )
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
)

CREATE TABLE Impiegato (
        Matricola       CHAR(6) PRIMARY KEY,
        Nome            CHAR(20)        NOT NULL,
        Cognome CHAR(20)        NOT NULL,
        Dipart          CHAR(15)        REFERENCES Dipartimento(NomeDip) ,
        Stipendio       NUMERIC(9)      DEFAULT 0,
        UNIQUE (Cognome,Nome)
)
```

Modifiche degli Schemi

```
ALTER DOMAIN NomeDominio <
        SET     DEFAULT ValoreDefault |
        DROP    DEFAULT |
        ADD     CONSTRAINT DefVincolo |
        DROP    CONSTRAINT NomeVincolo >

ALTER TABLE NomeTabella <
        ALTER COLUMN NomeAttributo <
                SET DEFAULT NuovoDefault |
                DROP DEFAULT > |
        DROP COLUMN NomeAttributo |
        ADD COLUMN DefAttributo |
        DROP CONSTRAINT NomeVincolo
        ADD CONSTRAINT DefVincolo >

DROP < schema, domain, table, view, ...> NomeElemento
        [ RESTRICT | CASCADE ]
```

Opzioni:

`RESTRICT` Impedisce drop se gli oggetti comprendono istanze non vuote.
`CASCADE` Applica drop agli oggetti collegati. Potenziale pericolosa reazione a catena.

Interrogazione

Interrogazioni Semplici

```
SELECT ListAttributi
FROM ListaTabelle
[WHERE Condizione]
```

Interrogazione Base: vengono selezionate, tra le righe che appartengono al prodotto cartesiano delle tabelle elencate nella clausola `FROM`, quelle che soddisfano le condizioni espresse nella clausola `WHERE`. Le colonne che vengono visualizzate sono quelle ottenute dalla valutazione delle espressioni che appaiono nella clausola `SELECT`.
SELECT Specifica gli elementi dello schema della tabella risultato. Come argomento può apparire * e rappresenta la selezione di tutti gli attributi delle tabelle elencate nella clausola FROM.
FROM L'interrogazione avviene sul prodotto cartesiano degli elementi dell'insieme che viene specificato in FROM.
WHERE Questa clausola ammette come argomento un'espressione booleana costruita combinando predicati semplici (che utilizzano gli operatori =, <>, <, >, <=,>=) con gli operatori and, or e not. E' disponibile anche l'operatore `LIKE` che permette di effettuare confronti con stringhe in cui compaiono i caratteri speciali _ (carattere arbitrario) e `%` (stringa di un numero arbitrario in cui compaiono caratteri arbitrari).
Per la gestione dei **valori nulli** viene fornito il predicato `is [not] null`.

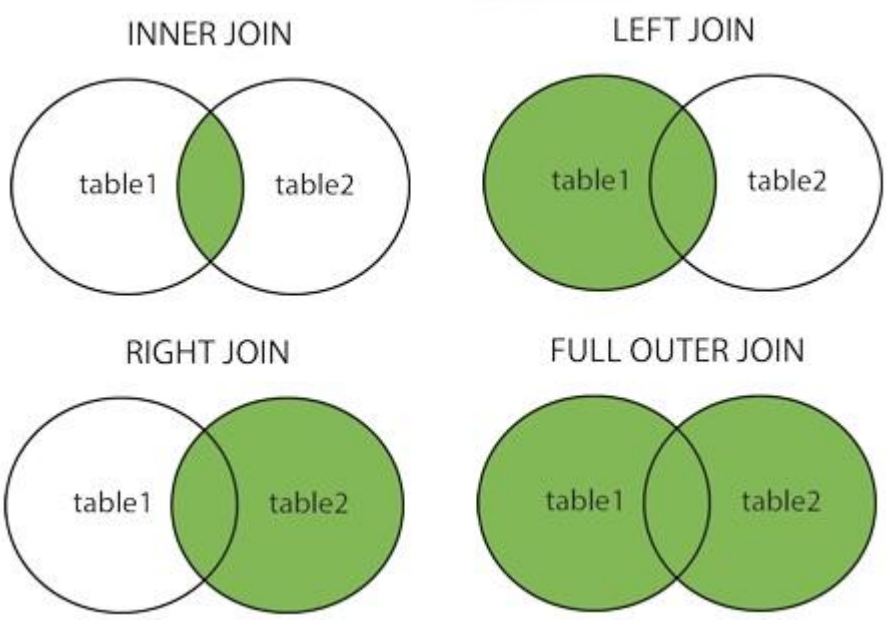
Join interni ed esterni

Una sintassi alternativa per la specifica dei Join permette di distinguere, quelle che rappresentano condizioni di join e quelle che rappresentano condizioni di selezione sulle righe. Mediante questa sintassi infatti, l acondizione di join compare come argomento della clausola where, ma viene invece spostata nell'ambito della clausola rom, associata alle tabelle che vengono coinvolte nel join.

