Capitolo quarto

Il Linguaggio SQL standard

4.1. Introduzione

L'**SQL (**Structured Query Language**)** è un linguaggio che tratta informazioni memorizzate in strutture di dati che seguono il modello relazionale. L'IBM, presso il San José Research Laboratory, nei primi anni '70 cercava un metodo con cui i non programmatori potessero estrarre e visualizzare le informazioni necessarie da una base di dati. I linguaggi che potevano essere utilizzati dai non programmatori erano detti di quarta generazione ed il risultato di questo primo sforzo fu il **QBE** (query by example).

Successivamente nel 1974 D.D. Chamberlain, sempre del San José Research Laboratory di IBM, elaborò il linguaggio **Sequel (**Structured English Query Language) che successivamente venne rinominato SQL. Su questo linguaggio IBM basò il progetto di ricerca che portò alla realizzazione del database relazionale System R che portò alla commercializzazione dei primi prodotti relazionali SQL/DS e DB2 rispettivamente nel 1981 e 1983.

Nel 1982 l'ANSI definì un linguaggio per database relazionali, seguendo le diverse proposte di IBM, e lo denominò RDL (Relational Database Language). Le caratteristiche dell'RDL furono accolte poi dall'ISO nella prima specifica dello standard SQL del 1987. (ISO 9075:1987). La versione attuale dello standard è del 1992 (ISO 9075:1992) e si prevede che una revisione dello standard si avrà nel 1998. Attualmente il linguaggio SQL insieme al QBE è divenuto uno standard industriale ottenendo lo status di linguaggio esclusivo per i database client-server. Un database server contiene i dati mentre una applicazione client aggiunge o modifica i dati e genera le istruzioni SQL[1].

[1] "Il modello Client/Server sta diventando uno standard per molte aziende e ciò che ha permesso lo sviluppo di questa architettura è proprio SQL, che in questo contesto funge da anello di congiunzione tra il mondo PC e le piattaforme hardware più sofisticate.....Un grosso vantaggio di SQL è quello di essere indipendente dall'hardware e dal software, il suo unico requisito è che tutte le periferiche di comunicazione utilizzino il medesimo dialetto. Il problema dei dialetti è attualmente il più grande ostacolo alla sua completa maturazione e malgrado gli sforzi la strada verso una totale standardizzazione sembra ancora lunga.....il

Spesso i database attuali offrono dei tools per la realizzazione di query in QBE utilizzando la tecnologia ad oggetti ; query che il RDBMS si occuperà di trasformare in linguaggio SQL.

Il linguaggio SQL standard non è però un linguaggio rigido poiché lascia alle case produttrici la libertà di elaborare propri dialetti. Infatti, le case produttrici di database conformi allo standard SQL sono libere di estendere il linguaggio purché vengano supportati i comandi di base.

L'SQL a dispetto di quanto il suo nome possa far pensare non è un semplice linguaggio di interrogazione delle basi di dati ma comprende comandi

- **DDL** (Data Definition Language) per la definizione della struttura dei dati, delle regole di integrità e dei privilegi degli utenti ;
- **DML** (Data Manipulation Language) per leggere , inserire, modificare e cancellare dei dati.

Con il DDL si può creare, cancellare e modificare le tabelle, la base dei dati e gli indici. Il controllo dei dati (Data Control Language) definisce i meccanismi per proteggere i dati da accessi non autorizzati.

I comandi DML permettono, invece, l'estrazione dei dati dal database, la manipolazione degli stessi (cioè la possibilità di compiere delle operazioni sui dati stessi) e l'aggiornamento dei dati (cioè inserimento cancellazione e modifica dei dati di una tabella)

Il linguaggio SQL può essere utilizzato in due modi :

- in maniera interattiva
- in maniera programmatica

Nel primo caso è l'utente che lo utilizza direttamente da terminale inserendo i comandi SQL, ottenendo una risposta immediata sullo schermo[2]. Nel

comitato ANSI sta facendo il possibile per imporre uno standard che sia realmente tale ma gli interessi commerciali spingono le case produttrici a offrire implementazioni SQL ottimizzate per un dato sistema e non conformi alle specifiche di base " Papetti Giorgio Bit 6/1991.

[2] "Questo accade nei prodotti DBMS nei quali l'onere di reperire ed organizzare al meglio i dati richiesti è esclusivamente del sistema. A questo scopo, l'unica operazione preventiva richiesta è la dichiarazione degli archivi che compongono la base di dati. Gli archivi una

secondo caso, invece, l'SQL viene utilizzato all'interno di un linguaggio di programmazione (host language) e per questo si definisce come embedded o incorporato.

4.2. Formato dei comandi SQL

Per descrivere il formato dei comandi viene solitamente utilizzata la sintassi BNF (Backus Naur Form) che prende nome dagli inventori che l'hanno introdotta. Nella tabella qui sotto si inseriranno alcune notazioni più importanti.

Notazioni	**Significato**	**Esempio**
Maiuscolo minuscolo	Sono indicati in maiuscolo i comandi SQL, mentre vengono indicati in minuscolo le variabili che devono essere inserite al momento in cui si utilizzano i comandi	SELECT Cod_articolo 'Codice', Descrizione FROM Articoli WHERE prezzo > 1000
''	Si ridenomina le colonne già selezionate o si da un nome a quelle create tramite particolari operatori. Si delimita delle stringhe alfanumeriche.	
,	Separa i nome delle colonne e delle tabelle	
[ ]	Indica un elemento opzionale	SELECT [ALL]
{ }	Raggruppano più elementi di una definizione.	
...	Indicano che un elemento precedente può essere ripetuto	CREATE TABLE Fornitori (Cod_for CHAR (6) Art_for CHAR (20) **...)**
|	Separa le alternative possibili per un dato elemento di sintassi	SELECT [ALL| DISTINCT]
*****	Seleziona tutte le colonne di una tabella	SELECT * FROM Articoli
( )	Per definire3 l'ordine delle operazioni che si dovrà eseguire	SELECT * FROM Articoli WHERE NOT ((Cat_Cod ='m10' OR Cat_Cod ='m20') AND Art_Prezzo < 5000)
_ %	Il simbolo _ indica un singolo carattere qualunque mentre il simbolo % indica una sequenza di caratteri qualunque. Questi simboli detti caratteri jolly vengono utilizzati con l'operatore LIKE	SELECT Cod_articolo 'Codice', Descrizione FROM Articoli WHERE Cod_art LIKE 'T_00' oppure WHERE Cod_art LIKE '%00'

volta dichiarati, possono essere referenziati in ogni momento e per qualsiasi operazione" F. Petroni eC. Petroni MC 2/1990

4.3. Interrogazioni

L'SQL offre tutta una serie di comandi per effettuare operazioni di ricerca sui dati contenuti nelle tabelle dei Database. *Una interrogazione* della base dei dati *permette di ritrovare ed elaborare informazioni memorizzate in più tabelle, e di vederle sotto forma di tabella dei risultati*[3].

I comandi di interrogazione fanno parte di quei comandi DML che servono a leggere, inserire, modificare e cancellare i dati memorizzati in tabelle relazionali.

I gestori di database relazionali normalmente permettono di utilizzare il linguaggio SQL in maniera interattiva riportando il risultato della interrogazione sullo schermo.

Si userà nelle interrogazioni i dati di queste tabelle :

CodF	NomeF	Livello	Città
F1	Sala	20	Londra
F2	Goggi	10	Parigi
F3	Bocca	30	Parigi
F4	Carli	20	Londra
F5	Amici	30	Atene

1)Tabella fornitori F

CodP	NomeP	Colore	Peso	Città
P1	Dado	rosso	5448	Londra
P2	bullone	verde	7718	Parigi
P3	vite	blu	7718	Roma
P4	vite	rosso	6356	Londra
P5	camma	blu	5448	Parigi
P6	dente	rosso	8626	Londra

2)Tabella prodotti P

CodF	CodP	QTA
F1	P1	300
F1	P2	200
F1	P3	400
F1	P4	200
F1	P5	100
F1	P6	100
F2	P1	300
F2	P2	400
F3	P2	200
F4	P2	200
F4	P4	300
F4	P5	400

3)Tabella prodotti forniti PF

4.3.1. Schema di una interrogazione

Tutte le interrogazioni sulla base di dati vengono effettuate utilizzando il comando SELECT.

Il comando SELECT viene utilizzato per esprimere tutte le operazioni dell'algebra relazionale. Una sintassi di carattere strettamente formale viene proposta da Ullman :

SELECT Ri1.A1,.....,RirAr

FROM R1,.....,Rk

WHERE ?

GROUP BY A1,.....Ak

HAVING ?

R1,.....,Rk rappresenta una lista di tabelle distinte e Ri1.A1,.....,RirAr indica una lista di colonne Am appartenenti alle rispettive tabelle Rim che vengono selezionate per essere poi stampate. ? è una formula che può contenere i connettivi logici AND, OR, NOT, gli operatori di confronto =>< ecc. Un altro modo di rappresentare la sintassi completa del comando di selezione è quella BNF :

SELECT [ALL | DISTINCT] colonna/e | *

FROM tabella/e

[WHERE condizione]

[GROUP BY nome colonna]

[HAVING condizione]

[ORDER BY ordine]

Ogni istruzione SELECT deve essere composta almeno dalla clausola SELECT e FROM, mentre le rimanenti sono facoltative. Inoltre, l'ordine delle clausole è fisso per cui ad esempio WHERE non potrà mai precedere FROM e così via, mentre la clausola HAVING può essere utilizzata solo in presenza della clausola GROUP BY. Le clausole ALL e DISTINCT sono opzionali ed alternative. In mancanza di entrambe viene assunta come valida (si dice come valore di default) la clausola ALL. Mentre DISTINCT visualizza solo le righe distinte eliminando i valori duplicati, ALL visualizza tutte le righe indistintamente.

La forma più semplice del comando di selezione è quella composta da :

SELECT [ALL | DISTINCT] nome colonna/e

FROM nome tabella/e

[3] Cupellini Testori Vaggi "Il linguaggio SQL" FrancoAngeli

Il risultato del comando è una tabella che contiene le colonne indicate, relative alla tabella o tabelle espresse nella clausola FROM. In questo caso il comando di selezione realizza una semplice operazione di *proiezione* come viene definita dall'algebra relazionale.

Esempio

SELECT nomep, colore, peso

FROM p

CodP	NomeP	Colore
P1	Dado	rosso
P2	bullone	verde
P3	vite	blu
P4	vite	rosso
P5	camma	blu
P6	dente	rosso

4.3.2. Interrogazione con la clausola WHERE

La clausola WHERE permette di scegliere le righe della tabella che devono essere lette nella tabella dei risultati. Questa clausola di conseguenza svolge la funzione dell'operatore relazionale di selezione[4].Le condizioni della clausola WHERE sono specificate per mezzo degli operatori di confronto semplici, dei connettori logici e degli operatori BETWEEN, IN, LIKE, IS NULL.

4.3.2.1. Gli operatori di confronto

[4] "la clausola WHERE non è il solo modo per ottenere la restrizione delle righe della tabella poiché, ad esempio, anche la clausola DISTINCT permette anch'essa la restrizione delle righe. Inoltre, è importante notare che il risultato di una selezione che utilizzi la clausola WHERE su una tabella non vuota può anche essere un insieme di righe vuoto". Guidi Dorbolò "Guida a SQL" McGraw Hill.

