Linguaggio SQL

- il linguaggio SQL e' un linguaggio per la definizione e la manipolazione dei dati, sviluppato originariamente presso il laboratorio IBM a San Jose (Calif.)

- e' un linguaggio oggi usato in numerosi DBMS disponibili come prodotti commerciali

- e' stato definito uno standard noto come SQL2 (noto anche come SQL-92)

- un nuovo standard, noto come SQL3, e' in via di definizione

- tra i DBMS che usano SQL ricordiamo
  - SQL/DS e DB2 (IBM)
  - Oracle
  - SQL Server (Microsoft)
  - Informix
  - Sybase
  - CA-Ingres (Computer Associates)

SQL - Tipi di Dato

- i tipi di dato in SQL sono suddivisi in 3 categorie
  - tipi numerici, tipi carattere e tipi temporali

- i vari tipi sono suddivisi in sottocategorie

- tipi numerici:
  - tipi numerici *esatti*
    - rappresentano valori interi e valori decimali in virgola fissa
  - tipi numerici *approssimati*
    - rappresentano valori numerici approssimati con rappresentazione in virgola mobile

SQL - Tipi di Dato

- tipi numerici esatti:

  INTEGER rappresenta i valori interi
  la precisione (numero totale di cifre) di questo tipo di dato e' espressa in numero di bit o cifre, a seconda della specifica implementazione di SQL

  SMALLINT rappresenta i valori interi
  i valori di questo tipo sono usati per eventuali ottimizzazioni in quanto richiedono minore spazio di memorizzazione
  l'unico requisito e' che la precisione di questo tipo di dato sia non maggiore della precisione del tipo di dato INTEGER

  NUMERIC rappresenta i valori decimali
  e' caratterizzato da una precisione e una scala (numero di cifre della parte frazionaria)
  la specifica di questo tipo di dato ha la forma NUMERIC (p, s)

  DECIMAL e' simile al tipo NUMERIC
  la specifica di questo tipo di dato ha la forma DECIMAL (p, s)
  La differenza tra NUMERIC e DECIMAL e' che il primo deve essere implementato esattamente con la precisione richiesta, mentre il secondo puo' avere una precisione maggiore

SQL - Tipi di Dato

- tipi numerici approssimati:

  REAL rappresenta valori a singola precisione in virgola mobile
  la precisione dipende dalla specifica implementazione di SQL

  DOUBLE PRECISION rappresenta valori a doppia precisione in virgola mobile
  la precisione dipende dalla specifica implementazione di SQL

  FLOAT permette di richiedere la precisione che si desidera
  formato FLOAT (p)

SQL - Tipi di Dato

- tipi carattere:

  CHARACTER questo tipo di dato (spesso abbreviato in CHAR) permette di definire stringhe di caratteri di lunghezza massima predefinita
  la specifica di questo tipo di dato ha la forma CHAR(n) dove n e' la lunghezza massima delle stringhe
  se non viene specificata alcuna lunghezza, il default e' 1

  CHARACTER VARYING
  questo tipo di dato (spesso abbreviato in VARCHAR) permette di definire stringhe di caratteri di lunghezza massima predefinita
  la specifica di questo tipo di dato ha la forma VARCHAR(n) dove n e' la lunghezza massima delle stringhe
  la differenza con il tipo CHAR e' che per questo tipo si alloca per ogni stringa la lunghezza massima predefinita, mentre per VARCHAR si usano strategie diverse

SQL - Tipi di Dato

- tipi temporali:

  DATE rappresenta le date espresse come anno (4 cifre), mese (2 cifre comprese tra 1 e 12), giorno (2 cifre comprese tra 1 e 31 ed ulteriori restrizioni a seconda del mese)

  TIME rappresenta i tempi espressi come ora (2 cifre), minuto (2 cifre) e secondo (2 cifre)

  TIMESTAMP rappresenta una "concatenazione" dei due tipi di dato precedenti con una precisione al microsecondo
  pertanto permette di rappresentare timestamps che consistono di: anno, mese, giorno, ora, minuto, secondo e microsecondo