```
SELECT AttrEspr [[as]Alias] {, AttrEspr [[as] Alias ] }
FROM Tabella [[as] Alias]
        {[TipoJoin] join Tabella [[as]Alias] on CondizionediJoin }
[WHERE AltraCondizione ]
```

Dove `TipoJoin` specifica qual è il tipo di join da usare (`inner, right outer, left outer, full outer`). Il qualificatore `outer` è omissibile. Se non viene specificato alcun tipo viene utilizzato il predefinito, `Inner Join`.

L'`inner join` rappresenta il tradizionale `theta-join` dell'algebra relazionale. Con il jion interno le righe che vengono coinvolte nel join sono in generale un sottoinsieme delle righe di ciascuna tabella. Può capitare che alcune righe non vengano considerate in quanto non esistea una corrispondente riga nell'altra tabella per cui la condizione sia soddisfatta.
Esisto tre varianti dei **join esterni**: `left, right, full`. Il `left join` fornisce come risultato il join interno esteso con le righe della tabella che compare a sinistra per le quali non esiste una corrispondente riga nella tabella di destra. Il `right join` si comporta in modo simmetrico (conserva le righe escluse dalla tabella di destra). Il `full join` resituisce il join interno esteso con le righe escluse di entrambe le tabelle.



Con **Alias** è possibile assegnare un nome più compatto alle tabelle che compaiono come argomento di FROM.
Ordinamento
Tramite la clausola ORDER BY si specifica un ordinamento delle righe del risultato di un'interrogazione. Per righe che nanno lo stesso valore di attributo si considerano gli attributi successivi, in sequenza.

Operatori aggregati
Questi operatori vengono gestiti come un'estensione delle normali interrogazioni. Prima viene normalmente eseguita l'interrogazione poi l'operatore aggregato viene applicato alla tabella risultatante. Essi sono: `count, sum, max, min e avg`.

```
count ( < * | [distinct | all ] ListaAttributi > )
```

distinct restituisce il numero di diversi valori gli attributi in `ListaAttributi`, **all** resituisce il numero di righe che possiedono valori diversi da `null` per gli attributi in `ListaAttributi`. Se si specifica un attributo e si omette distinct o all, si assume **all** come default.

