Il linguaggio SQL

Angelo Montanari

Dipartimento di Scienze Matematiche,
Informatice e Fisiche
Università degli Studi di Udine

Introduzione

SQL (Structured Query Language) = DDL + DML

Il linguaggio SQL supporta l'interrogazione, la definizione e l'aggiornamento dei dati.

SQL come linguaggio di interrogazione:

- è più **dichiarativo** dell'algebra relazionale;
- (in prima approssimazione) può essere visto come una **variante sintattica** del calcolo relazione su tuple con dichiarazioni di range.

Flessibilità vs. espressività del linguaggio SQL (esempio: uso delle *condizioni di join* vs. uso di *interrogazioni annidate* e dell'operatore *in*).

SQL vs. algebra/calcolo relazionale

Due elementi importanti di differenziazione rispetto all'agebra/calcolo relazionale:

(1) i risultati delle proiezioni sono **multi-insiemi** anziché insiemi (di default i duplicati non vengono rimossi);

(2) **valori** vs. relazioni (il risultato dell'applicazione di una funzione aggregata può, in specifici contesti, essere trattato come un valore e non come una relazione).

Inquadramento

L'attuale SQL è lo sviluppo di un linguaggio di interrogazione per il DBMS relazionale **System R**, proposto originariamente presso i laboratori di ricerca dell'IBM nella seconda metà degli anni settanta (SEQUEL - Structured English QUEry Language).

A partire dagli anni ottanta, è stato oggetto di un'intensa attività di **standardizzazione**, svolta prevalentemente nell'ambito degli organismi ANSI (American National Standards Institute, l'organismo nazionale statunitense che si occupa di standard) e ISO (l'organismo internazionale che coordina i vari organismi nazionali). Il processo di standardizzazione è sempre in corso.

Vantaggi della standardizzazione: **portabilità** e **uniformità** nell'accesso a basi di dati diverse.

Evoluzione dello standard SQL - 1

Le principali tappe del processo di standardizzazione di SQL possono essere riassunte nel modo seguente:

(1) SQL-86 (SQL base), prodotto nel 1986 dall'ANSI, contiene tutti i costrutti base del linguaggio di interrogazione, ma offre un supporto limitato alla definizione e all'aggiornamento dei dati;

(2) SQL-89 (SQL base), estensione limitata di SQL-86, emanata nel 1989, introduce la nozione di integrità referenziale;

(3) SQL-92 (**SQL-2**), pubblicato nel 1992, in larga misura compatibile con SQL-89, introduce numerose nuove funzionalità;

Evoluzione dello standard SQL - 2

(4) SQL:1999 (**SQL-3**), pienamente compatibile con SQL-2, introduce delle estensioni ad oggetti, vari nuovi costrutti e nuovi servizi quali trigger e viste ricorsive;

(5) SQL:2003, SQL:2006 e SQL:2011 introducono ulteriori estensioni ad oggetti, rimuovono alcuni costrutti non utilizzati degli standard precedenti e includono nuove parti quali SQL/JRT (integrazione di SQL col linguaggio Java) e SQL/XML (gestione dei dati XML).

A differenza di SQL-2, SQL-3 è organizzato in diverse parti opportunamente numerate e denominate (ad esempio, la parte 13 è quella relativa a SQL/JRT, la parte 14 quella relativa a SQL/XML). Ogni parte è prodotta da uno specifico comitato tecnico e può essere rinnovata in modo indipendente dalle altre.

Standard: core ed estensioni

A partire dall'SQL:1999, lo standard è suddiviso in un **core** e in un insieme di **estensioni specializzate**.

Tutti i DBMS SQL-compliant mettono (dovrebbero mettere) a disposizione il core; le estensioni possono essere implementate come moduli addizionali per specifiche applicazioni, quali:

- data mining
- dati geografici e/o temporali
- data warehousing
- OLAP (OnLine Analytical Processing)
- dati multimediali
- ...

DBMS e standard SQL

I diversi DBMS disponibili presentano delle piccole differenze nell'implementazione del linguaggio SQL, specie rispetto alle funzionalità più innovative.

L'uniformità è molto maggiore per quanto riguarda le funzionalità più consolidate.

Molti sistemi mettono a disposizione **interfacce amichevoli** per l'interrogazione, la definizione e l'aggiornamento dei dati (ad esempio, interfacce di natura visuale).

Tutti consentono, comunque, di utilizzare in modo **esplicito** (e completo) il linguaggio SQL.

Il linguaggio di interrogazione SQL

Un'interrogazione SQL può contenere fino a **6 clausole**, delle quali solo le prime 2 sono obbligatorie:

SELECT ⟨ lista di attributi ⟩ (target list)
FROM ⟨ lista di tabelle ⟩
WHERE ⟨ condizione ⟩
GROUP BY ⟨ lista di attributi (di raggruppamento) ⟩
HAVING ⟨ condizione (di raggruppamento) ⟩
ORDER BY ⟨ lista di attributi ⟩

Le prime 3 clausole costituiscono il **blocco fondamentale** (detto mapping).

Il blocco fondamentale

SELECT specifica gli attributi e/o le funzioni (in generale, le espressioni) il cui valore viene restituito dalla valutazione dell'interrogazione (schema della relazione risultato). Può essere specializzata in più modi (ad esempio, uso della parola chiave DISTINCT).

FROM specifica tutte le relazioni cui occorre accedere per recuperare l'informazione richiesta dall'interrogazione (non quelle coinvolte in eventuali interrogazioni nidificate).

WHERE specifica le condizioni (espressioni Booleane) per la selezione delle tuple delle relazioni indicate nella clausola FROM (condizioni di selezione e condizioni di join; in SQL-2 è possibile spostare le condizioni di join nella clausola FROM).

Le interrogazioni di base

Il **blocco fondamentale**: SELECT-FROM-WHERE

Sintassi:

SELECT *AttrEspr* [[*AS*]*ALIAS*] {, *AttrEspr* [[*AS*]*ALIAS*]}
FROM *Tabella* [[*AS*]*ALIAS*] {, *Tabella* [[*AS*]*ALIAS*]}
WHERE *Condizione*

Nel caso più semplice, l'esecuzione di tale blocco consiste nella PROIEZIONE sugli attributi specificati nella clausola SELECT del risultato della SELEZIONE delle tuple appartenenti al PRODOTTO CARTESIANO delle relazioni specificate nella clausola FROM in base alla condizione specificata nella clausola WHERE.

Uso degli alias e uso dell'asterisco

Esempio 1. Trovare lo stipendio degli impiegati il cui cognome è Bianco.

```
SELECT    STIPENDIO AS STIPENDIOBIANCO
FROM      IMPIEGATO
WHERE     COGNOME = 'BIANCO'
```

Esempio 2. Recuperare tutta l'informazione relativa agli impiegati il cui cognome è Bianco.

```
SELECT    *
FROM      IMPIEGATO
WHERE     COGNOME = 'BIANCO'
```

Uso delle espressioni e operazioni di join

Esempio 3. Trovare lo stipendio mensile degli impiegati il cui cognome è Bianco.

```
SELECT    STIPENDIO / 12 AS STIPENDIOMENSILE
FROM      IMPIEGATO
WHERE     COGNOME = 'BIANCO'
```

Esempio 4. Per ogni impiegato, recuperare il nome del dipartimento per cui lavora.

```
SELECT    CF, DNOME
FROM      IMPIEGATO, DIPARTIMENTO
WHERE     DIP = DNUMERO
```

Sottoblocco SELECT-FROM e uso di DISTINCT

Esempio 5. Recuperare lo stipendio di ogni impiegato.

```
SELECT    CF, STIPENDIO
FROM      IMPIEGATO
```