Impostare un confronto vuol dire definire le condizioni che devono essere utilizzate per individuare le righe volute. Possiamo quindi affermare che un confronto è una forma sintattica che può assumere i classici valori booleani (vero o falso)[5]. Il risultato del confronto seguirà quindi le regole delle tavole della verità di Boole (vedere il capitolo 3).

=,	I due elementi posti a confronto devono	SELECT *
>,<,<>	essere compatibili, ovvero numerici o	FROM p
,>= ,<=	alfanumerici. Con questa operazione si	WHERE peso >6000
	effettua una selezione di righe.	

CodP	NomeP	Colore	Peso	Città
P2	bullone	verde	7718	Parigi
P3	vite	blu	7718	Roma
P4	vite	rosso	6356	Londra
P6	dente	rosso	8626	Londra

4.3.2.2. Connettori logici

Per mezzo dei connettori logici è possibile realizzare delle operazioni di confronto complesse combinando più condizioni insieme. Gli operatori in questione sono AND, OR, NOT che vengono usati per combinare fra loro confronti semplici usando eventualmente delle parentesi per stabilire l'ordine di valutazione. L'ordine di valutazione degli operatori prevede che vengano prima applicati gli operatori NOT, poi gli AND ed infine gli OR. Quindi, per definire un ordine di valutazione diverso occorre usare le parentesi tonde.

AND	L'operatore AND restituisce vero quando entrambe le condizioni che lega sono vere (P1*P2)	SELECT * FROM f WHERE livello >10 AND città = 'Londra'

CodF	NomeF	Livello	Città
F1	Sala	20	Londra
F4	Carli	20	Londra

OR	L'operatore OR restituisce vero quando almeno una delle condizioni, quella che precede o quella che segue sono vere (P1+P2)	SELECT * FROM f WHERE livello >10 OR città = 'Londra'

CodF	NomeF	Livello	Città
F1	Sala	20	Londra
F3	Bocca	30	Parigi
F4	Carli	20	Londra
F5	Amici	30	Atene

NOT	L'operatore NOT restituisce vero quando la condizione che segui è falsa	SELECT * FROM f WHERE NOT livello >10 AND città = 'Londra'

CodF	NomeF	Livello	Città
F2	Goggi	10	Parigi
F3	Bocca	30	Parigi
F5	Amici	30	Atene

[5] Cupellini Testori Vaggi "Il linguaggio SQL" FrancoAngeli

4.3.2.3. Gli operatori BETWEEN, IN, LIKE

BETWEEN

L'operatore BETWEEN verifica se un argomento è compreso in un intervallo di valori. La sintassi è la seguente :

operando BETWEEN operando AND operando.

L'operatore BETWEEN riesce a realizzare una condizione più sintetica di quella che si otterrebbe con il solo AND. Spesso però viene preferita la seconda soluzione quando la complessità della interrogazione tende ad aumentare poiché l'uso di BETWEEN prevede che gli operandi siano espressi in un ordine corretto, ovvero prima il minore e poi il maggiore per cui un errore di sequenza porta a risultati scorretti.

```
SELECT Nomep, peso, città
FROM P
WHERE peso BETWEEN 4000 AND 5500
```

Utilizzando solo AND si avrebbe avuto :

```
SELECT Nomep, peso, città
FROM P
WHERE peso>= 4000 AND peso<= 5500
```

NomeP	Peso	Città
Dado	5448	Londra
camma	5448	Parigi

IN

L'operatore IN verifica se un operando è contenuto in una sequenza di valori. Si può pervenire allo stesso risultato utilizzando l'operatore OR più volte ma l'operatore IN ha il pregio di rendere più concisa l'operazione

```
SELECT Nomep, Colore, Città
FROM P
WHERE colore IN ('rosso', 'verde')
```

Utilizzando OR si avrebbe avuto :

```
SELECT Nomep, Colore, Città
FROM P
WHERE Colore = 'rosso' OR Colore = 'verde'
```

NomeP	Colore	Città
Dado	rosso	Londra
bullone	verde	Parigi
vite	rosso	Londra
dente	rosso	Londra

LIKE

L'operatore LIKE verifica se una stringa di caratteri corrisponde ad un determinato formato. Per definire il formato si utilizzano i caratteri jolly (_) e (%). Il primo identifica un singolo carattere mentre il secondo indica una sequenza di caratteri qualsiasi.

```
SELECT Nomep, Colore, Città
FROM P
WHERE LIKE 'ross_'
```

```
SELECT Nomep, Colore, Città
FROM P
WHERE Nomep LIKE 'd%'
```

NomeP	Colore	Città
dado	rosso	Londra
dente	rosso	Londra

IS NULL

L'operatore IS NULL verifica che il contenuto di un operando sia nullo. Può accadere che un elemento della tabella in una specifica colonna non contenga nessun valore

```
SELECT Nomep, Colore, Città
FROM P
WHERE peso IS NULL
```

Nessuna riga sarà selezionata.

4.3.3. Calcolo di espressioni

La lista selezionata dal comando SELECT non contiene solo nomi di colonne ma anche espressioni calcolate sui valori di una o più colonne. Nella valutazione di una espressione, se un operando è NULL allora il risultato della espressione è NULL, cioè non specificato.

SELECT Nomep, colore, peso, peso/1000 'peso in grammi'
FROM P

NomeP	Colore	Peso	Peso in grammi
Dado	rosso	5448	5,448
bullone	verde	7718	7,718
vite	blu	7718	7,718
vite	rosso	6356	6,356
camma	blu	5448	5,448
dente	rosso	8626	8,626

4.3.4. Funzioni di gruppo

Le funzioni di gruppo (o di aggregazione) consentono di calcolare espressioni su insiemi di righe. Queste funzioni, quindi, permettono di seguire calcoli sui valori di una colonna[6] . L'SQL ne prevede 5, ovvero MAX, MIN, SUM , AVG, COUNT[7].

MAX	La funzione MAX restituisce il valore massimo contenuto in una colonna	SELECT MAX (livello) 'Livello Max' FROM F	Livello Max 30
MIN	La funzione MIN restituisce il valore minimo contenuto in una colonna	SELECT MIN (livello) 'Livello Min' FROM F	Livello Min 10
SUM	La funzione SUN calcola la somma dei valori di una colonna. Si specificherà tramite DISTINCT se i valori sommati devono essere distinti o no. Il default è ALL.	SELECT SUM (livello) 'Tot. Livello' FROM F	Tot. Livello 110

[6] La funzione COUNT(*) opera su tutta la riga e non solo su di una colonna.

[7] "Le funzioni di aggregazione possono includere il termine DISTINCT, in tal caso vengono eliminati i valori duplicati. La presenza del termine DISTINCT con le funzioni max e min non influisce sull'elaborazione, in quanto la presenza di valori duplicati non incide sul valore minimo e massimo" Belski, Breschi, Pigni, Zocchi "dati e basi di dati: Il modello relazionale" - FrancoAngeli.

AVG	La funzione AVG calcola la media aritmetica (average) dei valori non nulli di una colonna. I valori nulli (NULL) nel calcolo AVG non vengono semplicemente considerati	SELECT AVG (livello) 'Livello Medio' FROM F	Livello medio 22
COUNT	La funzione COUNT determina il numero dei valori non nulli contenuti in una colonna o il numero di righe di una tabella. Tramite le specifiche ALL e DISTINCT è possibile contare rispettivamente il numero di valori o righe selezionati o dei valori distinti (e non nulli).	SELECT COUNT (*) 'Numero Righe' FROM F	Numero di righe 5

4.3.5. La clausola GROUP BY

La clausola GROUP BY fa aumentare la possibilità di interrogazione per mezzo delle funzioni di gruppo. La clausola GROUP BY raggruppa le righe della tabella selezionata dalla clausola FROM. In ogni gruppo tutte le righe hanno lo stesso valore nella colonna indicata nella clausola GROUP BY. La tabella risultato conterrà una riga per ogni gruppo individuato dalla clausola GROUP BY e tante colonne quante sono quelle selezionate dalla clausola SELECT[8].

```
SELECT Codf, Codp, Qta
FROM PF
GROUP BY Codf
```

CodF	CodP	QTA
F1	P1, P2	300,200,
	P3 ,P4	400,200,
	P5 ,P6	100,100
F2	P1 P2	300400
F3	P2	200
F4	P2 ,P4	200,300
	P5	400

[8] Secondo la sintassi proposta da Ullman avremo :
GROUP BY A1,.....,Ak
In questo modo si suddividerà la relazione in gruppi tali che due tuple si trovano nelle stesso gruppo se e solo se sono uguali su tutti gli attributi A1,.....Ak. Perché il risultato di una tale interrogazione abbia significato , gli attributi A1,.....Ak devono comparire anche nella clausola SELECT.

4.3.6. La clausola HAVING

La clausola HAVING può essere usata solo in combinazione con la clausola GROUP BY . Questa consente di porre una condizione su di una funzione di gruppo esattamente come la clausola WHERE lo consente su singole righe. Nella clausola HAVING possono essere utilizzati gli operatori di confronto, i connettori logici e gli operatori BETWEEN, IN e LIKE[9].

```
SELECT Codf, AVG(QTA) 'Quantità media per categoria'
FROM PF
GROUP BY Codf
```

CodF	Quantità media per categoria
F1	216,667
F2	350
F3	200
F4	300

```
SELECT Codf, AVG(Qta) 'Quanta media per categoria'
FROM PF
GROUP BY Codf
HAVING AVG(Qta) >=220
```

CodF	Quantità media per categoria
F2	350
F4	300

[9] Secondo la sintassi proposta da Ullman avremo :
GROUP BY A1,.....,Ak
HAVING Y
Dove Y è una condizione che si applica ad ogni gruppo individuato dalla clausola GROUP BY.
Invece, come afferma Guidi, la condizione realizzata dalla clausola WHERE si applica sulle singole righe individuando un singolo sottogruppo.

4.3.7. Clausola di ordinamento

La clausola di ordinamento serve per dare un ordine al risultato di una selezione. Se non viene specificata tale clausola , l'ordine delle righe di un comando SELECT non è definito. Inoltre, nel modello relazionale non esiste il concetto di ordine delle righe o delle colonne. L'ordinamento può essere effettuato in base ai valori di una o più colonne e per ogni colonna può essere crescente (**ASC**) o decrescente (**DESC**)[10]. Il valore di default è ASC. La sintassi è :

SELECT [ALL | DISTINCT] colonna/e |*
FROM tabella/e
[WHERE condizione]
[ORDER BY colonna |numero colonna [ASC|DESC]
[{, colonna | numero colonna [ASC| DESC]}]

Al posto del nome della colonna si può utilizzare il numero della stessa poiché queste sono numerate. Però questo modo, di far riferimento alle colonne viene sconsigliato soprattutto per motivi di leggibilità.