SQL - Tipi di Dato

Esempio DB2

Tipo di dato	Descrizione
SMALLINT	intero a 16 bit
INTEGER	intero a 32 bit
DECIMAL (p,s)	numero decimale con precisione p e scala s (default p=5, s=0) NUMERIC (p,s) e' sinonimo di DECIMAL (p,s)
DOUBLE	numero in virgola mobile a 64 bit FLOAT e DOUBLE PRECISION sono sinonimi di DOUBLE
CHAR(n)	stringa di lunghezza n, dove n $\leq$ 254 (default n=1)
VARCHAR(n)	stringa di lunghezza variabile con lunghezza massima pari ad n, dove n $\leq$ 4000
DATE	consiste di anno, mese e giorno
TIME	consiste di ora, minuto e secondo
TIMESTAMP	consiste di anno, mese, giorno, ora, minuto, secondo, microsecondo

SQL - Creazione di tabelle

- La creazione avviene tramite il comando di
  Create Table

- Sintassi:

```
CREATE TABLE table-name
   (spec_col1 [, ...., spec_coln])
   [, PRIMARY KEY (column-name-list)]
   [, FOREIGN KEY (column-name-list)
        REFERENCES other-table-name
     [ ON DELETE { NO ACTION | RESTRICT |
        CASCADE | SET NULL | SET DEFAULT}]
     [ ON UPDATE { NO ACTION | RESTRICT |
        CASCADE | SET NULL | SET DEFAULT}]
   [, FOREIGN KEY.....]])
```

dove:

- table-name e' il nome della relazione che viene creata
- $spec\_col_i$ (i=1,....,n) e' una specifica di colonna il suo formato e' il seguente

```
NomeColonnai Dominioi
     [DEFAULT valoredefault]
     [NOT NULL]
```

SQL - Creazione di tabelle

Opzioni riguardanti le azioni da eseguire nel caso di cancellazione di una tupla riferita tramite chiave esterna:

(1) NO ACTION: non viene effettuata alcuna azione
la cancellazione e' quindi permessa

(2) RESTRICT: la cancellazione di una tupla dalla tabella riferita e' eseguita solo se non esiste alcuna tupla nella tabella referente che ha come chiave esterna la chiave della tupla da cancellare

(3) CASCADE: la cancellazione di una tupla dalla tabella riferita implica la cancellazione di tutte le tuple della tabella referente che hanno come chiave esterna la chiave della tupla da cancellare

(4) SET NULL: la cancellazione di una tupla dalla tabella riferita implica che in tutte le tuple della tabella referente che hanno come chiave esterna la chiave della tupla da cancellare, la chiave esterna viene posta a valore NULL (se ammesso)

(5) SET DEFAULT: la cancellazione di una tupla dalla tabella riferita implica che in tutte le tuple della tabella referente che hanno come chiave esterna la chiave della tupla da cancellare, il valore della chiave viene posto uguale al valore di default specificato per le colonne che costituiscono la chiave esterna

SQL - Creazione di tabelle

Opzioni riguardanti le azioni da eseguire nel caso di modifica della chiave della tupla riferita tramite chiave esterna:

hanno lo stesso significato delle opzioni viste per la cancellazione

- <u>differenza</u>
  opzione CASCADE
  ha l'effetto di assegnare alla chiave esterna il nuovo valore della chiave della tupla riferita

- default: NO ACTION sia per la cancellazione che l'update

SQL - Creazione di tabelle

Esempio

```
CREATE TABLE Impiegati
    (Imp#        Decimal(4) NOT NULL,
     Nome        Char(20),
     Mansione    Char(10),
     Data_A      Date,
     Stipendio   Decimal(7,2),
     Premio_P    Decimal(7,2),
     Dip#        Decimal(2));
```


SQL - Creazione di tabelle

Esempio

```
CREATE TABLE Docente
    (Dno Char(7) NOT NULL,
     Dnome Varchar(20) NOT NULL,
     Residenza Varchar(15) NOT NULL,
     PRIMARY KEY (Dno));

CREATE TABLE Studente
    (Sno Char(6) NOT NULL,
     Sname Varchar(20) NOT NULL,
     Residenza Varchar(15),
     Birthdate Date NOT NULL
     PRIMARY KEY (Sno));


CREATE TABLE Relatore
    (Dno Char(7) NOT NULL,
     Sno Char(6) NOT NULL,
     PRIMARY KEY (Sno),
     FOREIGN KEY (Dno)
         REFERENCES Docente
         ON DELETE RESTRICT
         ON UPDATE CASCADE,
    FOREIGN KEY (Sno)
         REFERENCES Studente
         ON DELETE CASCADE,
         ON UPDATE CASCADE);
```

SQL - Constraints

- i constraints impongono vari tipi di condizioni sui valori delle tuple in una tabella