Esempio 6. Determinare le diverse fasce di stipendio degli impiegati dell'azienda.

```
SELECT    DISTINCT STIPENDIO
FROM      IMPIEGATO
```

Notazione puntata e nomi di attributo ambigui

Esempio 7. Per ogni impiegato, identificato da nome e cognome, recuperare i numeri dei progetti a cui lavora.

```
SELECT    IMPIEGATO.CF, IMPIEGATO.NOME,
          IMPIEGATO.COGNOME, L.PROGETTO
FROM      IMPIEGATO, LAVORA_A AS L
WHERE     CF = IMP
```

Esempio 8. Recuperare nome e data di nascita delle persone a carico di ogni impiegato.

```
SELECT    CF, P.NOME, P.DATA_NASCITA
FROM      IMPIEGATO, PERSONA_A_CARICO P
WHERE     CF = IMP
```

Alias e operazione di rinomina

Esempio 9. Recuperare nome e cognome dei supervisori degli impiegati che lavorano per il dipartimento 10.

```
SELECT    S.NOME, S.COGNOME
FROM      IMPIEGATO AS  I  S
WHERE     I.DIP = 10 AND I.SUPERVISORE = S.CF
```

Esempio 10. Trovare i dipartimenti che hanno almeno una sede in comune.

```
SELECT    DISTINCT S1.DNUMERO, S2.DNUMERO
FROM      SEDI_DIPARTIMENTO  S1  S2
WHERE     S1.DSEDE = S2.DSEDE AND
          S1.DNUMERO < S2.DNUMERO
```

N.B. DISTINCT opera a livello di tuple (non di singola componente) e non potrebbe essere diversamente in quanto SQL è orientato alle tuple

Espressioni Booleane nella clausola WHERE

Esempio 11. Recuperare nome e cognome degli impiegati di sesso maschile che guadagnano più di 40000 euro (si assuma che non vi siano omonimie).

```
SELECT    NOME, COGNOME
FROM      IMPIEGATO
WHERE     SESSO = 'M' AND STIPENDIO > 40000
```

Esempio 12. Determinare gli impiegati di cognome BIANCO che lavorano per il dipartimento 2 o il dipartimento 3.

```
SELECT    CF
FROM      IMPIEGATO
WHERE     COGNOME = 'BIANCO' AND (DIP = 2 OR
          DIP = 3)
```

Le operazioni insiemistiche - 1

Le **operazioni insiemistiche**: UNION (unione), EXCEPT (differenza insiemistica) e INTERSECT (intersezione).

I **duplicati** vengono rimossi, a meno che non venga richiesto in modo esplicito di mantenerli (uso della parola chiave ALL).

Esempio 13. Selezionare i nomi di tutti gli impiegati e di tutte le persone a carico.

```
SELECT    NOME
FROM      IMPIEGATO
UNION
SELECT    NOME
FROM      PERSONA_A_CARICO
```

Le operazioni insiemistiche - 2

Se gli attributi hanno nome diverso, l'unione può comunque essere effettuata e il risultato normalmente usa i nomi del primo operando (il nome degli attributi della relazione risultato può dunque variare al variare dell'ordine degli operandi).

Esempio 14. Selezionare i nomi e i cognomi di tutti gli impiegati.

```
SELECT    NOME
FROM      IMPIEGATO
UNION
SELECT    COGNOME
FROM      IMPIEGATO
```

Le operazioni insiemistiche - 3

Esempio 15. Selezionare i nomi e i cognomi di tutti gli impiegati del Dipartimento numero 10, mantenendo gli eventuali duplicati.

```
SELECT    NOME
FROM      IMPIEGATO
WHERE     DIP = 10
UNION ALL
SELECT    COGNOME
FROM      IMPIEGATO
WHERE     DIP = 10
```

Le operazioni insiemistiche - 4

Esempio 16. Selezionare i cognomi degli impiegati che sono anche nomi (di qualche impiegato).

```
SELECT    COGNOME
FROM      IMPIEGATO
INTERSECT
SELECT    NOME
FROM      IMPIEGATO
```

Esempio 17. Selezionare i nomi degli impiegati che non sono anche cognomi (di qualche impiegato).

```
SELECT    NOME
FROM      IMPIEGATO
EXCEPT
SELECT    COGNOME
FROM      IMPIEGATO
```

Le interrogazioni nidificate

Interrogazioni nidificate: nella clausola WHERE, SQL consente di confrontare, mediante i normali operatori di confronto, un valore (eventualmente ottenuto quale risultato di un'espressione valutata su una singola tupla) col risultato dell'esecuzione di una interrogazione (nidificata) SQL.

Nel caso più semplice tale valore è il **valore di uno specifico attributo**.

Come confrontare un singolo valore con un insieme di valori (risultato della valutazione di un'interrogazione)? Uso delle parole chiave ANY (equivalentemente SOME) e ALL.

L'uso delle parole chiave ANY e ALL

Come vengono valutate le interrogazioni nidificate? Una valutazione per ogni tupla, o combinazione di tuple, dell'interrogazione cui appartengono.

La **parola chiave ANY**: la tupla soddisfa la condizione se il confronto (con l'operatore specificato) tra il valore che l'attributo/i assume sulla tupla e almeno uno degli elementi restituiti dalla valutazione dell'interrogazione nidificata risulta VERO.

La **parola chiave ALL**: la tupla soddisfa la condizione se il confronto (con l'operatore specificato) tra il valore che l'attributo/i assume sulla tupla e ciascuno degli elementi restituiti dalla valutazione dell'interrogazione nidificata risulta VERO.

Nota Bene: è richiesta la compatibilità di dominio tra l'attributo/i restituito dall'interrogazione e l'attributo/i con cui avviene il confronto.

Esempi d'uso della parola chiave ANY

Esempio 18. Selezionare gli impiegati che afferiscono ad un dipartimento con una sede a Pordenone.

```
SELECT    *
FROM      IMPIEGATO
WHERE     DIP = ANY ( SELECT  DNUMERO
                      FROM    SEDI_DIPARTIMENTO
                      WHERE   DSEDE = 'PORDENONE')
```

Esempio 19. Selezionare il codice fiscale degli impiegati che hanno lo stesso cognome di un impiegato che lavora per il dipartimento 10.

```
SELECT    CF
FROM      IMPIEGATO
WHERE     COGNOME = ANY ( SELECT  COGNOME
                          FROM    IMPIEGATO
                          WHERE   DIP = 10)
```

Esempi d'uso della parola chiave ALL

Esempio 20. Selezionare tutti i dipartimenti nei quali non lavora alcun impiegato che guadagna più di 80000 euro.

```
SELECT    DNUMERO
FROM      DIPARTIMENTO
WHERE     DNUMERO <> ALL ( SELECT  DIP
                           FROM    IMPIEGATO
                           WHERE   STIPENDIO > 80000)
```

Esempio 21. Selezionare l'impiegato/i che percepisce lo stipendio massimo.

```
SELECT    CF
FROM      IMPIEGATO
WHERE     STIPENDIO >= ALL ( SELECT  STIPENDIO
                             FROM    IMPIEGATO)
```

Soluzioni alternative

Esempio 20'. Selezionare tutti i dipartimenti nei quali non lavora alcun impiegato che guadagna più di 80000 euro.

```
SELECT    DNUMERO
FROM      DIPARTIMENTO
EXCEPT
SELECT    DIP
FROM      IMPIEGATO
WHERE     STIPENDIO > 80000
```

Esempio 21'. Selezionare l'impiegato/i che percepisce lo stipendio massimo.

```
SELECT    CF
FROM      IMPIEGATO
EXCEPT
SELECT    I1.CF
FROM      IMPIEGATO AS  I1  I2
WHERE     I1.STIPENDIO < I2.STIPENDIO
```

L'operatore IN

Si noti che se un'interrogazione nidificata restituisce un unico valore, è indifferente usare la parola chiave ANY o la parola chiave ALL (la parola chiave può anche essere omessa).