SELECT Codf, Livello
FROM F
WHERE città = 'Parigi'
ORDER BY Livello

CodF	Livello
F2	10
F3	30

SELECT *
FROM F
ORDER BY Livello, Città DESC

[10] "Normalmente il valore NULL viene trattato come il massimo valore ammesso per quella colonna" Cupellini, Testori, Vaggi "il linguaggio SQL" - FrancoAngeli

CodF	NomeF	Livello	Città
F2	Goggi	10	Parigi
F1	Sala	20	Londra
F4	Carli	20	Londra
F3	Bocca	30	Parigi
F5	Amici	30	Atene

Nel secondo caso l'ordinamento che è stato creato si è basato sulla colonna Livello. All'interno dei valori di Livello uguali si inserisce l'ordinamento per città. Infatti, nelle righe con livello 30 Parigi precede Atene poiché si è dato alla colonna città un ordinamento decrescente.

4.3.8. La congiunzione (o Join) fra due tabelle

Due tabelle possono essere correlate insieme tramite una operazione di **join** se hanno logicamente in comune una colonna. Spesso i nomi delle colonne sono uguali ma questo non ha grande importanza. Quello che è importante è che devono appartenere allo stesso dominio, ovvero rappresentare lo stesso insieme di dati. La congiunzione fra le tabelle nei sistemi relazionali è assicurata dalla presenza di chiavi esterne che collegano le tabelle fra loro.

4.3.8.1. Tipi di Join

L'EQUI JOIN fra due tabelle produce una terza tabella le cui righe sono quelle ottenute dalle righe delle due tabelle in cui le colonne in comune sono tutte uguali.

Possiamo dire che l'EQUI JOIN si presenta come un caso particolare di INNER JOIN per cui la condizione che seleziona le righe non deve essere per forza quella di eguaglianza.

Esistono dei casi in cui valori delle colonne in comune non hanno corrispondenza. Affinché nelle join non si venisse a perdere questi valori si è realizzata la OUTER JOIN. La OUTER JOIN può mantenere i valori per i quali non esiste corrispondenza per una, per l'altra o per entrambe le tabelle.

Si parlerà quindi di OUTER JOIN sinistra quando si preserverà i valori non corrispondenti alla tabella specificata come primo operando, OUTER JOIN destra quando vengono preservati tutti i valori della tabella specificata come secondo operando, FULL JOIN quando si preserverà tutti i valori non corrispondenti di entrambe la tabelle.

Infine, il CROSS JOIN rappresenta il prodotto cartesiano fra due tabelle senza realizzare nessuna condizione nella clausola WHERE.

La congiunzione viene effettuata tramite un uso specifico della clausola FROM, in cui dovranno essere indicati i nomi di tutte le tabelle coinvolte nella join, e della clausola WHERE attraverso la quale si selezionerà le righe delle due colonne che appartengono allo stesso dominio che soddisfano le condizioni stabilite.

Queste considerazioni sono valide utilizzando una forma tradizionale di scrittura di una join tramite l'uso di SELECT. Nelle ultime versioni dello standard sono stati introdotti costrutti ad hoc, che però spesso non sono supportate dai DBMS commerciali attualmente più diffusi[11].

4.3.8.2. INNER JOIN e EQUI JOIN

La sintassi per queste operazioni è :

SELECT [ALL | DISTINCT] colonne selezionate dalle tabelle
FROM tabella1, tabella2
WHERE tabella1.colonna operatore tabella2.colonna

Per quanto riguarda le colonne selezionate, qualora si possano creare ambiguità riguardo l'appartenenza delle colonne alle tabelle, queste dovranno essere precedute dal nome della tabella e da un punto :

tabella1.colonna, colonna,....., colonna, tabella2.colonna, colonna,...., colonna.

L'operatore che viene indicato nella clausola WHERE può essere ogni operatore di confronto.

[11] Si tratta del comando INNER JOIN e OUTER JOIN. Si veda per questo "Guida a SQL" di Guidi Dorbolò.

Nella condizione di join è possibile utilizzare anche i connettori logici e gli operatori BETWEEN, LIKE e IN. Possono essere applicate le clausole GROUP BY, HAVING, ORDER BY.

Si consideri l'esempio :

```
SELECT F.*,P.*
FROM F,P
```

CodF	NomeF	Livello	Città	CodP	NomeP	Colore	Peso	Città
F1	Sala	20	Londra	P1	Dado	rosso	5448	Londra
F1	Sala	20	Londra	P2	bullone	verde	7718	Parigi
F1	Sala	20	Londra	P3	vite	blu	7718	Roma
F1	Sala	20	Londra	P4	vite	rosso	6356	Londra
F1	Sala	20	Londra	P5	camma	blu	5448	Parigi
F1	Sala	20	Londra	P6	dente	rosso	8626	Londra
F2	Goggi	10	Parigi	P1	Dado	rosso	5448	Londra
F2	Goggi	10	Parigi	P2	bullone	verde	7718	Parigi
F2	Goggi	10	Parigi	P3	vite	blu	7718	Roma
F2	Goggi	10	Parigi	P4	vite	rosso	6356	Londra
F2	Goggi	10	Parigi	P5	camma	blu	5448	Parigi
F2	Goggi	10	Parigi	P6	dente	rosso	8626	Londra
F3	Bocca	30	Parigi	P1	Dado	rosso	5448	Londra
F3	Bocca	30	Parigi	P2	bullone	verde	7718	Parigi
F3	Bocca	30	Parigi	P3	vite	blu	7718	Roma
F3	Bocca	30	Parigi	P4	vite	rosso	6356	Londra
F3	Bocca	30	Parigi	P5	camma	blu	5448	Parigi
F3	Bocca	30	Parigi	P6	dente	rosso	8626	Londra
F4	Carli	20	Londra	P1	Dado	rosso	5448	Londra
F4	Carli	20	Londra	P2	bullone	verde	7718	Parigi
F4	Carli	20	Londra	P3	vite	blu	7718	Roma
F4	Carli	20	Londra	P4	vite	rosso	6356	Londra
F4	Carli	20	Londra	P5	camma	blu	5448	Parigi
F4	Carli	20	Londra	P6	dente	rosso	8626	Londra
F5	Amici	30	Atene	P1	Dado	rosso	5448	Londra
F5	Amici	30	Atene	P2	bullone	verde	7718	Parigi
F5	Amici	30	Atene	P3	vite	blu	7718	Roma
F5	Amici	30	Atene	P4	vite	rosso	6356	Londra
F5	Amici	30	Atene	P5	camma	blu	5448	Parigi
F5	Amici	30	Atene	P6	dente	rosso	8626	Londra

Un risultato del genere rappresenta un prodotto cartesiano che noi indicheremo come CROSS JOIN.

Se su pone come condizione di selezionare le righe in cui le città sono uguali avremo :

SELECT F.*,P.*

FROM F,P

WHERE F.città = P.città

CodF	NomeF	Livello	Città	CodP	NomeP	Colore	Peso	Città
F1	Sala	20	Londra	P1	Dado	rosso	5448	Londra
F1	Sala	20	Londra	P4	vite	rosso	6356	Londra
F1	Sala	20	Londra	P6	dente	rosso	8626	Londra
F2	Goggi	10	Parigi	P2	bullone	verde	7718	Parigi
F2	Goggi	10	Parigi	P5	camma	blu	5448	Parigi
F3	Bocca	30	Parigi	P2	bullone	verde	7718	Parigi
F3	Bocca	30	Parigi	P5	camma	blu	5448	Parigi
F4	Carli	20	Londra	P1	Dado	rosso	5448	Londra
F4	Carli	20	Londra	P4	vite	rosso	6356	Londra
F4	Carli	20	Londra	P6	dente	rosso	8626	Londra

SELECT F.*,P.*

FROM F,P

WHERE F.città = P.città AND NOT (livello =10)

CodF	NomeF	Livello	Città	CodP	NomeP	Colore	Peso	Città
F1	Sala	20	Londra	P1	Dado	rosso	5448	Londra
F1	Sala	20	Londra	P4	vite	rosso	6356	Londra
F1	Sala	20	Londra	P6	dente	rosso	8626	Londra
F3	Bocca	30	Parigi	P2	bullone	verde	7718	Parigi
F3	Bocca	30	Parigi	P5	camma	blu	5448	Parigi
F4	Carli	20	Londra	P1	Dado	rosso	5448	Londra
F4	Carli	20	Londra	P4	vite	rosso	6356	Londra
F4	Carli	20	Londra	P6	dente	rosso	8626	Londra

4.3.8.3. OUTER JOIN

L'OUTER JOIN viene realizzato secondo i costrutti delle ultime versioni dello standard tramite dei comandi appositi. Ma ,spesso, i DBMS più diffusi attualmente sul mercato non impiegano queste indicazioni utilizzando per

questo propri metodi. Un modo per realizzare una OUTER JOIN era quello di usare l'operazione relazionale di unione utilizzando il comando UNION. Esempio :

CodF	NomeF	Livello	Città
F1	Sala	20	Londra
F2	Goggi	10	Parigi
F3	Bocca	30	Parigi
F4	Carli	20	Londra
F5	Amici	30	Atene
F6	Boghetto	40	Agliana

CodP	NomeP	Colore	Peso	Città
P1	Dado	rosso	5448	Londra
P2	bullone	verde	7718	Parigi
P3	vite	blu	7718	Roma
P4	vite	rosso	6356	Londra
P5	camma	blu	5448	Parigi
P6	dente	rosso	8626	Londra
P7	Pistone	cromo	10000	Quarrata

Se si effettua una equijoin sulle città si avrà :

```
SELECT F.*,P.*
FROM F,P
WHERE F.città = P.città
```

CodF	NomeF	Livello	Città	CodP	NomeP	Colore	Peso	Città
F1	Sala	20	Londra	P1	Dado	rosso	5448	Londra
F1	Sala	20	Londra	P4	vite	rosso	6356	Londra
F1	Sala	20	Londra	P6	dente	rosso	8626	Londra
F2	Goggi	10	Parigi	P2	bullone	verde	7718	Parigi
F2	Goggi	10	Parigi	P5	camma	blu	5448	Parigi
F3	Bocca	30	Parigi	P2	bullone	verde	7718	Parigi
F3	Bocca	30	Parigi	P5	camma	blu	5448	Parigi
F4	Carli	20	Londra	P1	Dado	rosso	5448	Londra
F4	Carli	20	Londra	P4	vite	rosso	6356	Londra
F4	Carli	20	Londra	P6	dente	rosso	8626	Londra

La OUTER JOIN può mantenere i valori per cui non esiste corrispondenza per una, per l'altra o per entrambe le tabelle e questa può essere effettuata tramite l'operazione di unione :

```
SELECT F.*,P.*
FROM F,P
WHERE F.città = P.città
UNION
SELECT ‘ ’,’ ‘; ‘ ’, ‘’, Codp, nomep, colore, peso, città
FROM P
```

```
WHERE NOT IN
(SELECT città
FROM F)
```

CodF	NomeF	Livello	Città	CodP	NomeP	Colore	Peso	Città
F1	Sala	20	Londra	P1	Dado	rosso	5448	Londra
F1	Sala	20	Londra	P4	vite	rosso	6356	Londra
F1	Sala	20	Londra	P6	dente	rosso	8626	Londra
F2	Goggi	10	Parigi	P2	bullone	verde	7718	Parigi
F2	Goggi	10	Parigi	P5	camma	blu	5448	Parigi
F3	Bocca	30	Parigi	P2	bullone	verde	7718	Parigi
F3	Bocca	30	Parigi	P5	camma	blu	5448	Parigi
F4	Carli	20	Londra	P1	Dado	rosso	5448	Londra
F4	Carli	20	Londra	P4	vite	rosso	6356	Londra
F4	Carli	20	Londra	P6	dente	rosso	8626	Londra
------	-----	------	------	P3	vite	blu	7718	Roma
-----	-----	-----	-----	P7	Pistone	cromo	10000	Quarrata