```
< sum | max | min | avg > ( [distinct | all ] AttrEspress )
```

`Sum` restituisce la somma dei valori posseduti dall'espressione;
`Max` e `min` restituiscono rispettivamente il valore massimo e minimo;
`Avg` restituisce la media dei valori
`Sum` e `Avg` ammettono come argomento solo espressioni che rappresentano valori numerici o intervalli di tempo, max e min invece richiedono solamente che sull'espressione sia definito un ordinamento.
Ricordare che gli operatori aggregati non rappresentano un meccanismo di selezione, ma restituiscono semplicemente dei valori quando sono applicati a un insieme.
E' tuttavia permesso che essi siano presenti nella clausola `SELECT` se si fa uso di `GROUP BY`, descritto di seguito.

Interrogazioni con raggruppamento

Molto spesso sorge l'esigenza di applicare l'operatore aggregato separatamente a sottoinsiemi di righe. `GROUP BY` permette di specificare come dividere le tabelle in sottoinsiemi. Le interrogazioni con raggruppamento e operatori aggregati vengono eseguite esattamente come se questi ultimi non esistessero. Il risultato viene analizzato e diviso in sottoinsiemi, in ognuno dei quali viene poi eseguito l'operatore aggregato.
Negli argomenti di select delle interrogazioni che fanno uso di `GROUP BY`, possono comparire solo sottoinsiemi degli attributi usati nella clausola `GROUP BY`.

Predicati sui gruppi

Si possono voler considerare i soli sottoinsiemi che soddisfano certe condizioni. Se queste condizioni sono verificabili a livello delle singole righe, allora basta porre gli opportuni predicati come argomento della clausola where. Se invece le condizioni sono di tipo aggregato, sarà necessario utilizzare il costrutto `HAVING`.
`HAVING` descrive le condizioni che si devono applicare al termine dell'esecuzione di un'interrogazione che fa uso della clausola group by. Ogni sottoinsieme di righe costruito da `GROUP BY` fa dunque parte del risultato solo se soddisfa l'argomento di `HAVING`.
Se viene utilizzato `HAVING` senza `GROUP BY`, l'intero insieme di righe viene trattato come un unico raggrupamento.

La forma sintattica generale di un'interrogazione SQL diventa (riassumento i vari arricchimenti che abbiamo apportato fin'ora):

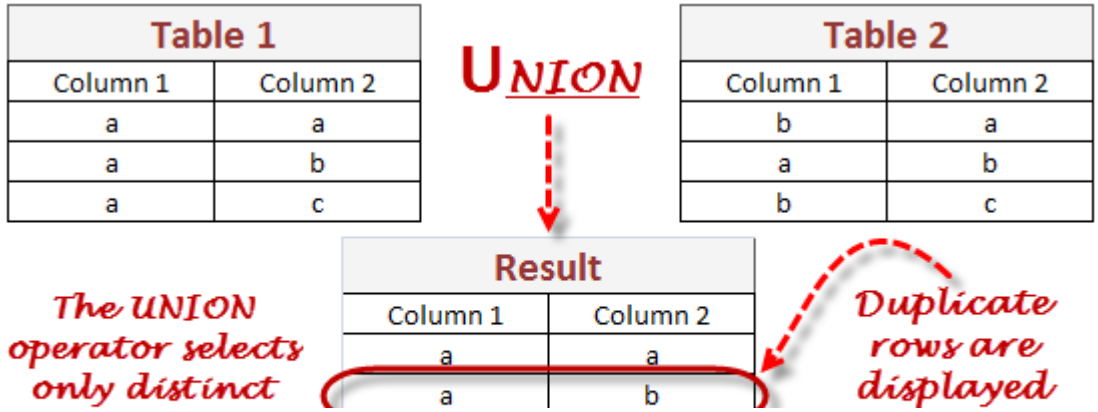
```
SELECT ListAttributi
FROM ListaTabelle
[WHERE Condizione]
[GROUP BY ListaAttributiDiRaggruppamento ]
[HAVING CondizioniAggregate ]
[ORDER BY ListaAttributiDiOrdinamento ]
```

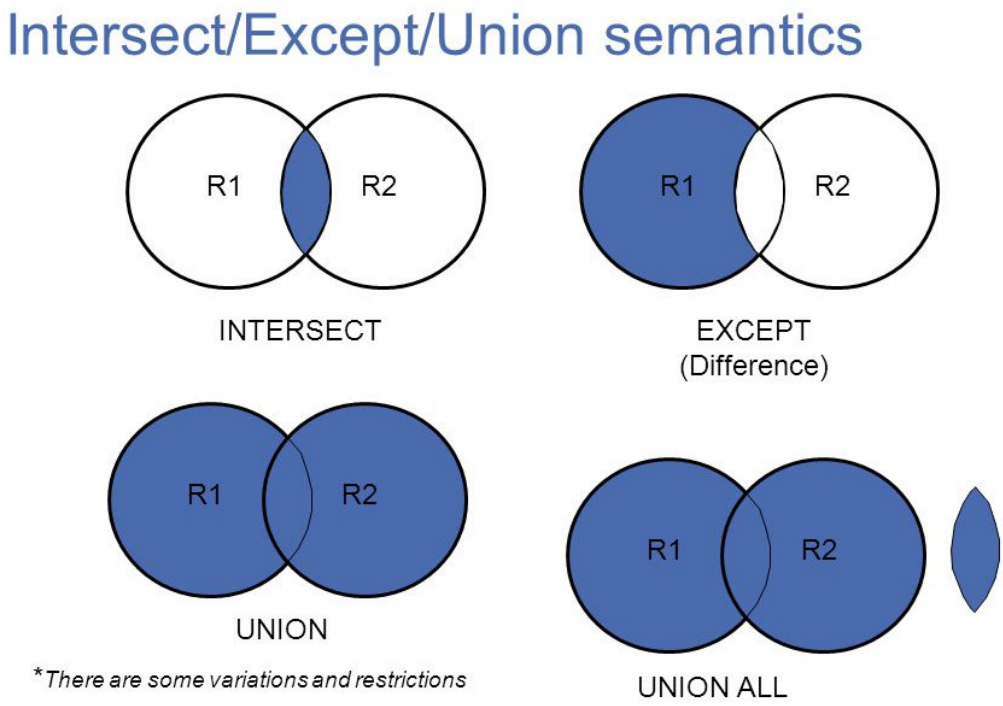
Interrogazioni di tipo insiemistico

Sono disponibili anche i classici operatori insiemistici: `union, intersect, except`.