Per controllare l'appartenza (rispettivamente, la non appartenenza) di un elemento/valore ad un insieme di elementi/valori restituiti da un'interrogazione nidificata, si possono utilizzare gli **operatori IN** (equivalente a = ANY) e **NOT IN** (equivalente a <> ALL).

Esempio 18'. Selezionare gli impiegati che afferiscono ad un dipartimento con una sede a Pordenone.

```
SELECT    *
FROM      IMPIEGATO
WHERE     DIP IN ( SELECT  DNUMERO
                   FROM    SEDI_DIPARTIMENTO
                   WHERE   DSEDE = 'PORDENONE')
```

Interrogazioni nidificate correlate e non correlate

In tutte le interrogazioni nidificate sin qui viste, l'interrogazione nidificata può essere **eseguita una sola volta** (prima di analizzare le tuple dell'interrogazione esterna): il risultato della valutazione dell'interrogazione nidificata **non dipende** dalla specifica tupla presa in esame dall'interrogazione esterna (interrogazione nidificata **non correlata** all'interrogazione esterna).

Talvolta, l'interrogazione nidificata deve poter far riferimento alle tuple delle tabelle dichiarate nell'interrogazione esterna: il risultato della valutazione dell'interrogazione nidificata **dipende** dalla specifica tupla presa in esame dall'interrogazione esterna.

Occorre un meccanismo per il *passaggio di binding*: attributi di tabelle dichiarate nell'interrogazione più esterna devono poter essere richiamati dall'interrogazione più interna.

Valutazione di interrogazioni nidificate correlate

Per ogni tupla dell'interrogazione più esterna,

prima viene valutata l'interrogazione nidificata,

poi viene valutata la condizione della clasuola WHERE dell'interrogazione più esterna (valutazione che coinvolge il risultato dell'interrogazione nidificata).

Tale processo può essere ripetuto un numero arbitrario di volte, pari al numero di interrogazioni nidificate l'una nell'altra in un'interrogazione (problemi di **leggibilità** delle interrogazioni).

Regole di visibilità (e uso degli alias)

Regole di visibilità (o scoping): gli attributi di una tabella sono utilizzabili solo nell'ambito dell'interrogazione in cui la tabella è dichiarata o di un'interrogazione nidificata (a qualsiasi livello) all'interno di essa.

Uso degli alias: ci possono essere delle ambiguità quando sono presenti più attributi con lo stesso nome, uno che compare in una tabella dichiarata nell'interrogazione esterna, uno che compare in una tabella dichiarata nell'interrogazione nidificata.

Regola di disambiguazione: un attributo non qualificato fa riferimento alla tabella dichiarata nell'interrogazione (nidificata) **più interna**. Per far riferimento ad attributi con lo stesso nome di tabelle dichiarate in interrogazioni più esterne occorre utilizzare degli **alias** (visti come variabili di tupla).

Violazione delle regole di visibilità

Interrogazione che non rispetta le regole di visibilità di SQL.

Esempio 22. Selezionare gli impiegati che afferiscono al dipartimento Ricerca o ad un dipartimento che ha una sede in (almeno) una città in cui ha una sede il dipartimento Ricerca.

```
SELECT    CF
FROM      IMPIEGATO
WHERE     DIP IN ( SELECT  DNUMERO
                   FROM    DIPARTIMENTO AS D1,
                           SEDI_DIPARTIMENTO AS S1
                   WHERE   D1.DNUMERO = S1.DNUMERO AND
                           D1.DNOME = 'RICERCA') OR
                   DIP IN ( SELECT  DNUMERO
                   FROM    SEDI_DIPARTIMENTO AS D2
                   WHERE   S1.DSEDE = D2.DSEDE)
```

Tale soluzione è scorretta perché la variabile di tupla S1 non è visibile nella seconda interrogazione nidificata. Trovare una soluzione corretta.

Interrogazioni nidificate correlate e uso di alias

Esempio 23. Determinare nome e cognome di tutti gli impiegati che hanno una persona a carico del loro sesso con lo stesso nome.

```
SELECT    NOME, COGNOME
FROM      IMPIEGATO  I
WHERE     CF IN ( SELECT  IMP
                  FROM    PERSONA_A_CARICO
                  WHERE   I.NOME = NOME AND I.SESSO = SESSO)
```

Esempio 24. Selezionare gli impiegati che percepiscono uno stipendio diverso da tutti gli altri impiegati del loro dipartimento.

```
SELECT    CF
FROM      IMPIEGATO AS  I
WHERE     STIPENDIO NOT IN ( SELECT  STIPENDIO
                             FROM    IMPIEGATO
                             WHERE   CF <> I.CF AND DIP = I.DIP)
```

La funzione Booleana EXISTS

SQL introduce una funzione Booleana (**funzione EXISTS**) che consente di verificare se il risultato di un'interrogazione nidificata correlata è vuoto o meno.

Relazione risultato non vuota: **EXISTS**

Relazione risultato vuota: **NOT EXISTS**

Dato che non siamo interessati alle specifiche tuple restituite dall'interrogazione nidificata, ma solo alla presenza/assenza di tuple, la clausola select dell'interrogazione nidificata assume di norma la forma: **SELECT** $*$.

Esempi d'uso di EXISTS e NOT EXISTS

Esempio 25. Selezionare gli impiegati che non hanno persone a carico.

```
SELECT    CF
FROM      IMPIEGATO
WHERE     NOT EXISTS ( SELECT  *
                       FROM    PERSONA_A_CARICO
                       WHERE   CF = IMP)
```

Esempio 26. Restituire il nome e il cognome dei manager che hanno almeno una persona a carico.

```
SELECT    NOME, COGNOME
FROM      IMPIEGATO
WHERE     EXISTS ( SELECT  *
                   FROM    DIPARTIMENTO
                   WHERE   CF = MANAGER) AND
          EXISTS ( SELECT  *
                   FROM    PERSONA_A_CARICO
                   WHERE   CF = IMP)
```

INTERSECT ed EXCEPT via EXISTS - 1

Dallo studio del calcolo relazionale su tuple con dichiarazioni di range sappiamo che gli operatori insiemistici di intersezione e differenza, a differenza dell'operatore di unione, non sono strettamente necessari.

Vediamo come possono essere definiti in SQL mediante la funzione Booleana EXISTS.

Siano date due relazioni $R(A, B)$ e $S(C, D)$.

Intersezione

```
SELECT   A, B
FROM     R
INTERSECT
SELECT   C, D
FROM     S
```

Differenza insiemistica

```
SELECT   A, B
FROM     R
EXCEPT
SELECT   C, D
FROM     S
```

INTERSECT ed EXCEPT via EXISTS - 2

Le due interrogazioni precedenti si possono riscrivere nel seguente modo:

```
SELECT    A, B
FROM      R
WHERE     EXISTS ( SELECT  *
                   FROM    S
                   WHERE   C = A AND D = B)

SELECT    A, B
FROM      R
WHERE     NOT EXISTS ( SELECT  *
                       FROM    S
                       WHERE   C = A AND D = B)
```

L'operatore CONTAINS

L'implementazione originaria di SQL (System R) prevedeva un operatore CONTAINS che consentiva di stabilire se un insieme era o meno contenuto in un altro.