Si è realizzata in questo caso una RIGHT JOIN. Allo stesso modo si può realizzare una LEFT JOIN selezionando come righe da unire le righe della tabella F o una FULL JOIN selezionando le righe che non rientrano nella condizione WHERE F. città = P. città.

```
SELECT F.*,P.*
FROM F,P
WHERE F. città = P. città
UNION
SELECT F.* , P.*
FROM P,F
WHERE NOT IN
(SELECT F. città
FROM F,P
WHERE F. città = P. città)
```

CodF	NomeF	Livello	Città	CodP	Nome P	Colore	Peso	Città
F1	Sala	20	Londra	P1	Dado	rosso	5448	Londra
F1	Sala	20	Londra	P4	vite	rosso	6356	Londra
F1	Sala	20	Londra	P6	dente	rosso	8626	Londra
F2	Goggi	10	Parigi	P2	bullone	verde	7718	Parigi
F2	Goggi	10	Parigi	P5	camma	blu	5448	Parigi
F3	Bocca	30	Parigi	P2	bullone	verde	7718	Parigi
F3	Bocca	30	Parigi	P5	camma	blu	5448	Parigi
F4	Carli	20	Londra	P1	Dado	rosso	5448	Londra
F4	Carli	20	Londra	P4	vite	rosso	6356	Londra
F4	Carli	20	Londra	P6	dente	rosso	8626	Londra
------	-----	------	------	P3	vite	blu	7718	Roma
-----	-----	-----	-----	P7	Pistone	cromo	10000	Quarrata
F5	Amici	30	Atene	------	------	-----	------	-------
F6	Boghetto	40	Agliana	-----	------	-----	------	-------

4.3.8.4. Join su più di due tabelle

La congiunzione di due tabelle non è altro che un caso particolare della combinazione fra **n** tabelle. Si possono, quindi, estendere le considerazioni fatte precedentemente a più di due tabelle. Si può affermare che *la join fra n tabelle è una tabella dei risultati in cui ogni riga è formata dalla concatenazione di n righe, una per ciascuna delle n tabelle, secondo i criteri espressi dalla clausola WHERE*[12]. La sintassi è :

SELECT [ALL | DISTINCT] elementi
FROM tabella [{, tabella}]
[WHERE condizione di ricerca]
Esempio :

```
SELECT DISTINCT F. Nomef
FROM F, PF, P
WHERE F. codf = PF. Codf AND
        PF. Codp = P. Codp
        P. città = 'Londra'
```

Nomef
Sala
Goggi
Carli

4.3.8.5. Join riflessiva

La join riflessiva è una operazione di join effettuata su di una sola tabella. Questa viene combinata con se stessa ottenendo una nuova tabella che conterrà delle righe che saranno il risultato della concatenazione di una riga con un'altra riga della stessa tabella. Per realizzare la join riflessiva occorre indicare la tabella nelle clausola FROM due volte. Per fare questo occorre definire due *alias*

sulla tabella in questione, occorre cioè nominare la tabella due volte usando due *alias* differenti.

Sintassi :

SELECT [ALL | DISTINCT]

FROM tabella alias1, tabella alias2

WHERE alias1.colonna operatore alias2.colonna

con alias1 diverso da alias2

Esempio :

```
SELECT uno.Codf, due. Codf
FROM F uno, F due
WHERE uno.Città = Due. Città AND NOT
        (uno.Codf = due. Codf)
```

Codf	**Codf**
F1	F4
F2	F3
F3	F2
F4	F1

Si indicano così tutte le coppie di fornitori che hanno sede nella stessa città.

4.4. Operatori su insiemi

Gli operatori **UNION, EXCEPT, INTERSECT,** realizzano rispettivamente le operazioni relazionali di unione, differenza e intersezione. Questi operatori utilizzano come operandi due tabelle risultanti da comandi SELECT

[12] Cupellini Testori Vaggi "Il linguaggio SQL" FrancoAngeli

restituendo come risultato una terza tabella che contiene nel caso della unione tutte le righe, nel caso della differenza le righe che non si trovano nella seconda tabella e nel caso dell'intersezione le righe che sono comuni alle due tabelle[13]. Le due tabelle utilizzate come operandi devono però avere la stessa struttura (vedere nel terzo capitolo l'Unione). Gli operatori UNION, EXCEPT, INTERSECT, funzionano come operatori "puri" eliminando tutte le righe duplicate. Qualora le ripetizioni fossero rilevanti, allora si utilizzerà l'opzione ALL.

Sintassi :

comando SELECT

Operatore d'insieme

comando SELECT

4.5. Le subquery

Una subquery è un comando SELECT racchiuso fra parentesi tonde, inserito all'interno di un comando SQL[14]. Il risultato cui si perviene tramite le subquery può a volte essere raggiunto tramite una join ma con una maggiore leggibilità ed immediatezza[15]. A volte, però, è l'unico modo che si ha per ottenere risultati che altrimenti, utilizzando le join, non si potrebbe realizzare.

Si possono distinguere tre tipologie di subquery :scalare, di colonna, di tabella.

---

[13] Spesso in luogo dei comandi INTERSECT e EXCEPT per realizzare le operazioni relazionali di unione e intersezione delle subquery con l'utilizzo gli operatori di confronto quantificati =ANY e <> ANY.

[14] La subquery spesso viene utilizzata all'interno della clausola WHERE. Però, le subquery che producono un solo valore (dette scalari) possono trovare utilizzazione nella lista di selezione come espressione. Vedere Guidi Dorbolò : Guida ad SQL pag 90

[15] " Una domanda che sorge spontanea riguarda l'efficienza dell'una e dell'altra forma sintattica. Esistono in effetti delle differenze elaborative nel valutare l'una o l'altra forma e sembra che la forma con interrogazioni nidificate sia più onerosa e quindi meno efficiente della join : Ad esempio nell'SQL DB2 si preferisce l'uso della join a quello delle sottointerrogazoni. Infine, sulle motivazioni che hanno portato alla esistenza dei due formati perfettamente intercambiabili di porre le interrogazioni si deve notare che la forma nidificata (strutturata) è stata la prima in ordine di tempo ad essere introdotta in SQL, tanto che da

La subquery **scalare** è un comando SELECT che restituisce un solo valore, mentre la subquery di **colonna** restituisce una colonna (con più righe). Infine, la subquery di **tabella** restituisce come risultato una tabella. Possiamo dire che la subquery scalare costituisce un caso particolare di subquery di colonna e che a sua volta la subquery di colonna costituisce un caso particolare di subquery di tabella.

Le subquery possono essere nidificate, ovvero essere inserite le une nelle altre racchiuse ordinatamente in parentesi tonde. Si analizzerà, così, prima quella più interna e via via tutte le altre fino all'ultima che restituirà o un valore o una colonna o una tabella e su questo o questi valori ottenuti si eseguirà in comando SQL in cui queste subquery sono inserite.

4.5.1. Subquery semplice

Si ha una subquery semplice quando il secondo elemento di un confronto che utilizza operatori booleani uno degli operatori di confronto viene sostituito da una sottointerrogazione. Nel primo caso la subquery deve restituire una serie di valori per cui sarà ammessa solo una query di colonna, mentre nel secondo caso deve restituire un solo valore e quindi la sola query ammessa è quella scalare. Non è possibile effettuare un confronto fra due subquery ne è possibile utilizzare le clausole GROUP BY ed HAVING.

Esempi :

Ricercare il nome dei fornitori che producono il prodotto P2.

```
SELECT Nomef
FROM F
WHERE Codf IN
        (SELECT Codf
         FROM PF
         WHERE Codp = 'P2')
```

essa deriva anche il termine di "Structured Query Language"" Cupellini Testori Vaggi :Il

Si poteva utilizzare in questo caso anche una join :

```
SELECT Nomef
FROM F, FP
WHERE F. Codf = FP. codf  AND codp = P2
```

NomeF
Sala
Goggi
Bocca
Carli

Ritrovare i codici dei fornitori che forniscono prodotti di colore rosso.

```
SELECT codf
FROM PF
WHERE Codp  =
        (SELECT Codp
         FROM P
         WHERE colore = 'rosso')
```

Usando le join :

```
SELECT codf
FROM PF, P
WHERE FP. Codp  =P. Codp AND PF. Colore = 'rosso'
```

Codf
F1
F2
F4

linguaggio SQL.

4.5.2. Subquery a più livelli (o nidificate)

Le subquery possono essere nidificate a più livelli di profondità. In questi casi l'interrogazione più interna viene valutata per prima e fornisce valori per quella di livello superiore arrivando via via a valutare l'intera interrogazione.

Esempio :

```
SELECT NOME F
FROM F
WHERE Codf IN
        (SELECT Codf
         FROM PF
         WHERE Codp IN
                 (SELECT Codp
                  FROM P
                  WHERE Colore = 'Rosso'))
```

Nomef
Sala
Goggi
Carli

4.5.3. Subquery di confronto quantificato (clausola ANY e ALL)

La clausola ANY permette di confrontare un elemento con un insieme di valori individuati da una subquery per verificare se almeno uno di essi soddisfa il criterio di confronto.

Invece la clausola ALL permette di confrontare un elemento con un insieme di valori individuati da una subquery per verificare se tutti soddisfano il criterio di confronto.

```
SELECT [ALL | DISTINCT] elementi
FROM tabelle
WHERE elemento operatore ANY | ALL
        (SELECT [ALL | DISTINCT] elementi
         FROM tabelle
```

[WHERE condizione])

l'elemento può essere sia una colonna o un espressione mentre l'operatore può essere un qualsiasi operatore di confronto.

Esempi :

Ritrovare i codici dei fornitori il cui livello è inferiore ad almeno uno dei livelli dei fornitori.

```
SELECT
FROM F
WHERE Livello < ANY
        (SELECT Livello
         FROM F)
```

Codf
F1
F2
F4

Ritrovare i codici dei fornitori il cui livello è minore o uguale a tutti i livelli dei fornitori.

```
SELECT
FROM F
WHERE Livello <=ALL
        (SELECT Livello
         FROM F)
```

Codf
F2

Si può notare come in questi casi l'uso della clausola ANY corrisponda ad una "concatenazione" di confronti effettuati con il connettore logico OR mentre ALL corrisponda sempre ad una "concatenazione" di confronti con AND.

Tramite un uso particolare della clausola ANY (= ANY , <>ANY) si possono realizzare le operazioni di intersezione e differenza fra due tabelle. Spesso, infatti, i DBMS più diffusi in commercio non includono gli operatori INTERSET e EXCEPT per cui queste operazioni devono essere effettuate con operatori tradizionali.

4.5.4. Subquery con clausola EXISTS

La clausola EXISTS permette di verificare l'esistenza di almeno una riga in una tabelle dei risultati prodotta da una subquery. Quindi si tratta in questo caso di una subquery di tabella.