- la specifica NOT NULL associata ad una colonna e' un esempio di constraint

- altri esempi includono la specifica di range di valori per gli attributi di una colonna

- i vincoli sono specificati tramite l'uso di una o piu' clausole di constraints nel comando per la definizione delle tabelle

- formato comando (DB2 V2):

CONSTRAINT constraint-name
    CHECK <check-condition>

- la condizione di controllo (check-condition) e' un qualsiasi predicato (o combinazione booleana di predicati) con l'unico vincolo che la condizione deve poter essere valutata esaminando ogni singola tupla

- Esempio di constraints corretti

CONSTRAINT check1 **CHECK**
(Mansione **IN** ('dirigente', 'ingegnere','tecnico', 'segretaria'))

CONSTRAINT check2
CHECK (Stipendio < 10000 AND Premio_P $\leq$ Stipendio)

- Esempio di constraint non corretto

CONSTRAINT check3
    CHECK (salary < (Select MAX(salary)
                FROM Emp))

SQL - Constraints

- Esempio:

```
CREATE TABLE Esame
   (Sno      Char(6) NOT NULL,
   Cno       Char(6) NOT NULL,
   Cnome     Varchar (20) NOT NULL,
   Voto      Integer NOT NULL,
   PRIMARY KEY (Sno,Cno),
   FOREIGN KEY (Sno) REFERENCES Studente
        ON DELETE CASCADE,
   CONSTRAINT Cname-constr
       CHECK Cname IN ('Documentazione Automatica',
                       'SEI I', 'SEI II'),
   CONSTRAINT Voto-constr
        CHECK Voto ≥ 18 AND Voto ≤ 30);
```

- valutazione constraints:

(a) un constraint da una tupla e' violato se la condizione valutata sulla tupla da valore False

(b) un constraint la cui valutazione su una data tupla da' valore True o Unknown (a causa di valori nulli) non e' violato

(c) durante l'inserzione o la modifica di un insieme di tuple, la violazione di un vincolo per una delle tuple causa la non esecuzione dell'intero insieme;
il programma tuttavia continua la sua esecuzione

SQL - Cancellazione e ridenominazione di relazioni e modifiche a schemi di relazioni

- DROP TABLE R
  dove R e' il nome della relazione da cancellare

  esempio:
  DROP TABLE Impiegati;

- RENAME R_v TO R_n
  dove R_v e R_n sono, rispettivamente, il vecchio e nuovo nome della relazione

  esempio:
  RENAME Impiegati TO Imp;

- ALTER TABLE R ADD COLUMN spec_col
  aggiunge una nuova colonna ad una relazione

  esempio:
  ALTER TABLE Impiegati
  ADD COLUMN (Prog# Decimal(3));

- ALTER TABLE R ALTER COLUMN spec_col
  modifica una colonna di una relazione

  esempio:
  ALTER TABLE Impiegati
  ALTER COLUMN (Prog# Decimal(4));

- ALTER TABLE R DROP COLUMN nome_col
  rimuove una colonna da una relazione

  esempio:
  ALTER TABLE Impiegati
  DROP COLUMN Premio_P;

SQL - Interrogazioni formato di base comando SELECT

- la forma piu' comune di interrogazione in SQL ha la seguente struttura:

  SELECT $R_{i1}.C_1$, $R_{i2}.C_2$, ..., $R_{in}.C_n$

  FROM R_1, R_2, ..., R_k

  WHERE F;

  dove

  - R_1, R_2, ..., R_k e' una lista di nomi distinti di relazioni;
  - $R_{i1}.C_1$, $R_{i2}.C_2$, ..., $R_{in}.C_n$ e' una lista di nomi di colonne

- la notazione R.C indica la colonna di nome C nella relazione R
- se una sola relazione nella lista di relazioni nella clausola FROM ha una colonna di nome C, si puo' usare C invece di R.C
- F e' un predicato analogo ai predicati visti nel caso dell'operazione relazionale σ
- il significato di una query SQL puo' essere espresso per mezzo della seguente espressione algebrica

$$\pi_{Ri1.C1,\ Ri2.C2,\ .....,\ Rin.Cn}(\sigma_F(R_1 \times R_2 \times ... \times R_k))$$

SQL - Interrogazioni
esempi dalla base di dati impiegati e dipartimenti

Q1: selezionare gli impiegati che hanno uno stipendio maggiore di 2000

$\sigma_{Stipendio>2000}(Impiegati)$

```
SELECT * FROM Impiegati
    WHERE Stipendio>2000;
```