Esempio 27. Trovare gli impiegati che lavorano a tutti i progetti controllati dal dipartimento 10.

```
SELECT    CF
FROM      IMPIEGATO
WHERE     ( SELECT PROGETTO
            FROM   LAVORA_A
            WHERE  CF = IMP)
          CONTAINS
          ( SELECT PNUMERO
            FROM   PROGETTO
            WHERE  DNUM = 10)
```

CONTAINS è stato successivamente rimosso da SQL.

Come esprimere l'operatore CONTAINS

Esempio 27'. Trovare gli impiegati che lavorano a tutti i progetti controllati dal dipartimento 10.

```
SELECT CF
FROM   IMPIEGATO
WHERE  NOT EXISTS ( SELECT  *
                    FROM    PROGETTO
                    WHERE   DNUM = 10
                            AND
                            NOT EXISTS ( SELECT *
                                         FROM   LAVORA_A
                                         WHERE  CF = IMP AND
                                                PNUMERO =
                                                PROGETTO))
```

La funzione Booleana UNIQUE

La funzione Booleana **UNIQUE** consente di verificare che nel risultato di un'interrogazione nidificata non siano presenti dei duplicati.

Esempio 28. Trovare gli impiegati che non hanno a carico due o più persone dello stesso sesso.

```
SELECT CF
FROM   IMPIEGATO
WHERE  UNIQUE ( SELECT  SESSO
                FROM    PERSONA_A_CARICO
                WHERE   CF = IMP)
```

Le operazioni di join

SQL-2 consente di specificare una tabella che si ottiene come risultato di un'operazione di join nella clausola FROM (separazione tra condizioni di selezione, nella clausola WHERE, e condizioni di join, nella clausola FROM).

Tipi di join:

- INNER JOIN (o semplicemente JOIN)
- LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN (la parola OUTER si può omettere)
- NATURAL JOIN
- NATURAL LEFT/RIGHT/FULL OUTER JOIN

INNER JOIN

Esempio 29. Recuperare nome, cognome e indirizzo degli impiegati che afferiscono al dipartimento Ricerca.

```
SELECT NOME, COGNOME, INDIRIZZO
FROM   (IMPIEGATO JOIN DIPARTIMENTO
        ON DIP = DNUMERO)
WHERE DNOME = 'RICERCA'
```

Esempio 30. Per ogni progetto con sede Tolmezzo, restituire il numero del progetto, il dipartimento che lo controlla e il cognome del manager di tale dipartimento.

```
SELECT PNUMERO, DNUMERO, COGNOME
FROM   ((PROGETTO JOIN DIPARTIMENTO ON DNUM = DNUMERO)
        JOIN IMPIEGATO ON MANAGER = CF)
WHERE PSEDE = 'TOLMEZZO'
```

OUTER JOIN

Esempio 31. Restituire nome e cognome di ogni impiegato e del relativo supervisore diretto (se esiste).

```
SELECT I.NOME AS NOMEIMP, I.COGNOME AS COGNOMEIMP,
       S.NOME AS NOMESUP, S.COGNOME AS COGNOMESUP
FROM   (IMPIEGATO AS I LEFT OUTER JOIN
       IMPIEGATO AS S ON I.SUPERVISORE = S.CF)
```

Esempio 32. Restituire tutti i dipartimenti con gli eventuali progetti da essi controllati.

```
SELECT PNUMERO, DNUMERO
FROM   (PROGETTO RIGHT OUTER JOIN DIPARTIMENTO
       ON DNUM = DNUMERO)
```

Funzioni aggregate in SQL - 1

SQL fornisce alcune funzioni built-in quali **COUNT**, **SUM**, **AVG**, **MAX** e **MIN** (alcune funzioni per il calcolo della varianza, mediana, etc. sono state aggiunte in SQL-99).

Esempio 33. Determinare la somma totale degli stipendi pagati dall'azienda ai suoi impiegati, lo stipendio medio, lo stipendio massimo e lo stipendio minimo.

```
SELECT SUM(STIPENDIO), AVG(STIPENDIO),
       MAX(STIPENDIO), MIN(STIPENDIO)
FROM   IMPIEGATO
```

Nell'esempio precedente, tutte le funzioni aggregate prendono in considerazione tutti i valori diversi dal valore nullo dell'attributo STIPENDIO (lo stesso effetto si otterrebbe utilizzando la parola chiave **ALL**).

Funzioni aggregate in SQL - 2

L'operatore COUNT consente di contare il numero di righe di una tabella (opzione $*$), il numero di valori distinti assunti dall'attributo/i cui viene applicata la funzione (parola chiave **DISTINCT**) e il numero di righe che assumono un valore diverso dal "valore" NULL sull'attributo/i cui viene applicata la funzione (parola chiave **ALL**).

Esempio 34. Determinare il numero di impiegati del dipartimento 3.

```
SELECT COUNT (*)
FROM   IMPIEGATO
WHERE DIP = 3
```

Esempio 35. Determinare il numero di fasce diverse di stipendio degli impiegati dell'azienda.

```
SELECT COUNT(DISTINCT STIPENDIO)
FROM   IMPIEGATO
```

Interrogazioni inconsistenti (mismatch)

Le funzioni aggregate non forniscono un meccanismo di selezione. Esse vengono applicate ad un insieme (di valori) e restituiscono un singolo valore.

Ne segue che la clausola SELECT non può contenere sia un attributo (o più), che genera un valore per ogni tupla selezionata, sia una funzione aggregata (o più), che restituisce un singolo valore per l'intero insieme di tuple.

Un esempio di **interrogazione inconsistente**:

```
SELECT NOME, COGNOME, MAX(STIPENDIO)
FROM   IMPIEGATO, SEDI_DIPARTIMENTO
WHERE DIP=DNUMERO AND DSEDE = 'TRIESTE'
```

Interrogazioni con raggruppamento

Funzioni aggregate e clausola **GROUP BY**, che consente di partizionare le tuple di una relazione.

Esempio 36. Per ogni dipartimento, determinare la somma degli stipendi degli afferenti.

```
SELECT    DIP, SUM(STIPENDIO)
FROM      IMPIEGATO
GROUP BY DIP
```

In presenza della clausola GROUP BY, la clausola SELECT può contenere solo funzioni aggregate e (un sottoinsieme de) gli attributi di raggruppamento

Interrogazioni sintatticamente scorrette

Esempio 37. Per ogni dipartimento, restituire il numero di afferenti e il manager (**soluzione scorretta**).

```
SELECT    DIP, COUNT(*), MANAGER
FROM      IMPIEGATO, DIPARTIMENTO
WHERE     DIP = DNUMERO
GROUP BY DIP
```

Esempio 37'. Per ogni dipartimento, restituire il numero di afferenti e il manager (**soluzione corretta**).

```
SELECT    DIP, COUNT(*), MANAGER
FROM      IMPIEGATO, DIPARTIMENTO
WHERE     DIP = DNUMERO
GROUP BY DIP, MANAGER
```

Predicati sui gruppi

La clausola HAVING può essere utilizzata per restringersi alle sole classi della partizione (indotta dagli attributi della clausola GROUP BY) che soddisfano una determinata condizione

Esempio 38. Selezionare tutti e soli i dipartimenti che spendono più di 1000000 di euro in stipendi per i loro afferenti (riportando tale spesa).

```
SELECT    DIP, SUM(STIPENDIO) AS TOTALESTIPENDI
FROM      IMPIEGATO
GROUP BY DIP
HAVING    SUM(STIPENDIO) > 1000000
```

Condizioni WHERE vs. condizioni HAVING - 1

Vengono valutate **prima** le condizioni specificate dalla clausola **WHERE**, per restringere l'insieme delle tuple; **poi**, una volta partizionato l'insieme di tuple così ottenuto, vengono valutate le condizioni specificate dalla clausola **HAVING** per eliminare alcune classi della partizione.