La sintassi è :

```
SELECT [ALL | DISTINCT] elementi |*
FROM tabelle
WHERE  [NOT] EXISTS
        (SELECT elementi |*
         FROM tabelle
         [WHERE condizione])
```

La clausola EXISTS è verificata (e quindi viene soddisfatta la condizione posta dalla WHERE) se la tabella prodotta dalla subquery è costituita da almeno una riga. Invece la clausola NOT EXISTS è verificata se la subquery produce un insieme vuoto.

Esempio :

Ricercare il nome dei fornitori che forniscono il prodotto P2

```
 SELECT Nomef
FROM F
WHERE EXISTS
        (SELECT *
         FROM PF
```

```
        WHERE Codp = F. Codf AND
                Codp = P. Codp))
```

NomeF
Sala
Goggi
Bocca
Carli

In questo caso per ogni riga della tabella F viene verificato se la subquery produce almeno una riga. In questo caso la clausola WHERE è verificata e quindi il Nomef della riga F viene selezionato. La clausola NOT EXISTS, invece, viene verificata se la tabella prodotta dalla subquery non contiene neppure una riga.

```
SELECT Nomef
FROM F
WHERE NOT EXISTS
        (SELECT *
         FROM PF
         WHERE Codp = F. Codf AND
                Codp = P. Codp))
```

NomeF
Amici

Le condizioni EXISTS e NOT EXISTS sono forme semplificate di subquery di confronto quantificato (vedere esempi di Guidi Dorbolò :Guida a SQL).

4.6. Manipolazione dei dati

I comandi di manipolazione fanno parte come i comandi di interrogazione dei comandi **DML** del linguaggio SQL. Questi comandi permettono di

realizzare l'inserimento, la modifica e la cancellazione di dati nelle tabelle tramite i comandi INSERT, UPDATE e DELETE .

4.6.1. Comando INSERT

Il comando INSERT viene utilizzato per l'inserimento dei dati nelle tabelle. La sintassi è :

INSERT INTO Nome tabella [(lista delle colonne)]

Origine

"Nome tabella" indica il nome della tabelle di origine in cui i dati dovranno essere inseriti, mentre lista delle colonne indica le colonne in cui dovranno essere inseriti i dati. Se le colonne non verranno specificate si prenderanno in considerazione tutte le colonne della tabella. "Origine", invece, indica i valori o la tabella di provenienza dei dati da inserire. Quindi i valori da inserire possono essere rappresentati da valori nuovi o possono essere derivati da un'altra tabella o da una query. Nel primo caso origine è specificata dalla clausola VALUES. I nuovi valori devono essere inseriti nella tabella rispettando l'ordine delle colonne. La eventuale presenza di valori nulli deve essere effettuata esplicitamente tramite la parola chiave NULL.

Esempio :

INSERT INTO F

VALUES ('F6', 'Boghetto', 40, 'Agliana')

Viene in questo modo inserita una riga alla tabella F.

Nel caso in cui "Origine" sia costituito da una tabella le colonne di questa tabella devono appartenere ordinatamente agli stessi domini della tabella di inserimento. La stessa cosa può essere detta qualora i dati derivino da una query.

Esempio :

INSERT INTO F

SELECT *

FROM F1

In questo caso le colonne della F1 devono appartenere ordinatamente agli stessi domini della F.

Il risultato sarà l'inserimento delle righe della F1 nella F.

Se la lista delle colonne viene omessa nella INSERT si considererà la tabella composta da tutte le colonne. Nel caso in cui "origine" sia VALUES o una query e non vengano specificate tutte le colonne della tabella di destinazione, oppure l'ordine delle colonne della tabella di destinazione sia diverso da quello di creazione, allora sarà necessario inserire la "lista delle colonne". Occorre però nel caso di indicazione della "lista delle colonne" osservare alcune regole :

1. il nome di una colonna non può apparire più di una volta altrimenti si genera ambiguità
2. se non vengono specificate tutte le colonne quelle non incluse devono ammettere valore NULL. Occorre, quindi, specificare almeno le colonne che costituiscono chiave primaria.

Esempio :

INSERT INTO P (Codp , NomeP)

VALUES ('P7', 'Pistone')

4.6.2. Il comando UPDATE.

Il comando UPDATE viene utilizzato per la modifica dei dati nelle tabelle. La sintassi è :

UPDATE tabella

SET colonna = espressione [colonna =espressione **...**]

WHERE [condizione]

"tabella" indica la tabella nella quale i dati dovranno essere modificati, mentre la clausola SET identifica le colonne e i valori con cui devono essere modificate. Questa clausola può specificare più colonne, indicando per ognuna le modalità di aggiornamento tramite una espressione. Questa espressione potrà anche essere una subquery di tipo scalare, basta cioè che dia come risultato un unico valore. E' possibile assegnare il valore NULL ad esclusione ovviamente della chiave primaria della tabella e di eventuali altre colonne che non accettano valori nulli. La clausola WHERE pone una condizione con la quale si identifica le righe a cui va limitato l'aggiornamento. La condizione nella clausola WHERE può essere realizzata tramite una subquery che deve però essere valutata una sola volta prima della esecuzione dell'aggiornamento poiché, altrimenti, potrebbe portare ad una modificazione a catena delle righe della tabella[16].

Se viene omessa, tutte le righe della tabella saranno soggette ad aggiornamento.

Esempio :

```
UPDATE P
SET Peso = (SELEC Peso
            FROM P
            WHERE Codp =P3)
WHERE  Codp = P4
```

Sostituisco il peso della vite rossa con quello della vite blu.

```
UPDATE P
SET Peso = Peso *1.2, città = Agliana
WHERE  Codp = P4
```

[16] Vedere esempi specifici in Guidi Dorbolò : Guida all'SQL

Nella riga P4 il peso della vite viene aumentato del 20% mentre la città viene cambiata da Londra in Agliana.

4.6.3. Il comando DELETE

Il comando DELETE serve per cancellare le righe di una tabella. La sintassi è :

DELETE FROM tabella

[WHERE condizione]

Se la condizione non è specificata allora tutte le righe della tabella verranno cancellate. La condizione può essere realizzata tramite una subquery che deve però essere valutata una sola volta prima della esecuzione della cancellazione perché potrebbe portare ad una cancellazione a catena delle righe della tabella.

```
DELETE FROM P
WHERE Peso = (SELECT MIN (Peso)
              FROM P)
```

4.7. Definizione dei dati

Attraverso i comandi DDL si realizzano funzioni per la gestione degli schemi, ovvero la per la creazione, modifica e cancellazione delle tabelle e degli altri oggetti del database[17]. Attraverso i comandi DDL si tradurrà lo schema logico ottenuto seguendo le regole relazionali. I comandi DDL per la definizione della struttura dei dati , delle regole di integrità e dei privilegi degli utenti sono :

CREATE SCHEMA

CREATE TABLE

CREATE VIEW

GRANT

[17] "Alcuni di questi comandi permettono di realizzare funzioni DCL (Data Control Language) perché permettono di designare il proprietario degli oggetti di uno schema logico influenzando le autorizzazioni su tali oggetti e perché alcuni comandi permettono di effettuare la gestione degli indici" Guidi Dorbolò :Guida a SQL .

4.7.1. Creazione dello schema

In uno schema logico vengono definite tutte le tabelle che compongono un database relazionale. A ciascun schema è associato un *proprietario* che di norma è la persona che ha creato lo schema e che solitamente si identifica con *l'amministratore del database*. Il proprietario detiene tutti i privilegi possibili sugli elementi definiti all'interno dello schema. Un concetto molto importante è quello di catalogo (o dizionario dei dati)[18]. Questo è a sua volta uno schema in cui vengono inseriti tutti i "dati salienti" relativi alla struttura del database (è la base dati dei dati). I dizionari dei dati contengono una descrizione del database nel suo insieme, di ogni tabella che esso contiene, dei campi che compongono la tabella, delle chiavi primarie ed esterne e dei valori che possono essere assegnati ai campi. Vengono inclusi, a volte, anche lo scopo e la descrizione di ogni applicazione che fa uso del database.

Esempio di dizionario

DATE : 31 / 10 / 96 TIME : 18:01:04 PAGE : 4

DATA DICTIONARY REPORT

Schema Listing

TABLE / FIELD REF TYPE LENGTH LONG NAME

[18] "Il catalogo ha un rapporto con la base di dati che è uguale al rapporto che esiste fra l'indice di un libro ed il libro stesso..... Infatti una base di dati può contenere non solo le tabelle dell'utente ma anche certe tabelle del sistema, la cui funzione è di contenere i dati necessari al sistema stesso per gestire la base di dati per l'utente. .....Il catalogo può essere aggiornato ed interrogato con le normali possibilità di un linguaggio per le basi di dati" C.J. Date : Introduzione ai database - Masson

" Il catalogo è una base di dati relazionale, nella quale sono mantenute definizioni, informazioni di controllo ed informazioni di carattere generale sui dati, sugli utenti e sui programmi presenti nel sistema. Esso comprende un insieme di tabelle, aggiornate o direttamente dall'SQL, o indirettamente dall'utente che svolge attività di amministratore della base di dati, e interrogabile da ogni utente. Alcune tabelle presenti nel dizionario sono :

- SYSCATALOG : contiene tante righe quante sono le tabelle presenti nel sistema. Per ogni tabella mantiene il nome dell'utente che l'ha definita, il numero delle righe, l'indirizzo in memoria (non accessibile all'utente) ecc.. Nel SYSCATALOG compaiono anche informazioni relative alle tabelle del dizionario dei dati e alle tabelle virtuali.
- SYSCOLUMNS :descrive le colonne presenti nel sistema, indicanti nome, tipo, lunghezza del campo e se sono permessi i valori nulli.
- SYSINDEX :raccoglie informazioni sugli indici
- SYSVIEW :contiene la definizione di tutte le tabelle virtuali.
- SYSERUTH :mantiene l'identificatore, le password e l'autorità di ogni utente." Albano :Basi di dati -Boringhieri

```
   *al_cod            COMB        Codice_Alleg
    al_pfx           STRING    1  Pfx_Alleg
    al_num            NUMERIC    6  Numero_Alleg
   al_cia           AMOUNT   11  Corr_Impon_Att
   al_cva            AMOUNT   11  Corr_Iva_Att
   al_cea            AMOUNT   11  Corr_Esente_Att
   al_cna            AMOUNT   11  Corr_NonImp_Att
   al_cip           AMOUNT   11  Corr_Impon_Prec
   al_cvp            AMOUNT   11  Corr_Iva_Prec
   al_cep            AMOUNT   11  Corr_Esente_Prec
   al_cnp            AMOUNT   11  Corr_NonImp_Prec
   al_pia           AMOUNT   11  Prec_Impon_Att
   al_pva            AMOUNT   11  Prec_Iva_Att
   al_pea            AMOUNT   11  Prec_Esente_Att
   al_pna            AMOUNT   11  Prec_NonImp_Att
   al_pip           AMOUNT   11  Prec_Impon_Prec
   al_pvp            AMOUNT   11  Prec_Iva_Prec
   al_pep            AMOUNT   11  Prec_Esente_Prec
   al_pnp            AMOUNT   11  Prec_NonImp_Prec
   al_cfa           NUMERIC   6  Corr_NumFat_Att
   al_cfp           NUMERIC   6  Corr_NumFat_Prec
   al_pfa           NUMERIC   6  Prec_NumFat_Att
   al_pfp           NUMERIC   6  Prec_NumFat_Prec
   al_nie           AMOUNT   11  Non_imp_101_102
ammfisc     300                 ammfisc
  *amf_cod           COMB        Codice
   amf_ces    ces_cod  STRING   10  Cespite_Ammort
    amf_ann          NUMERIC   4  Anno_Ammort
   amf_mes          NUMERIC   2  Mese_Ammort
   amf_flv          STRING    1  Flag_Simulazione
   amf_imp           AMOUNT   11  Imp_Amm_Norm
   amf_ima           AMOUNT   !1  Hmp^Am4_@nt
.............              ..............      ..........................
.............              ..............      ..........................
```