Nota: il simbolo * nella clausola di proiezione indica che tutte le colonne delle relazione devono essere ritrovate;

risultato Q1

Imp#	Nome	Mansione	Data_A	Stipendio	Premio_P	Dip#
7566	Rosi	dirigente	02-Apr-81	2975,00	?	20
7698	Blacchi	dirigente	01-Mag-81	2850,00	?	30
7782	Neri	ingegnere	01-Giu-81	2450,00	200,00	10
7839	Dare	ingegnere	17-Nov-81	2600,00	300,00	10
7977	Verdi	dirigente	10-Dic-80	3000,00	?	10

SQL - Interrogazioni
esempi dalla base di dati impiegati e dipartimenti

Q2: selezionare il nome e il numero di dipartimento degli impiegati che hanno uno stipendio maggiore di 2000 e hanno mansione di ingegnere

$\pi_{Nome,Dip\#}(\sigma_{Stipendio>2000 \wedge Mansione='Ingegnere'}(Impiegati))$

```
SELECT Nome, Dip# FROM Impiegati
    WHERE Stipendio>2000 AND
          Mansione = 'ingegnere';
```

risultato Q2

Nome	Dip#
Neri	10
Dare	10

Q3: selezionare il numero degli impiegati che lavorano nel dipartimento 30 e sono ingegneri o tecnici

$\pi_{Imp\#}(\sigma_{Dip\#=30 \wedge (Mansione= 'ingegnere' \vee Mansione= 'tecnico')}(Impiegati))$

```
SELECT Imp# FROM Impiegati
    WHERE Dip#=30 AND
    (Mansione = 'ingegnere' OR Mansione = 'tecnico');
```

risultato Q3

Imp#
7499
7521
7844
7900

SQL - Interrogazioni

condizioni su intervalli di valori

- l'operatore BETWEEN permette di determinare le tuple che contengono in un dato attributo valori in un intervallo dato;
- formato

    C BETWEEN v_1 AND v_2

    forma negata

    C NOT BETWEEN v_1 AND v_2

- esempio

```
SELECT Nome, Stipendio FROM Impiegati
WHERE Stipendio BETWEEN 1100 AND 1400;
```

risultato

Nome	Stipendio
Adami	1100,00
Milli	1300,00

SQL - Interrogazioni
ricerca di valori in un insieme

- l'operatore IN permette di determinare le tuple che contengono, in un dato attributo, uno tra i valori in un insieme specificato;
- formato

    C IN (v_1, v_2,....,v_n) C IN *sq*
    (dove *sq* e' una sottoquery)

    forma negata

    C NOT IN (v_1, v_2,....,v_n) C NOT IN *sq*

- esempio

```
SELECT * FROM Dipartimenti
WHERE Dip# IN (10,30);
```

risultato

Dip#	Nome_Dip	Ufficio	Divisione	Dirigente
10	Edilizia Civile	1100	D1	7977
30	Edilizia Stradale	5100	D2	7698

SQL - Interrogazioni
confronto tra stringhe di caratteri

- l'operatore LIKE permette di eseguire alcune semplici operazioni di *pattern-matching* su colonne di tipo stringa

- un predicato di confronto espresso con l'operatore LIKE ha il seguente formato

  C LIKE *pattern*

  dove *pattern* e' una stringa di caratteri che puo' contenere i caratteri speciali % e _

  il carattere % denota una sequenza di caratteri arbitrari di lunghezza qualsiasi (anche zero)

  il carattere _ denota esattamente un carattere

- esempio: determinare tutti gli impiegati che hanno 'R' come terza lettera del cognome

```
SELECT Nome FROM Impiegati
WHERE Nome LIKE '_ _ R%';
```

risultato

Nome
Martini
Neri
Dare
Turni
Fordi
Verdi

- i predicati formati con gli operatori BETWEEN, IN e LIKE possono essere connessi con AND ed OR nella specifica di condizioni complesse

SQL - Interrogazioni
ordinamento del risultato di una query

- negli esempi visti, l'ordine delle tuple risultato di una interrogazione e' determinato dal sistema
  (dipende dalla strategia usata per eseguire l'interrogazione)

- e' possibile specificare un ordinamento diverso aggiungendo alla fine dell'interrogazione la clausola ORDER BY