Esempio 39. Per ogni dipartimento con più di 5 afferenti, determinare il numero di afferenti che guadagnano più di 60000 euro (**soluzione scorretta**).

```
SELECT    DIP, COUNT(*)
FROM      IMPIEGATO
WHERE     STIPENDIO > 60000
GROUP BY DIP
HAVING    COUNT(*) > 5
```

Condizioni WHERE vs. condizioni HAVING - 2

Esempio 39'. Per ogni dipartimento con più di 5 afferenti, determinare il numero di afferenti che guadagnano più di 60000 euro (**soluzione corretta**).

```
SELECT    DIP, COUNT(*)
FROM      IMPIEGATO
WHERE     STIPENDIO > 60000 AND
          DIP IN ( SELECT    DIP
                   FROM      IMPIEGATO
                   GROUP BY DIP
                   HAVING    COUNT(*) > 5)
GROUP BY DIP
```

Relazioni come valori

Esempio 40. Determinare cognome e nome degli impiegati che hanno due o più persone a carico (usando le funzioni aggregate).

```
SELECT   COGNOME, NOME
FROM     IMPIEGATO
WHERE    (SELECT   COUNT(*)
          FROM     PERSONA_A_CARICO
          WHERE    CF = IMP) >= 2
```

Miscellanea - 1

Nei confronti col risultato di interrogazioni nidificate, si possono usare **tuple di attributi/valori** anziché singoli attributi/valori.

Esempio 41. Selezionare gli impiegati che dedicano ad un progetto lo stesso numero di ore che vi dedica l'impiegato MNTGVN89S14J324Q.

```
SELECT    DISTINCT IMP
FROM      LAVORA_A
WHERE     (PROGETTO, ORE_SETTIMANA) IN
                (SELECT    PROGETTO, ORE_SETTIMANA
                FROM       LAVORA_A
                WHERE      IMP = 'MNTGVN89S14J324Q')
```

Nella clausola WHERE, possono essere introdotti in modo **esplicito** insiemi di valori.

Esempio 42. Selezionare gli impiegati che lavorano per i progetti 1, 2 o 3.

```
SELECT    DISTINCT IMP
FROM      LAVORA_A
WHERE     PROGETTO IN {1, 2, 3}
```

Miscellanea - 2

Pattern matching su stringhe: l'**operatore LIKE**.

Vengono utilizzati due caratteri riservati: %, che rimpiazza un numero arbitrario (0 o più) di caratteri, e _, che rimpiazza un singolo carattere.

Esempio 43. Selezionare gli impiegati che risiedono in una (parte di) città con Codice di Avviamento Postale 33210.

```
SELECT    NOME, COGNOME
FROM      IMPIEGATO
WHERE     INDIRIZZO LIKE '%33210%'
```

Esempio 44. Selezionare gli impiegati nati negli anni sessanta.

```
SELECT    NOME, COGNOME
FROM      IMPIEGATO
WHERE     DATA_NASCITA LIKE '_ _ 6 _ _ _ _ _ _ _ _'
```

Miscellanea - 3

Se % o _ devono essere utilizzati come elementi di una stringa, ogni loro occorrenza deve essere preceduta da un **carattere di "escape"**, che viene specificato alla fine della stringa utilizzando la parola chiave ESCAPE.

Esempi: la sequenza 'AB_CD\%EF'ESCAPE'\' rappresenta la stringa 'AB_CD%EF' in quanto il carattere \ è specificato quale carattere di escape.

Per dati di tipo stringa, si può utilizzare l'**operatore di concatenazione** || per concatenare due stringhe.

Nei confronti, si può utilizzare l'**operatore BETWEEN** che consente di esprimere alcune condizioni in modo più compatto.

Esempio 45. Selezionare gli impiegati del dipartimento 3 che guadagnano tra i 30000 e i 40000 euro (estremi compresi).

```
SELECT    *
FROM      IMPIEGATO
WHERE     (STIPENDIO BETWEEN 30000 AND 40000) AND DIP=3
```

Miscellanea - 4

Le tuple appartenenti al risultato di un'interrogazione possono essere ordinate, in modo crescente o decrescente, sulla base di uno o più dei loro attributi usando la clausola ORDER BY.

Esempio 46. Restituire l'elenco degli impiegati e dei progetti ai quali lavorano, ordinati sulla base del dipartimento di afferenza (in modo decrescente) e, all'interno di ogni dipartimento, in ordine alfabetico (in modo crescente) rispetto a cognome e nome.

```
SELECT    DNOME, COGNOME, NOME, PNOME
FROM      IMPIEGATO, DIPARTIMENTO, PROGETTO, LAVORA_A
WHERE     DIP = DNUMERO AND CF=IMP AND
          PROGETTO = PNUMERO
ORDER BY DNOME DESC, COGNOME ASC, NOME ASC
```

L'ordine ascendente è il default (ASC può essere omessa nella clausola ORDER BY).

Miscellanea - 5

Gestione dei valori nulli: per selezionare le tuple che assumono (risp., non assumono) il valore NULL su un certo attributo, SQL fornisce il predicate **IS NULL** (risp., **IS NOT NULL**), che restituisce il valore vero se e solo se l'attributo ha valore nullo (risp., non ha valore nullo).

Sintassi: ATTRIBUTO IS [NOT] NULL

Esempio 47. Selezionare tutti gli impiegati dei quali non è noto lo stipendio.

```
SELECT    *
FROM      IMPIEGATO
WHERE     STIPENDIO IS NULL
```

Osservazione. SQL-89 usa la logica a 2 valori; SQL-2 la logica a tre valori (true, false, unknown).

Definizione dei dati in SQL

Elementi fondamentali:

- definizione dei domini
- definizione dello schema e delle tabelle
- definizione dei vincoli
- specifica dei valori di default
- vincoli intrarelazionali
- vincoli interrelazionali
- modifica degli schemi
- cataloghi relazionali

I domini

I domini specificano gli insiemi di valori che gli attributi possono assumere.

Distinguiamo:

- **domini elementari**, messi a disposizione da SQL
- **domini definiti dall'utente** (tramite i comandi per la definizione dei domini)

I principali **domini elementari**:

- i caratteri
- i tipi numerici esatti e quelli approssimati
- istanti e intervalli temporali
- domini introdotti in SQL-3

I caratteri

Singoli caratteri o **stringhe** di caratteri di **lunghezza** fissa o variabile (per le stringhe di lunghezza variabile, si indica la lunghezza massima).

Si può usare un insieme di caratteri diverso da quello di default (**alfabeto** latino, greco, cirillico, ..).

Sintassi: character [varying] [(lunghezza)]
[character set NomeFamigliaCaratteri]

Esempi.

stringa di 30 caratteri: *character(30)* (abbreviato *char(30)*)

stringa di al più 15 caratteri dell'alfabeto greco: *character varying (15)* (abbreviato *varchar2(15)*) *character set Greek*

singolo carattere: *character* (abbreviato *char*)

Il tipo Boolean e i tipi numerici esatti - 1

Boolean: permette di rappresentare **singoli** valori Booleani (vero o falso). Sostituisce il dominio *bit*, previsto da SQL-2, ma non implementato dai sistemi (e pertanto rimosso da SQL-3).

Tipi numerici esatti: valori esatti, **interi** o con una **parte decimale** di lunghezza prefissata

Sintassi (quattro alternative):

- numeric [(Precisione [, Scala])]
- decimal [(Precisione [, Scala])]
- integer
- smallint

Il tipo Boolean e i tipi numerici esatti - 2

Numeri in base decimale: domini **numeric** e **decimal**, dove

precisione = numero di cifre significative (se non viene specificata, si usa un valore caratteristico dell'implementazione),

scala = numero di cifre dopo la virgola (se non viene specificata, si assume uguale a 0).