DATE : 31 / 10 / 96 TIME : 18:01:26 PAGE : 66

DATA DICTIONARY REPORT

Table List

TABLE	NUMBER	EXPECTED	LENGTH
agenda	60	200	754
agenzie	93	1500	396
aggreg	151	100	1760
allegati	68	1000	162
ammfisc	76	300	106
assrib	153	200	88
banche	40	600	270
basirib	134	200	74
bilric	98	100	78
caldesc	154	50	26
calendar	89	500	22
cashcau	94	100	56
cashimp	95	200	66
cashmov	96	200	22
caufat	47	100	54
cauiva	27	50	244
causclv	85	100	78
causcoge	43	250	104
cdc	9	100	98
cescau	77	100	36
cesclc	87	100	412
cesleg	104	100	64
cesmov	114	100	60
cespctrl	115	1	76
cespiti	116	100	382
cfling	172	50	40
cicdes	13	100	52
cicli	11	100	242
clcosto	140	100	64
clifor	8	1500	1246
clisped	39	300	278

commesse	15	100	148
conti	29	300	360
contratt	45	100	176
contris	150	30	506
control	41	1	242
control2	42	1	348
coscau	58	100	26
cospar	34	1000	338
cpling	174	50	86
cririb	147	200	88
ctrlav	10	100	394
ctrlcoan	145	1	210
depositi	49	50	298
.....	...	.....	.....
.....	...	....	....

DATE : 31 / 10 / 96 TIME : 18:01:28 PAGE : 71

DATA DICTIONARY REPORT

Field List

FIELD	NUMBER	TABLE	TYPE	LENGTH
age_min	2047	agenzie	NUMERIC	2
age_mit	1612	agenzie	NUMERIC	5
age_msa	2050	agenzie	NUMERIC	2
age_msc	2053	agenzie	NUMERIC	2
ag_ena	931	tabage	STRING	16
age_pif	1134	agenzie	STRING	15
age_pro	1610	agenzie	STRING	2
age_rgs	1606	agenzie	STRING	30
age_sin	2049	agenzie	NUMERIC	6
age_spe	1502	agenzie	FLOAT	82
age_spo	1604	agenzie	NUMERIC	5
age_ssa	2052	agenzie	NUMERIC	6
age_ssc	2055	agenzie	NUMERIC	6
age_sto	1501	agenzie	NUMERIC	6
age_tci	1611	agenzie	STRING	1
age_tes	2273	agenzie	TEXT	78
ag_fap	1649	tabage	AMOUNT	11
ag_fat	264	tabage	AMOUNT	11

ag_ind	42	tabage	STRING	35
ag_isp	1999	tabage	NUMERIC	3
ag_loc	260	tabage	STRING	35
ag_pro	261	tabage	STRING	2
ag_prp	1648	tabage	AMOUNT	11
ag_prv	263	tabage	AMOUNT	11
ag_pva	932	tabage	FLOAT	62
ag_pvs	1997	tabage	FLOAT	52
agr_1	2775	aggreg	STRING	100
agr_10	2784	aggreg	STRING	100
agr_11	2785	aggreg	STRING	100
agr_12	2786	aggreg	STRING	100
agr_13	2787	aggreg	STRING	100
agr_14	2788	aggreg	STRING	100
agr_15	2789	aggreg	STRING	100
agr_16	2790	aggreg	STRING	100
agr_17	2791	aggreg	STRING	1
agr_18	2792	aggreg	STRING	100
agr_19	2793	aggreg	STRING	100
agr_2	2776	aggreg	STRING	100
agr_20	2794	aggreg	STRING	100
agr_21	2795	aggreg	STRING	100
agr_3	2777	aggreg	STRING	50
agr_4	2778	aggreg	STRING	50
agr_5	2779	aggreg	STRING	100
agr_6	2780	aggreg	STRING	100
agr_7	2781	aggreg	STRING	10
agr_8	2782	aggreg	STRING	1
agr_9	2783	aggreg	STRING	100
.....	.....	.....	.....	....
.....	.....	.....	.....	....

Figura 4.1. Esempio di Catalogo della Società Cotton Joy di Quarrata (PT)

Comunque, l'utilizzo del catalogo non è indispensabile , ma può essere usato quando un DBMS sia utilizzato per la gestione di più di un database e sia necessario raggruppare logicamente gli schemi in categorie distinte . La sintassi di definizione di un comando è :

CREATE SCHEMA nome schema

[CREATE SCHEMA nome catalogo. nome schema]

CREATE SCHEMA AUTORISATION nome proprietario

CREATE SCHEMA nome schema AUTORISATION nome proprietario
[DEFAULT CHARACTER SET set di caratteri]
[elementi dello schema **...**]

Come si vede la creazione di un catalogo può essere opzionale. La specifica [DEFAULT CHARACTER SET set di caratteri] viene utilizzata qualora si vogliano usare set di caratteri particolari come ad esempio quello russo o cinese. Se la clausola non viene specificata si usano i caratteri attualmente in uso.
La specifica [elementi dello schema ...] comprende la definizione delle tabelle, delle viste, e di altri elementi come i domini, le assertion ecc.. che comunque possono essere effettuate anche in maniera indipendente e al di fuori del comando CREATE SCHEMA.
Esempio :
CREATE SCHEMA Cottonjoy
CREATE SCHEMACatalog. Cottonjoy
CREATE SCHEMA AUTORISATION Josef
CREATE SCHEMA Cottonjoy AUTORISATION Josef

4.7.2. Creazione di tabelle

Le ultime versioni di SQL prevedono due tipi di tabelle : le tabelle permanenti e quelle temporanee. La tabelle temporanee si differenziano dalle altre per il fatto che, mentre la loro struttura rimane permanente, il loro contenuto, cioè le righe della tabella possono essere cancellate al termine di ogni transazione. Nella sintassi relativa alla creazione delle tabelle dovrà essere indicato se si tratta di tabella temporanea o permanente. Se si vuole che la tabella sia del tipo temporaneo occorre indicarlo esplicitamente tramite la clausola TEMPORARY. In questo caso dovranno essere aggiunte anche tutta una serie di informazioni attraverso delle clausole specifiche che nella

definizione della tabella di tipo permanente non troveranno applicazione. La definizione delle tabelle temporanee, secondo quanto afferma lo standard, non è supportata dalla maggior parte dei DBMS commerciali per cui si rimanda lo studio relativo alla creazione di queste tabelle alla manualistica specifica[19]. Invece, la sintassi di una tabella permanente è :

```
CREATE TABLE tabella
   (definizione elemento della tabella
    [{, definizione elemento della tabella }...])
```

La "definizione elemento della tabella" è costituito dalla **definizione delle colonne** e dai **vincoli di tabella.**

4.7.3. Definizione delle colonne

La sintassi è :

```
CREATE TABLE tabella
   (nome colonna    tipo di dati
    [clausola di default]
    [definizione del vincolo di colonna]
```

La clausola di default, che è opzionale, indica il valore di default da assegnare agli elementi della colonna. Potranno essere, questi valori, costanti o funzioni che dovranno essere, però, sempre dello stesso tipo della colonna cui si riferiscono. Allo stesso modo è opzionale il vincolo di colonna nelle quali vengono definiti i vincoli di integrità.

4.7.3.1. Tipi di dati

Il linguaggio SQL prevede tutta una serie di tipi di dati che sono riportati nella tabella :

[19] Vedere Guidi Dorbolò : Guida a SQL

CHARACTER	**BIT**
NUMERIC	**DATE**
DECIMAL	**TIME**
INTEGER	**TIMESTAMP**
SMALLINT	**INTERVAL**
FLOAT	
REAL	
DOUBLE	

I dati alla sinistra appartengono tutti alla ultima revisione dello standard e spesso non vengono utilizzati. Comunque, quello di uso più frequente è DATE.

CHARATER	Viene usato per specificare il tipo di dati **caratteri** e la lunghezza degli stessi all'interno della colonna. La sintassi è : nome colonna CHARACTER [VARYING] [(lunghezza)] [CHARACTER SET set di caratteri] Se viene indicata l'opzione VARYING la lunghezza può variare da zero fino al massimo consentito, altrimenti questa è fissa. La indicazione della lunghezza nel caso dell'opzione VARYING è obbligatoria, non lo è nel caso in cui non si utilizzi per cui il valore di default sarà 1. La parola CHARACTER può essere abbreviata con **CHAR**	Possibili specifiche del tipo CHARACTER : CHAR CHAR (10) CHAR VARYING (100)
NUMERIC **DECIMAL** **INTEGER** **SMALLING**	Questi tipi numerici rappresentano numeri decimali positivi e negativi con o senza la virgola. La sintassi è : NUMERIC [(precisione [,scala])] DECIMAL [(precisione [,scala])] INTEGER SMALLING Per DECIMAL e INTEGER si possono utilizzare anche le abbreviazioni DEC e INT. La "precisione" indica il numero massimo complessivo di cifre a sinistra e a destra della virgola. La "scala" indica il numero massimo di cifre a destra della virgola. La "precisione" deve essere	Possibili specifiche : NUMERIC NUMERIC (5) DECIMAL (5,3) INTEGER come si vede NUMERIC (5) è un intero composto al massimo da 5 cifre, mentre DECIMAL è un tipo numerico composto al massimo da 5 cifre di cui 3 a destra della virgola.

	maggiore o uguale a 1, mentre la scala maggiore o uguale a 0. La precisione massima per il tipo DECIMAL deve essere maggiore o uguale a quella fornita per il tipo NUMERIC. La "precisione" di default è definita dai vari DBMS mentre quella di "scala" è 0. Spesso, però, in molti DBMS questi tipi sono sinonimi. Lo stesso può essere detti per INTEGER e SMALLING che rappresentano tipi numerici esatti con scala uguale a zero.
FLOAT **REAL** **DOUBLE PRECISION**	Questi sono tipi numerici approssimati. Vengono rappresentati tramite m (mantissa) ed n (esponente). La mantissa può essere un numero decimale positivo o negativo con o senza la virgola. Esempio : mEn è la rappresentazione del numero $m*10^n$ è il valore L'esponente può essere un numero intero positivo o negativo. La sintassi per ciascuno dei tipi numerici approssimati è : FLOAT [(precisione)] REAL DOUBLE PRECISION In FLOAT la "precisione" si riferisce alla mantissa che deve essere maggiore di zero. La precisione massima è definita dal singolo DBMS. La "precisione" per REAL e DUOBLE PRECISION è definita dal BDMS, comunque la precisione di DOUBLE PRECISION deve essere maggiore di REAL.
DATE	Il tipo date rappresenta le date intese come sequenza di tre valori . Infatti, le costanti di tipo DATE sono espresse da questo formato : DATE 'AAAA-MM-GG' La funzione CURRENT DATE da la data corrente derivandola dal sistema.