```
SelectSQL { < union | intersect | except > [all] SelectSQL }
```

Questi operatori, al contrario del resto di SQL, assumono come default di eseguire una eliminazione dei duplicati (**all** permette di evitare questo comportamento). Non è richiesto che gli schemi su cui vengono effettuate queste operazioni siano identici, la corrispondenza tra gli attributi non si basa sul nome ma sulla posizione degli attributi. Il numero di colonne deve essere uguale e i domini compatibili.
Se gli attributi hanno nome diverso, il risultato normalmente usa i nomi del primo operando.





Interrogazioni nidificate

In generale si è visto che l'argomento della clausola where si basa su condizioni composte da predicati semplici(tramite gli operatori logici not, and e or), in cui ciascun predicato rappresenta un semplice confronto fra due valori.

SQL ammette anche l'uso di predicati con una struttura più complessa, in cui si confronta un valore (ottenuto come risultato di un'espressione valutata sulla singola riga) con il risultato dell'esecuzione di un'interrogazione SQL, definita direttamente nel predicato interno alla clausola where.

Attenzione: Se in un predicato si confronta un attributo con il risultato di un'interrogazione, sorge il **problema di disomogeneità** dei termini del confronto. Infatti, da una parte si ha il risultato di un'interrogazione SQL (in generale un insieme di valori), mentre dall'altra abbiamo il valore di un attributo per la particolare riga.

Tale problema viene risolto da SQL tramite l'utilizzo di alcune parole chiave (`all, any, in, not in, exists, not exists`) che **estendono** i normali operatori di confronto relazionale (`=, <>, <, >, <=, >=`). Any coincide con `in` e `not in` coincide con `<> all`.

La parola chiave **any** (specifica che la riga soddisfa la condizione se risulta vero il confronto (con l'operatore specificato) tra il valore dell'attributo per la riga ed **almeno uno** degli elementi restituiti dall'interrogazione nidificata.

La parola chiave **all** specifica che la riga soddisfa la condizione solo se **tutti gli elementi** restituiti dall'interrogazione nidificata rendono vero il confronto.

Ovviamente, la sintassi richiede la **compatibilità di dominio** tra l'attributo restituito dall'interrogazione nidificata e l'attributo con cui avviene il confronto.

Interrogazioni nidificate complesse

TODO

Manipolazione

`INSERT` inserisce nuove tuple nel DB
`DELETE` cancella tuple dal DB

UPDATE modifica tuple del DB

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.

- 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

Esempi
Insert-Values

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

Insert-Select

```
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”)
```

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.
- 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.
- Se la clausola where viene omessa, si intende `WHERE True`.

Esempi

```
DELETE FROM Persone
     WHERE Eta < 35

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

DELETE FROM Paternita
```

Delete e Drop

Per cancellare tutte le tuple da STUDENTE (mantenendo lo schema della tabella):
`DELETE FROM Studente`

Opzioni RESTRICT e CASCADE
`RESTRICT` impedisce drop se gli oggetti comprendono istanze
`CASCADE` applica drop agli oggetti collegati

```
DROP TABLE nomeTabella [CASCADE | RESTRICT]
```

Esempi
Per cancellare completamente la tabella STUDENTE (contenuto e schema) ed a tutte le tabelle o viste che ad essa fanno riferimento.

```
DROP TABLE Studente CASCADE
```

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

```
DROP TABLE Studente RESTRICT
```

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 la clausola WHERE, si suppone WHERE TRUE e quindi si opera la modifica su tutte le righe.
- L'istruzione UPDATE può fare uso di una condizione (per specificare le tuple da modificare) e di espressioni (per determinare i nuovi valori).
- Anche l'UPDATE può portare a violare il vincolo di integrità referenziale.

Esempi

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

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

Vincoli di integrità generici: CHECK
La clausola check può essere usata per esprimere vincoli arbitrari nella definizione dello schema CHECK (condizione)

Esempi

```
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.

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

E' un vincolo CHECK corretto, ma viene controllato solo sulla tupla che sto aggiornando -> se diminuisco lo stipendio dei dirigenti 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'))
```

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.

```
CREATE VIEW nomeVista [ ( ListaAttributi ) ] AS query
[ with [ local | cascaded ] CHECK option ]
```

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.

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

Esempi

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

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

_template:

Title 1

Title 2

Title 3

Title 4

Paragraph Text

`Monospaced Text`