- esempio: elencare lo stipendio, la mansione e il nome di tutti gli impiegati del dipartimento 30, ordinando le tuple in ordine crescente in base allo stipendio

```
SELECT Stipendio, Mansione, Nome
    FROM Impiegati
    WHERE Dip#=30
    ORDER BY Stipendio;
```

risultato - le tuple sono ordinate in ordine crescente

Stipendio	Mansione	Nome
800,00	tecnico	Andrei
800,00	tecnico	Bianchi
800,00	segretaria	Martini
1500,00	tecnico	Turni
1950,00	ingegnere	Gianni
2850,00	dirigente	Blacchi

SQL - Interrogazioni
ordinamento del risultato di una query

- l'ordinamento non e' limitato ad una sola colonna, ne' ad un ordine crescente

- esempio: si vuole elencare mansione, nome e stipendio di tutti gli impiegati ordinando le tuple in base alla mansione in ordine crescente, ed in base allo stipendio in ordine decrescente

```
SELECT Mansione, Stipendio, Nome
    FROM Impiegati
    ORDER BY Mansione, Stipendio DESC;
```

risultato

Mansione	Stipendio	Nome
dirigente	3000,00	Verdi
dirigente	2975,00	Rosi
dirigente	2850,00	Blacchi
ingegnere	2450,00	Neri
ingegnere	2000,00	Dare
ingegnere	1950,00	Gianni
ingegnere	1600,00	Rossi
ingegnere	1300,00	Milli
ingegnere	1100,00	Adami
segretaria	1000,00	Fordi
segretaria	800,00	Martini
segretaria	800,00	Scotti
tecnico	1500,00	Turni
tecnico	800,00	Andrei
tecnico	800,00	Bianchi

SQL - Interrogazioni
eliminazione dei duplicati

- supponiamo di voler ritrovare la lista di tutte le mansioni presenti nella relazioni Impiegati

```
SELECT Mansione FROM Impiegati;
```

risultato

Mansione
ingegnere
tecnico
tecnico
dirigente
segretaria
dirigente
ingegnere
segretaria
ingegnere
tecnico
ingegnere
ingegnere
segretaria
ingegnere
dirigente

- e' possibile richiedere l'eliminazione dei duplicati tramite la clausola DISTINCT

```
SELECT DISTINCT Mansione FROM Impiegati;
```

risultato

Mansione
ingegnere
tecnico
dirigente
segretaria

SQL - Operazione di join

- l'operazione di join rappresenta un'importante operazione in quanto permette di correlare dati rappresentati da relazioni diverse

- SQL non prevede un'operazione di join esplicita; il join e' espresso in SQL tramite un prodotto Cartesiano a cui sono applicati uno o piu' *predicati di join*

- un predicato di join esprime una relazione che deve essere verificata dalle tuple risultato dell'interrogazione

- esempio: determinare il nome del dipartimento in cui lavora l'impiegato Rossi

```
SELECT  Nome_Dip
    FROM Impiegati, Dipartimenti
    WHERE Nome = 'Rossi' AND
    Impiegati.Dip# = Dipartimenti.Dip#;
```

il predicato di join e'

```
    Impiegati.Dip# = Dipartimenti.Dip#
```

- e' possibile eseguire il join tra una tupla di una relazione e piu' tuple di un'altra relazione

SQL - Operazione di join

- il risultato di un'operazione di join e' una relazione; e' pertanto possibile richiedere che tale relazione sia ordinata anche in base a valori di colonne di relazioni diverse o che le tuple duplicate siano eliminate

- si vuole ritrovare per ogni impiegato il nome, lo stipendio, la mansione, e il nome del dipartimento in cui l'impiegato lavora. Inoltre si vuole ordinare le tuple in ordine crescente in base al nome del dipartimento, ed in ordine decrescente in base allo stipendio.

```
SELECT Nome_Dip, Nome, Mansione, Stipendio
    FROM Impiegati, Dipartimenti
    WHERE Impiegati.Dip# = Dipartimenti.Dip#
    ORDER BY Nome_Dip, Stipendio DESC;
```

risultato

Nome_Dip	**Nome**	**Mansione**	**Stipendio**
Edilizia Civile	Verdi	dirigente	3000,00
Edilizia Civile	Neri	ingegnere	2450,00
Edilizia Civile	Dare	ingegnere	2000,00
Edilizia Civile	Milli	ingegnere	1300,00
Edilizia Stradale	Blacchi	dirigente	2850,00
Edilizia Stradale	Gianni	ingegnere	1950,00
Edilizia Stradale	Turni	tecnico	1500,00
Edilizia Stradale	Andrei	tecnico	800,00
Edilizia Stradale	Bianchi	tecnico	800,00
Edilizia Stradale	Martini	segretaria	800,00
Ricerche	Rosi	dirigente	2975,00
Ricerche	Rossi	ingegnere	1600,00
Ricerche	Adami	ingegnere	1100,00
Ricerche	Fordi	segretaria	1000,00
Ricerche	Scotti	segretaria	800,00