4.7.4. I vincoli di integrità

Nel linguaggio SQL sono stabilite le opzioni di definizione del vincolo di tabella e del vincolo di colonna. Queste hanno come scopo quello di definire dei vincoli di integrità sui dati del database limitando , in questo modo, i valori ammissibili per una determinata colonna della tabella in base a particolari criteri. Una prima forma di controllo viene realizzata dai tipi di dati e riguarda il formato dei dati inseriti. Ad esempio, se vogliamo inserire

in una colonna definita dal tipo CHARACTER una cifra, questa operazione verrà impedita dal sistema. Con la definizione dei vincoli di integrità si fornisce un controllo che non riguarda solo il singolo valore da inserire ma anche le relazioni di questo valore all'interno della stessa colonna, della tabella e le relazioni che questo ha con le colonne di altre tabelle.

La sintassi per un vincolo di colonna è :

[CONSTRAINT nome vincolo]
{vincolo NOT NULL | vincolo di univocità | vincolo di chiave esterna | vincolo di controllo}
[attributi del vincolo]

La sintassi di un vincolo di tabella è :

[CONSTRAINT nome vincolo]
{vincolo NOT NULL | vincolo di univocità | vincolo di chiave esterna | vincolo di controllo}
[attributi del vincolo][20]

Sia per il vincolo di tabella quanto per il vincolo di colonna dare un nome a questo vincolo è facoltativo. Se viene omesso è lo stesso DBMS che lo assegna.

[20] "mentre gli "attributi del vincolo" sono :

[NOT] DEFERRABLE
[INITIALLY IMMEDIATE | INITIALLY DEFERRED] |
[INITIALLY IMMEDIATE | INITIALLY DEFERRED]
[NOT] DEFERRABLE

Le opzioni "attributi del vincolo" si riferiscono alla modalità del vincolo, che può essere immediata o differita. Un vincolo immediato viene verificato dal DBMS dopo ogni comando SQL ; un vincolo differito viene verificato dal DBMS al temine di ogni transazione sul database. Come si può dedurre dalla presenza della parola chiave INITIALLY, la modalità di un vincolo può essere cambiata successivamente alla definizione del vincolo stesso." Guidi Dorbolò : Guida a SQL.

4.7.5. Il vincolo NOT NULL

Il vincolo NUT NULL stabilisce che la colonna non può avere valori nulli. Questo vincolo è riferito solo ad una colonna e quindi può essere impiegato solo come vincolo di colonna .

```
CREATE TABLE articoli
(...
Desc_articoli CHAR (50) NOT NULL
...)
```

4.7.6. Vincolo di unicità

I vincoli di unicità sono due **UNIQUE** e **PRIMARY KEY** . Questa clausola consente di *stabilire che nella tabella non possano esistere delle righe con valori duplicati per una colonna o per una combinazione di colonne*[21]. Il vincolo UNIQUE verifica che ciascun valore sia unico all'interno di una o più colonne. Nel primo caso può essere espresso sia come vincolo di colonna che come vincolo di tabella, nel secondo solo come vincolo di tabella. Ad esempio in una ipotetica tabella Negozi se si vuol specificare che i nomi dei negozi sono unici:

```
CREATE TABLE Negozi
(...
NomeNeg CHAR (40) UNIQUE,
...)
```

in questo caso tramite UNIQUE utilizzato come vincolo di colonna si specifica che il NomeNeg deve essere unico all'interno della colonna.

Invece, si poteva arrivare allo stesso risultato utilizzandolo come vincolo di tabella :

[21] Cupellini Testori Vaggi :Il linguaggio SQL

```
CREATE TABLE Negozi
(...
NomeNeg CHAR (40)
...
 UNIQUE (NomeNeg)
...)
```

Se si volesse controllare che non esistano due negozi con lo stesso nome nella stessa città si dovrà scrivere :

```
CREATE TABLE Negozi
(...
NomeNeg CHAR (40)
NomeCittà CHAR (20)
....
 UNIQUE (NomeNeg, NomeCittà)
...)
```

Quindi, si avrà che nella tabella Negozi non possono esistere duplicazioni dell'insieme dei valori delle colonne NomeNeg e NomeCittà.
Il vincolo UNIQUE permette l'esistenza di più valori NULL perché il valore NULL non viene considerato come un vero e proprio valore.
Il vincolo PRIMARY KEY permette di identificare la chiave primaria di una tabella. L'effetto che si ottiene è quello di una combinazione dei vincoli NOT NULL e UNIQUE, con la differenza che PRIMARY KEY, poiché individua la chiave primaria di una tabella, può essere riferito ad una sola colonna o combinazione di colonne. Nel primo caso può essere utilizzato sia come vincolo di colonna che come vincolo di tabella, nel secondo solo come vincolo di tabella. Esempio :

```
CREATE TABLE Articoli
(Art_cod CHAR (4) PRIMARY KEY,
...)

CREATE TABLE Articoli
(Art_cod CHAR (4)
...
PRIMARY KEY (Art_cod),
...)

CREATE TABLE Articoli
(Ord_cod CHAR (6)
Art_cod CHAR (4)
...
PRIMARY KEY (Art_cod, Ord_cod),
...)
```

4.7.7. La regola di integrità referenziale (vincolo di chiave esterna)

Nel modello relazionale la regola dell'integrità referenziale gestisce le associazioni fra le tabelle, stabilendo che se una tabella contiene una chiave esterna che la lega ad un'altra tabella allora ogni valore contenuto nella chiave esterna deve avere lo stesso valore che ha nella tabella in cui è chiave primaria oppure avere valore NULL. (vedere cap. 3). Prendiamo il caso del legame tra due tabelle dette *padre* e *figlio.* Il legame viene realizzato perché nella tabella figlio è stata posta una chiave esterna.. Il linguaggio SQL permette di definire la regola di integrità referenziale attraverso opportune clausole che vengono dichiarate al momento della creazione della tabella figlio[22].

[22] "E' tecnicamente possibile referenziare anziché una chiave primaria una colonna su cui sia stato definito il vicolo UNIQUE" Guidi Dorbolò : Guida all'SQL

La sintassi del vincolo di chiave esterna nella tabella figlio può essere espresso tramite o un vincolo di colonna o un vincolo di tabella.

Vincolo di chiave esterna di colonna	Esempi
	CREATE TABLE PF
REFERENCES tabella padre [(colonna padre)	(...
[{,colonna padre}...]	CodP CHAR (2) REFERENCES P (CodP)
[MATCH {FULL | PARTIAL }]	...)
[azione innescata]	

Vincolo di chiave esterna di tabella	
	CREATE TABLE PF
FOREIGN KEY (colonne figlio) REFERENCES	(...
tabella padre [(colonna padre) [{,colonna padre}...]	CodP CHAR (2)
[MATCH {FULL | PARTIAL }]	...
[azione innescata]	FOREIGN KEY (Codp) REFERENCES P (CodP)
	...)

L*'azione innescata* indica al DBMS cosa deve fare quando viene violata una delle condizioni del vincolo di chiave esterna dovuto ad una operazione di aggiornamento o di cancellazione.

La sintassi è :

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

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

Può essere specificata indifferentemente prima la clausola ON UPDATE oppure la clausola ON DELETE.

Nel caso ON DELETE l'azione attivata dalla cancellazione delle righe nella tabella padre viene eseguita sulle righe corrispondenti della tabella figlio e

con un effetto che è diverso a seconda dell'impiego di **CASCADE | SET NULL | SET DEFAULT | NO ACTION.**

CASCADE	le righe della tabella figlio vengono anche loro cancellate
SET NULL	le colonne figlio vengono poste a NULL
SET DEFAULT	le colonne figlio vengono impostate secondo il valore di default definito per la colonna
NO ACTION	il sistema non innesca alcuna azione speciale. Il DBMS rifiuta semplicemente di effettuare cancellazioni sui dati che violino l'integrità referenziale.

Nel caso ON DELETE l'azione attivata dall'aggiornamento delle righe nella tabella padre viene eseguita sulle righe corrispondenti della tabella figlio e con un effetto che è diverso a seconda dell'impiego di **CASCADE | SET NULL | SET DEFAULT | NO ACTION.**

CASCADE	le righe della tabella figlio vengono impostate allo stesso valore assegnato alle righe della colonna padre
SET NULL	le colonne figlio vengono poste a NULL
SET DEFAULT	le colonne figlio vengono impostate secondo il valore di default definito per la colonna
NO ACTION	il sistema non innesca alcuna azione speciale. Il DBMS rifiuta semplicemente di effettuare cancellazioni sui dati che violino l'integrità referenziale.

E' importante notare che se si usano le azioni innescate, le azioni su di una tabella possono avere effetti in cascata anche sulle altre tabelle.

4.7.8. Il vincolo di controllo

Il vincolo di controllo viene utilizzato per verificare particolari condizioni sui valori di una colonna.

La sintassi è :

CHECK (condizione)

La condizione può essere espressa dai normali operatori di confronto, dai connettori logici, dagli operatori BETWEEN, IN, LIKE. Il vincolo può essere espresso sia come vincolo di colonna, e quindi i valori coinvolti nella

condizione dovranno riguardare la sola colonna, oppure può essere espresso tramite il vincolo di tabella. In questo caso i valori coinvolti nella condizione potranno riguardare più colonne.

Come vincolo di colonna

```
CREATE TABLE Articoli
(...
Spese di trasporto NUMERIC (9) CHECK (Spese di trasporto < 200000),
...)
```

Come vincolo di tabella

```
CREATE TABLE Articoli
(...
Spese di trasporto NUMERIC (9)
Prezzo NUMERIC (9)
...
CHECK (Spese di trasporto < Prezzo),
...)
```

Qualora i vincoli di controllo riguardino le colonne di tabelle diverse il linguaggio SQL permette di utilizzare le **ASSERTION.** Si crea, così, un vincolo di integrità tenuto separato dalle tabelle a cui è riferito.

```
CREATE ASSERTION nome dell'assertion
CHECK (condizione)
[attributi del vincolo]
```

```
CREATE ASSERTION Controlla i prezzi
CHECK (SELECT MAX (Prezzo) FROM Articoli>
        SELECT MAX (Costo) FROM Componenti)
```

4.7.9. Creazione di un dominio

Il linguaggio SQL permette la realizzazione di domini. Un dominio è un tipo di dato che può essere utilizzato nelle tabelle al posto dei tipi di dato "classici" da associare alle singole colonne. Però la maggior parte dei DBMS non supporta la sintassi SQL standard adottando propri sistemi.

Il dominio è utile sia per definire una sola volta un insieme dati, per cui si farà sempre riferimento al dominio quando si vorrà impiegare questo insieme di dati senza doverli riscrivere tutte le volte, sia perché in caso di modifica del dominio tutte le colonne che lo utilizzano risultano automaticamente aggiornate.

Una caratteristica dei domini è che non possono essere ciclici e cioè non possono far riferimento al loro interno ad altri domini ne possono far riferimento a colonne. La sintassi standard, che però come già detto sopra viene in pratica disattesa da quasi tutti gli attuali DBMS, è :

CREATE DOMAIN nome vincolo AS tipo di dati
[clausola di default]
[[CONSTRAINT nome vincolo]
vincolo di controllo
[attributi del vincolo]...]