Per numeric la precisione rappresenta un valore esatto, per decimal un requisito minimo

Esempio: *numeric(6,4)* consente di rappresentare i valori compresi tra -99,9999 e +99,9999

I domini **integer** e **smallint** si basano sulla rappresentazione interna binaria del calcolatore (la precisione è lasciata all'implementazione).

Tipi numerici approssimati

Usati per la rappresentazione di **valori reali approssimati**.

Sintassi (tre alternative):

- float [(Precisione)] (la *precisione* indica il numero di cifre dedicate alla mantissa; il numero di cifre dedicate all'esponente dipende dall'implementazione)
- real (precisione fissa)
- double precision (dimensione doppia rispetto a real)

Usano una rappresentazione in virgola mobile. Ciascun numero è descritto da una coppia di valori (mantissa ed esponente). La mantissa è un numero frazionario, l'esponente un numero intero.

Esempio: $0.17E16$ rappresenta il valore $1,7 \cdot 10^{15}$, $-0.4E-6$ rappresenta il numero $-4 \cdot 10^{-7}$.

Istanti temporali

Gli istanti temporali sono stati introdotti in SQL-2. Tre formati:

- date
- time [(Precisione)] [with time zone]
- timestamp [(Precisione)] [with time zone]

Il dominio **date** ammette i campi *year*, *month* e *day*; il dominio **time** i campi *hour*, *minute* e *second*; il dominio **timestamp** tutti i campi da year a *second*. La *precisione* indica il numero di cifre decimali da usare per rappresentare le frazioni di secondo.

L'opzione *with time zone* consente di accedere a due campi, *timezone_hour* e *timezone_minute*, che rappresentano la differenza di fuso orario tra ora locale e ora universale (Coordinated Universal Time o UTC). Esempio: $21:03:04 + 1:00$

Intervalli temporali - 1

Gli intervalli temporali sono stati introdotti in SQL-2.

Sintassi: interval PrimaUnitaDiTempo
[to UltimaUnitaDiTempo]

PrimaUnitaDiTempo e *UltimaUnitaDiTempo* indicano le unità di misura da usare, dalla più precisa alla meno precisa.

Esempio: *interval year to month* (la durata va misurata in numero di anni e di mesi).

L'insieme delle unità di misura è diviso in due sottoinsiemi: *year* e *month* in uno, da *day* a *second* nell'altro (giorni e mesi non possono essere confrontati con facilità, in quanto a un mese possono corrispondere da 28 a 31 giorni).

Intervalli temporali - 2

Alla prima unità che compare nella definizione può essere associata una precisione (numero di cifre in base 10 usate nella rappresentazione).

Quando l'unità più piccola è *second*, si può specificare una precisione che indica il numero di cifre decimali dopo la virgola.

Esempi.

interval year(5) to month consente di rappresentare intervalli di ampiezza al più uguale a 99999 anni e 11 mesi

interval day(4) to second(6) consente di rappresentare intervalli di ampiezza al più uguale a 9999 giorni, 23 ore, 59 minuti e 59,999999 secondi (precisione del milionesimo di secondo).

Domini (elementari) introdotti in SQL-3

Il dominio **BigInt**: si aggiunge a *smallint* e *integer*. Non vi sono limiti di rappresentazone del dominio (unica condizione: non possono essere inferiori a quelli di *integer*)

I domini **BLOB** e **CLOB**: essi consentono di rappresentare oggetti di grandi dimensioni, costituiti da una sequenza arbitraria di valori binari (Binary Large OBject, BLOB) o di caratteri (Character Large OBject, CLOB).

Tali valori possono essere memorizzati (in un'area apposita del sistema), ma non utilizzati in condizioni di selezione di interrogazioni. Utili per gestire informazioni semi-strutturate e multimediali (immagini, documenti, video).

SQL-3 mette a disposizione anche un **insieme di costruttori** (ad esempio, *array*), che recepiscono elementi del paradigma ad oggetti.

Definizione dello schema

SQL definisce uno schema come una **collezione di oggetti** (domini, tabelle, ..).

Sintassi: create schema [NomeSchema]
[[authorization] Autorizzazione]
{ DefinizioneElementoSchema }

dove *Autorizzazione* è il nome del proprietario dello schema (se omesso, si assume che il proprietario sia chi ha eseguito il comando).

Se il nome dello schema è omesso, si assume il nome del proprietario come nome dello schema.

Esempio: CREATE SCHEMA azienda AUTHORIZATION MarioRossi

Definizione delle tabelle - 1

Una tabella SQL è costituita da un insieme ordinato di attributi, coi relativi domini, e da un insieme eventualmente vuoto di vincoli.

Definizione dello schema di una tabella:

```
CREATE TABLE DIPARTIMENTO
(
    DNUMERO      INTEGER       PRIMARY KEY,
    DNOME        VARCHAR2(20),
    MANAGER      CHAR(16),
    DATA_INIZIO  DATE,
    UNIQUE(DNOME),
    FOREIGN KEY (MANAGER) REFERENCES IMPIEGATO(CF)
                                  ON DELETE SET NULL
                                  ON UPDATE CASCADE
)
```

In presenza di più schemi, *CREATE TABLE Azienda.Dipartimento*

Definizione delle tabelle - 2

Sintassi (per la definizione di tabelle):

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

La tabella è inizialmente vuota. Il creatore possiede tutti i privilegi su di essa (diritto di accedere al suo contenuto e di modificarlo).

Definizione dei domini

Nella definizione delle tabelle si può far riferimento ai domini predefiniti del linguaggio o a domini definiti dall'utente a partire dai domini predefiniti.

Sintassi:

CREATE DOMAIN NomeDominio AS TipoDiDato
    [ValoreDiDefault]
    [Vincolo]

Esempio.

CREATE DOMAIN CODICEFISCALE AS CHAR(16)

Specifica di valori di default

Il termine *ValoreDiDefault*, che può essere utilizzato nella definizione di tabelle e domini, specifica il valore che un attributo deve assumere quando viene inserita una tupla priva di un valore per tale attributo (se non specificato, NULL viene assunto quale valore di default).

Sintassi (può essere assegnato loro un nome):

DEFAULT ⟨ ValoreGenerico | user | NULL ⟩

ValoreGenerico è un valore compatibile col dominio (dell'attributo).

L'opzione *user* impone come valore di default l'identificativo dell'utente che esegue il comando di aggiornamento della tabella.

Esempio: *DIP SMALLINT DEFAULT 1*

Specifica di vincoli

Vincoli intrarelazionali e vincoli interrelazionali.

I **vincoli intrarelazionali** sono vincoli che coinvolgono un'unica tabella (vincoli *NOT NULL*, *UNIQUE* e *PRIMARY KEY*).

I **vincoli interrelazionali** sono vincoli che coinvolgono più tabelle (vincoli di integrità referenziale, asserzioni come vincoli di integrità generici).

Ai vincoli può essere assegnato un **nome** (utile in fase di modifica dello schema).

Vincoli intrarelazionali - 1

Vincolo *NOT NULL*: il valore *NULL* non è ammesso quale possibile valore dell'attributo (se all'attributo è associato un default, non occorre necessariamente specificare un valore per l'attributo).

Esempio: COGNOME VARCHAR2(20) NOT NULL

Vincolo *UNIQUE*: si applica a un attributo o ad un insieme di attributi di una tabella e impone che tuple differenti della tabella non possano assumere lo stesso valore su tale/i attributo/i (viene fatta un'eccezione per il valore NULL, che può comparire in più righe senza violare il vincolo).

Esempio: DNOME VARCHAR2(20) UNIQUE

Vincoli intrarelazionali - 2

Vincoli su singoli attributi vs. vincoli su un insieme di attributi

```
NOME VARCHAR2(20) NOT NULL UNIQUE,
COGNOME VARCHAR2(20) NOT NULL UNIQUE
```

vs.

```
NOME VARCHAR2(20) NOT NULL,
COGNOME VARCHAR2(20) NOT NULL,
UNIQUE(NOME,COGNOME)
```

Vincolo *PRIMARY KEY*: può essere specificato una sola volta per ogni tabella.

```
NOME VARCHAR2(20),
COGNOME VARCHAR2(20),
PRIMARY KEY (NOME,COGNOME)
```

Vincoli interrelazionali: chiavi esterne - 1

Vincoli di integrità referenziale, espressi attraverso il vincolo di FOREIGN KEY (**chiave esterna**).

Può essere definito in due modi (la scelta del secondo è obbligata se la chiave esterna è costituita da più attributi):

```
DIP SMALLINT REFERENCES DIPARTIMENTO(DNUMERO)
```

vs.

```
...
MANAGER    CHAR(16),
...
FOREIGN KEY (MANAGER) REFERENCES IMPIEGATO(CF)
                                  ON DELETE SET NULL
                                  ON UPDATE CASCADE
```

Vincoli interrelazionali: chiavi esterne - 2

Il vincolo di chiave esterna crea un legame tra i valori di un attributo (o più) della tabella (**tabella interna**) su cui è definito e i valori di un attributo (o più) della stessa o di un'altra tabella (**tabella esterna**).

L'attributo/i cui si fa riferimento deve essere soggetto (almeno) ad un vincolo *UNIQUE* (di norma, vincolo di CHIAVE PRIMARIA).

Le **variazioni** della **tabella interna** (inserimento di una nuova tupla o modifica del valore dell'attributo referente) che provocherebbero una violazione del vincolo di integrità referenziale vengono rifiutate.

Alle **variazioni** della **tabella esterna** (cancellazione di una tupla o modifica del valore dell'attributo riferito) che provocherebbero una violazione del vincolo di integrità referenziale si può, invece, reagire in vario modo.

Vincoli interrelazionali: chiavi esterne - 3

Possibili reazioni a operazioni di modifica/cancellazione:

- CASCADE: il nuovo valore (risp., la cancellazione) viene riportato (risp., viene propagata) su tutte le corrispondenti tuple della tabella interna (a tutte le corrispondenti tuple della tabella interna);
- SET NULL: all'attributo referente viene assegnato il valore NULL;
- SET DEFAULT: all'attributo referente viene assegnato il valore di default;
- NO ACTION: l'azione di modifica (risp., cancellazione) non viene consentita.

  Sintassi: on ⟨ DELETE | UPDATE ⟩
  ⟨ CASCADE | SET NULL | SET DEFAULT | NO ACTION ⟩

Vincoli interrelazionali: asserzioni - 1

Per specificare ulteriori vincoli, non necessariamente legati a singole tabelle (o coppie di tabelle), SQL-2 ha introdotto la **clausola CHECK**.

Sintassi: CHECK (Condizione)

dove la struttura della condizione è la stessa delle condizioni argomento di una clausola WHERE.

La clausola CHECK può essere usata per specificare i **vincoli predefiniti**. Ad esempio, possiamo specificare che DIP è chiave esterna di IMPIEGATO rispetto a DIPARTIMENTO inserendo la seguente clausola CHECK nella definizione della tabella IMPIEGATO:

CHECK (DIP IN (SELECT DNUMERO FROM DIPARTIMENTO))

Vincoli interrelazionali: asserzioni - 2

Tale opportunità è poco interessante: i vincoli predefiniti sono più compatti e chiari, e possono essere gestiti in modo più efficiente dal DBMS; inoltre, essi consentono di specificare una politica di reazione alle potenziali violazioni.

Vincoli non predefiniti. La clausola CHECK è utile (indispensabile) per specificare vincoli che non appartengono all'insieme dei vincoli predefiniti. Ad esempio, può essere usata per specificare vincoli a livello di tupla (vedi lucidi sul modello relazionale).

Esempio. Ogni impiegato o non ha un supervisore o ha un supervisore del suo dipartimento (clausola CHECK da inserire nella definizione della tabella IMPIEGATO).

```
CHECK (SUPERVISORE IS NULL OR
          DIP IN ( SELECT I.DIP
                   FROM   IMPIEGATO I
                   WHERE  I.CF = IMPIEGATO.SUPERVISORE))
```

Vincoli interrelazionali: asserzioni - 3

Le **asserzioni** sono un'ulteriore componente dello schema di una base di dati che può essere definita usando la clausola CHECK.

Le asserzioni sono vincoli che non vengono associati a specifici attributi o tabelle (essi appartengono direttamente allo schema).

Sintassi: CREATE ASSERTION NomeAsserzione CHECK (Condizione)

Esempio. Vincolo che impone che la tabella IMPIEGATO contenga sempre almeno una tupla.

```
CREATE ASSERTION ALMENOUNIMPIEGATO
    CHECK (1 =< (  SELECT COUNT(*)
                   FROM   IMPIEGATO))
```

Vincoli interrelazionali: asserzioni - 4

Ai vincoli di integrità definiti tramite CHECK o asserzione è associata una politica di controllo:

- **controllo immediato** – immediatamente dopo ogni modifica della base di dati;
- **controllo differito** – al termine dell'esecuzione di un insieme di operazioni raccolte in una transazione.

Osservazione: il controllo differito consente di gestire coppie di vincoli di integrità incrociati (esempio: DIP chiave esterna di IMPIEGATO rispetto a DIPARTIMENTO e MANAGER chiave esterna di DIPARTIMENTO rispetto a IMPIEGATO).

Per garantire la consistenza della base di dati, se un vincolo viene violato da un'operazione di modifica, il sistema annulla (in inglese UNDO, disfa) gli effetti prodotti dall'operazione (se tale operazione appartiene ad una transazione, l'intera transazione viene abortita).

Modifica degli schemi - 1

Per modificare lo schema di una base di dati si possono usare i comandi ALTER e DROP.

Il **comando ALTER** consente di modificare domini e schemi di tabelle.

Sintassi: ALTER DOMAIN NomeDominio
⟨ SET DEFAULT ValoreDefault |
DROP DEFAULT |
ADD CONSTRAINT DefVincolo |
DROP CONSTRAINT NomeVincolo ⟩

Sintassi: ALTER TABLE NomeTabella ⟨
ALTER COLUMN NomeAttributo
⟨ SET DEFAULT ValoreDefault | DROP DEFAULT ⟩ |
ADD CONSTRAINT DefVincolo |
DROP CONSTRAINT NomeVincolo |
ADD COLUMN DefAttributo |
DROP COLUMN NomeAttributo ⟩

Modifica degli schemi - 2

Nota Bene: quando si definisce un nuovo vincolo, esso deve essere soddisfatto dai dati già presenti (altrimenti l'inserimento viene rifiutato).

Esempio. Aggiungere alla tabella DIPARTIMENTO un attributo NROAFF che consenta di registrare il numero di afferenti di ciascun dipartimento.

ALTER TABLE DIPARTIMENTO ADD COLUMN NROAFF NUMERIC(5)

Questione: come garantire che il valore di un tale attributo (ridondante) sia consistente?

Modifica degli schemi - 3

Il **comando DROP** consente di rimuovere dei componenti della base di dati (schemi, domini, tabelle, ..)