4.7.10. I comandi di eliminazione

Tramite il comando **DROP** è possibile realizzare l'eliminazione di uno schema, o di un dominio o do una tabella o di una assertion. La sintassi è :

DROP SCHEMA nome schema {RESTRICT | CASCADE}
DROP DOMAIN nome dominio {RESTRICT | CASCADE}
DROP TABLE nome tabella {RESTRICT | CASCADE}
DROP ASSERTION nome assertion

Le opzioni RESTRICT e CASCADE sono alternative ed obbligatorie. Con la prima si effettuerà l'eliminazione dello schema o della tabella solo se è vuota e, nel caso del dominio, solo se non è stato mai utilizzato nella definizione di alcun oggetto del dello schema. Con la seconda opzione si effettuerà comunque l'eliminazione. In caso di DROP ASSERTION non è previsto l'impiego delle suddette opzioni.

4.7.11. I comandi di modifica ALTER DOMAIN e ALTER TABLE

Attraverso i comandi ALTER DOMAIN e ALTER TABLE è possibile modificare i domini e le tabelle. Il linguaggio SQL non permette invece una

modifica per assertion e per schema. In particolare ci si soffermerà sul comando ALTER TABLE .

Questo permette di modificare alcune caratteristiche di una tabella . E' possibile :

1. aggiungere ed eliminare colonne
2. aggiungere ed eliminare vincoli di tabella
3. assegnare ed eliminare vincoli di default

Il comando ha forme sintattiche diverse a seconda della funzione svolta.

1. aggiungere ed eliminare colonne

ALTER TABLE nome tabella
ADD [COLUMN] definizione di colonna

Il comando permette di aggiungere un nuova colonna alla tabella. La "definizione di colonna" è uguale a quella del comando CREATE TABLE.

ALTER TABLE nome tabella
DROP [COLUMN] definizione di colonna {RESTRICT | CASCADE}

Il comando permette di eliminare una colonna dalla tabella. Le opzioni RESTRICT e CASCADE sono obbligatorie ed alternative. Con RESTRICT la colonna viene eliminata solo se non è *referenziata* in altre definizioni (e cioè nella stessa tabella, in altre tabelle o altri oggetti del database).

Se viene utilizzata l'opzione CASCADE invece, oltre a essere eliminata la colonna definita nel comando ALTER TABLE, vengono eliminate tutte le altre dipendenze nelle altre definizioni dello schema.

2. aggiungere ed eliminare vincoli di tabella

ALTER TABLE nome tabella ADD vincolo tabella

Il comando permette di aggiungere un nuovo vincolo di tabella a quelli esistenti. La sintassi per il "vincolo di tabella" è lo stesso di quello descritta per la creazione tabella. Il vincolo che viene creato è valido sia per i vecchi

dati sia per i nuovi. Se questi dati non fossero compatibili il comando ALTER TABLE fallirebbe. Se si vuole invece eliminare un vincolo avremo :

ALTER TABLE nome tabella

DROP nome vincolo {RESTRICT | CASCADE}

Il comando permette di eliminare un vincolo di tabella. Le opzioni RESTRICT e CASCADE sono alternative ed obbligatorie. Se si specifica RESTRICT non sarà possibile eliminare i vincoli di unicità su di una colonna se esistono su questa colonna vincoli di chiave esterna. L'opzione CASCADE permette sempre di effettuare una eliminazione.

3. assegnare ed eliminare vincoli di default

ALTER TABLE nome tabella

ALTER [COLUMN] nome colonna

SET clausola di default

Il comando permette di aggiungere un valore di default a una colonna. Se era già stato definito il valore di default, questo viene sostituito dal nuovo.

ALTER TABLE nome tabella

ALTER [COLUMN] nome colonna DROP DEFAULT

il comando permette di eliminare il valore di default per una colonna.

4.8. Gestione degli indici

Un indice è un struttura di dati ausiliaria che il DBMS associa alle tabelle per poter eseguire con maggior efficienza la ricerca dei dati. Ciascun indice è relativo una o più colonne di una tabella[23].

L'indice di una tabella relativo ad una colonna fa si che, per trovare un valore di una colonna, non si debba effettuare la scansione sequenziale di tutta la tabella con evidente risparmio di tempo di accesso ai dati della tabella. Le

[23] Guidi Dorbolò :Guida a SQL

operazioni che controllano l'uso degli indici non sono state oggetto di standardizzazione . Però si possono individuare nei DBMS alcune linee comuni :

- Il DBMS crea automaticamente un indice per chiave primaria di ogni tabella e spesso per ogni colonna su cui sia definito un vincolo di unicità.
- Esiste tutta una serie di comandi che consente di creare o di cancellare esplicitamente un indice.

4.9. Le viste logiche

Le viste logiche o *view* possono essere definite come *tabelle virtuali* poiché non contengono dati propri, ma sono costituite da una nuova aggregazione dei dati contenuti nelle tabelle (dette tabelle di base). Le view sono quindi indipendenti dalle tabelle anche se logicamente in relazione con esse ed hanno una loro struttura all'interno dello schema logico del database. Questa struttura, infatti, non viene definita in maniera indipendente dagli altri oggetti del database, ma in base alla struttura delle tabelle già presenti. Sono solo le tabelle che contengono i dati, mentre le view forniscono solo una visione aggiornata di quegli stessi dati[24].

La view può essere definita su una o più tabelle. Se è definita su di una sola tabella la view non fa altro che rendere visibile una selezione di dati, mentre se si basa su più tabelle si utilizzerà le tecniche delle join o delle subquery ottenendo in questo modo una nuova aggregazione dei dati che meglio si confà alle esigenze dell'utilizzatore. Essa appare all'utente come una normale tabella che può essere usata per compiere normali operazioni sui database come la ricerca e in certi casi l'aggiornamento (occorre che la view sia *aggiornabile*). I dati modificati da una vista risulteranno modificati anche nelle tabelle su cui la vista è definita e allo stesso modo le modifiche

[24] Una tabella virtuale (o view), è una tabella calcolata con una espressione SQL da altre tabelle sia di base che virtuali. Essa è virtuale nel senso che non corrisponde a dati fisicamente esistenti, ma denota dei dati ricavabili da altri secondo l'espressione usata per definire la tabella. L'espressione viene valutata ogni volta che si opera sulla tabella stessa." Albano :Basi di dati - Boringheri

effettuate sulle tabelle avranno come effetto una modifica dei dati visibili dalla vista.

Le view vengono definite tramite il comando DDL CREATE VIEW attraverso l'utilizzo di una particolare sintassi che qui si tralascia[25].

Le view permettono di vedere i dati del database *con livelli di astrazione* superiore rispetto alle tabelle realizzando tutta una serie di vantaggi :

- **Facilitazione nell'accesso ai dati**. Le tabelle "normalizzate" secondo lo schema logico relazionale possono talvolta far perdere all'utilizzatore una visione d'insieme che egli ha dei dati, per cui realizzando queste view che non intaccano però lo schema logico dal database, si può recuperare questa visione.
- **Diverse visioni dei dati**. Le viste costituiscono gli schemi esterni o i sottoschemi della architettura di un DBMS. Quindi, è possibile limitare la vista che un utente ha di una o più tabelle, operando una restrizione sull'insieme delle colonne e delle righe.
- **Indipendenza logica**. Le view realizzano l'indipendenza logica delle applicazioni e delle operazioni eseguite dagli utenti della struttura logica dei dati. Si possono fare, cioè, delle modifiche dello schema senza dover portare modifiche alle applicazioni che lavorano sui dati che sono stati modificati o alle operazioni che gli utenti compiono su quei dati.

4.10. Gestione delle transazioni

Il linguaggio SQL si occupa anche di gestire, tramite dei comandi appositi (COMMIT e ROLLBACK), le transazioni che si verificano su di una base di dati. Nel primo capitolo si è definita la transazione come una sequenza di operazioni indivisibile, per cui una transazione non può avere effetto parziale : o tutte le operazioni che la riguardano vanno a buon fine oppure nessuna. Quindi, una transazione può terminare in due modi :

? con un successo e di conseguenza la base di dati passa ad un nuovo stato (*commit*)

? con un fallimento e di conseguenza è come se per il sistema non fosse mai iniziata (*abort*)

Le tecniche transazionali vengono utilizzate per garantire l'integrità fisica dei dati (e non quella logica che riguarda i vincoli di integrità di tabella e di colonna). Gli eventi che mettono a rischio tale integrità sono i **malfunzionamenti hardware** e **software** del sistema e **l'accesso concorrente ai dati**. Quindi, questi sono problemi che riguardano la congruenza di stato ed in particolare il recupero degli errori e la concorrenza di accesso. Nel caso specifico di accesso concorrente, l'SQL standard non prevede un uso particolare di operazioni di lock ma permette di definire in maniera implicita diverse modi di realizzare il lock.

Il comando utilizzato in questi casi è il COMMIT , il quale comunica al sistema che la transazione è terminata. Questo comando si usa alla chiusura di una sessione o di un programma e ha come scopo quello di confermare tutte le operazioni fatte con comandi SQL. Le modifiche fatte a questo punto sono a disposizione di tutti gli altri utenti. Al contrario, il comando ROLLBACK comunica al sistema relazionale che la transazione è terminata con l'annullo di tutte le operazioni effettuate dai comandi SQL. Lo stato della base dei dati viene riportato a quello esistente all'inizio della transazione, cioè all'ultimo COMMIT valido[26].

4.11. La sicurezza

Per sicurezza s'intende la protezione della base di dati da accessi non autorizzati, volontari o accidentali. In un DBMS esistono tecniche per garantire la sicurezza che possono essere così raggruppate :

1. identificazione degli utenti tramite un parola chiave.

[25] Per approfondimenti vedere Guidi Dorbolò :Guida a SQL e Cupellini Testori Vaggi :Il linguaggio SQL.

2. accesso ai dati tramite view
3. assegnazione di privilegi agli utenti sugli oggetti del database
4. crittografia del database.

Il linguaggio SQL si occupa solo della seconda e terza tecnica attraverso la definizione delle view e la assegnazione o revoca dei privilegi sugli oggetti della base di dati. Il proprietario di un oggetto ha tutti i diritti possibili sull'oggetto, mentre gli altri dispongono dei diritti di accesso sull'oggetto concessi, in maniera diretta o indiretta, dal proprietario[27]. Per oggetto in una base di dati si intende :

- un dominio
- una tabella
- una colonna
- un set di caratteri

I possibili tipi di privilegi sugli oggetti sono quelli relativi all'inserimento, modifica, cancellazione ecc..

Questi privilegi vengono gestiti attraverso una matrice. I comandi per l'assegnazione dei privilegi e per la loro revoca sono GRANT e REVOKE[28].

[26]Per la sintassi dei comandi vedere Guidi Dorbolò :Guida a SQL e Cupellini Testori Vaggi :Il Linguaggio SQL.

[27]" Si noti che quando si crea uno schema si deve determinare il proprietario dello schema stesso che solitamente è l'amministratore della base di dati. Sarà questo a concedere o a togliere a tutti gli altri utenti i privilegi sugli oggetti della base di dati". Ullman

[28]Per la sintassi dei comandi vedere Guidi Dorbolò :Guida a SQL e Cupellini Testori Vaggi :Il Linguaggio SQL.