Sintassi: DROP ⟨ SCHEMA | DOMAIN | TABLE | VIEW | ASSERTION ⟩
NomeElemento [RESTRICT | CASCADE]

Opzione RESTRICT: il comando non viene eseguito in presenza di componenti non vuoti (schema che contiene tabelle, dominio che compare nella definizione di tabelle, tabelle che contengono tuple o che compaiono nella definizione di altre tabelle o di viste, viste utilizzate nella definizione di altre viste). RESTRICT è l'opzione di default.

Opzione CASCADE: tutti gli eventuali componenti dell'elemento rimosso vanno a loro volta rimossi. Nel caso di rimozione di un dominio in uso, gli attributi definiti su tale dominio vengono associati al dominio elementare usato nella definizione del dominio.

Esempio. Se il dominio STRINGALUNGA è definito come VARCHAR2(80), DROP DOMAIN STRINGALUNGA CASCADE provoca la sostituzione di tutte le eventuali occorrenze di STRINGALUNGA con VARCHAR2(80).

I cataloghi relazionali

Tutti i DBMS relazionali gestiscono il proprio *dizionario dei dati* (descrizione delle tabelle presenti nella base di dati) mediante una struttura relazionale.

Due tipi di tabelle: (i) tabelle che contengono i dati veri e propri e tabelle che contengono i **metadati** (dati che descrivono i dati). Questo secondo insieme di tabelle viene detto **CATALOGO** della base di dati.

Vantaggi: il DBMS può gestire i metadati con le stesse funzioni usate per la gestione dei dati.

Dizionario dei dati e SQL-2

SQL-2 prevede un'articolazione del dizionario dei dati in due livelli.

Livello del **DEFINITION_SCHEMA**: insieme di tabelle che contengono la descrizione di tutte le strutture della base di dati. Può variare da sistema (implementazione) a sistema.

Livello dell'**INFORMATION_SCHEMA**: insieme di viste costruite sul DEFINITION_SCHEMA (quali le viste TABLES, VIEWS, COLUMNS, DOMAINS, ..); costituisce un'interfaccia verso il dizionario dei dati che deve essere garantita dai sistemi che vogliono essere conformi allo standard SQL. Per la descrizione della struttura della base di dati, vengono messe a disposizione 23 viste.

Aggiornamento dei dati in SQL: inserimento - 1

Comandi che permettono di modificare il contenuto della base di dati: inserimento, cancellazione e modifica.

Comando per l'**inserimento** di tuple (due modalità alternative):

INSERT INTO NomeTabella [(ListaAttributi)]
⟨ VALUES (ListaValori) | SelectSQL ⟩

Esempio: inserimento di singole tuple.

```
INSERT INTO DIPARTIMENTO(DNUMERO, DNOME)
        VALUES (5, 'Magazzino')
```

Esempio: inserimento di insiemi di tuple (estratte dalla base di dati).

```
INSERT INTO IMPIEGATI5
        ( SELECT CF, NOME, COGNOME, STIPENDIO
          FROM   IMPIEGATI
          WHERE DIP = 5)
```

Aggiornamento dei dati in SQL: inserimento - 2

La prima modalità viene usata all'interno di programmi ('embedded SQL') per riempire una tabella con dati forniti direttamente dagli utenti (tipicamente, riempimento di una maschera o form).

La seconda modalità permette di inserire dati in una tabella a partire da informazioni già presenti nella base di dati.

Agli attributi per i quali non viene specificato un valore viene assegnato il **valore di default** (in assenza di questo, il valore NULL). Se viola un vincolo di NOT NULL, l'inserimento viene rifiutato.

La corrispondenza tra gli attributi della tabella e i valori da inserire è data dall'ordine in cui compaiono gli attributi nella *ListaAttribuiti* o nella definizione della tabella (se la *ListaAttributi* viene omessa).

Aggiornamento dei dati in SQL: cancellazione - 1

Comando per la **cancellazione** di tuple:

DELETE FROM NomeTabella [WHERE Condizione]

Tale comando rimuove tutte e sole le tuple che soddisfano la condizione; se viene omessa la clausola WHERE, vengono rimosse tutte le tuple.

Se esiste un vincolo di integrità referenziale che fa riferimento, con politica CASCADE, alla tabella in oggetto, l'operazione di cancellazione può propagarsi alle tuple della tabella contenente la chiave esterna (e da qui, eventualmente, ad altre tabelle).

Esempio: cancellazione di singole tuple.

DELETE FROM DIPARTIMENTO WHERE DNOME = 'MAGAZZINO'

Aggiornamento dei dati in SQL: cancellazione - 2

Nella clausola WHERE possono comparire anche interrogazioni nidificate che possono far riferimento ad altre tabelle.

Esempio: cancellazione di insiemi di tuple (eliminazione di tutti i dipartimenti privi di afferenti).

```
DELETE FROM DIPARTIMENTO
    WHERE DNUMERO NOT IN ( SELECT DIP
                           FROM   IMPIEGATI)
```

Comando DELETE vs. comando DROP:

DELETE FROM DIPARTIMENTO

DROP TABLE DIPARTIMENTO CASCADE

DROP TABLE DIPARTIMENTO RESTRICT

Aggiornamento dei dati in SQL: modifica - 1

Comando per la **modifica** di tuple:

UPDATE NomeTabella
    SET Attributo = ⟨ Espressione | SelectSQL | NULL | DEFAULT ⟩
    {, Attributo = ⟨ Espressione | SelectSQL | NULL | DEFAULT ⟩ }
[WHERE Condizione]

UPDATE consente di aggiornare uno o più attributi delle tuple della tabella *NomeTabella* che soddisfano l'eventuale condizione. Se la clausola WHERE è assente, vengono modificate tutte le tuple.

Il nuovo valore può essere:

1. il risultato della valutazione di un'espressione sugli attributi della tabella (che può far riferimento al valore corrente dell'attributo che si vuole modificare);
2. il risultato di un'interrogazione SQL;
3. il valore NULL;
4. il valore di default per il dominio.

Aggiornamento dei dati in SQL: modifica - 2

Esempio: modifica di singole tuple (aumento di 5000 euro dello stipendio dell'impiegato MNTGVN89S14J324Q)..

```
UPDATE IMPIEGATO
        SET STIPENDIO = STIPENDIO +5000
        WHERE CF = MNTGVN89S14J324Q
```

Esempio: modifica di insiemi di tuple (aumento del 10% dello stipendio di tutti gli impiegati del dipartimento 5).

```
UPDATE IMPIEGATO
        SET STIPENDIO = STIPENDIO * 1.1
        WHERE DIP = 5
```

La natura set-oriented (e non tuple-oriented) di SQL va tenuta presente quando vengono eseguiti comandi di aggiornamento.

Aggiornamento dei dati in SQL: modifica - 3

Esempio: si voglia aumentare lo stipendio degli impiegati che guadagnano 25000 euro o meno del 15% e di quelli che guadagnano più di 25000 euro del 10%. Supponiamo di eseguire la seguente coppia ordinata di comandi di aggiornamento.

```
UPDATE IMPIEGATO
        SET STIPENDIO = STIPENDIO * 1.15
        WHERE STIPENDIO =< 25000

UPDATE IMPIEGATO
        SET STIPENDIO = STIPENDIO * 1.1
        WHERE STIPENDIO > 25000
```

Un impiegato con uno stipendio iniziale di 25000 euro riceverà prima l'aumento del 15% e poi, superando i 25000 euro grazie a tale aumento, l'aumento del 10%.

Possibile soluzione: invertire l'ordine degli aggiornamenti (più in generale, usare il costrutto CASE introdotto in SQL-2).