SQL - Espressioni e funzioni aritmetiche

- i predicati usati nelle interrogazioni possono coinvolgere, oltre a nomi di colonna, anche espressioni aritmetiche

- tali espressioni sono formulate applicando gli operatori aritmetici (+, -, *, /) ai valori delle colonne delle tuple

- le espressioni artimetiche possono comparire nella clausola di proiezione e nelle espressioni di assegnamenti del comando di UPDATE

- esempio: trovare il nome, lo stipendio, il premio di produzione, e la somma dello stipendio e del premio di produzione di tutti gli ingegneri

```
SELECT Nome, Stipendio, Premio_P, Stipendio+Premio_P
        FROM Impiegati
        WHERE Mansione = 'ingegnere';
```

risultato

Nome	Stipendio	Premio_P	Stipendio + Premio_P
Rossi	1600,00	500,00	2100,00
Neri	2450,00	200,00	2650,00
Dare	2000,00	300,00	2300,00
Adami	1100,00	500,00	1600,00
Gianni	1950,00	?	1950,00
Milli	1300,00	150,00	1450,00

SQL - Espressioni e funzioni aritmetiche

- esempio: trovare il nome, lo stipendio, il premio di produzione, e la somma dello stipendio e del premio di produzione di tutti gli ingegneri per cui la somma dello stipendio e del premio di produzione e' maggiore di 2000

```
SELECT Nome, Stipendio, Premio_P, Stipendio+Premio_P
        FROM Impiegati
        WHERE Mansione = 'ingegnere' AND
              Stipendio+Premio_P > 2000;
```

risultato

Nome	Stipendio	Premio_P	Stipendio + Premio_P
Rossi	1600,00	500,00	2100,00
Neri	2450,00	200,00	2650,00
Dare	2000,00	300,00	2300,00

- una espressione aritmetica usata nella clausola di proiezione di un'interrogazione da' luogo ad una colonna, detta *virtuale*, non presente nella relazione su cui si effettua l'interrogazione

- le colonne virtuali non sono fisicamente memorizzate; sono solo *materializzate* come risultato delle interrogazioni

- nel calcolo delle espressioni aritmetiche il valore nullo viene considerato uguale al valore zero

SQL - Funzioni scalari

- sono fornite non in tutte le implementazioni di SQL
  (non sono oggetto di standardizzazione)

- funzioni:

  - abs(n)
    calcola il valore assoluto del valore numerico n

  - mod(n,b)
    calcola il resto intero della divisione di n per b

  - sqrt(n)
    calcola la radice quadrata

- alcuni DBMS supportano anche funzioni trigonometriche e funzioni per il calcolo della parte intera superiore ed inferiore

- le funzioni possono essere usate nella clausola di proiezione e nella clausola WHERE

  WHERE mod(A,5) > 3
  dove A e' un nome di colonna

SQL - Espressioni e Funzioni per Stringhe

- espressioni di tipo stringa possono essere definite tramite l'operatore di concatenazione denotato da ||

  ```
  SELECT Cognome || ' ' || Nome || ' ' || Indirizzo
  FROM Persone;
  ```

  restituisce un'unica stringa che contiene congnome, nome ed indirizzo separati da uno spazio bianco

- funzioni:
  - length(str)
    calcola la lunghezza di un stringa

    WHERE length(Cognome) > 3

  - substr(str, m, [,n])
    (m ed n sono interi)
    estrae dalla stringa 'str' la sottostringa dal carattere di posizione m per una lunghezza n (se n e' specificato) oppure fino all'ultimo carattere

SQL - Date e tempi

- <u>funzioni per conversione tra date/tempi ed altri tipi di dato</u> tra le funzioni di conversione le piu' rilevanti includono
  - le funzioni DATE, TIME, e TIMESTAMP convertono rispettivamente un valore scalare in una data, un tempo, un timestamp

    esempio: DATE('6/20/1997')
  - la funzione CHAR che converte un valore di data/tempo in una stringa di caratteri; tale funzione puo' inoltre ricevere una specifica di formato

    esempio: CHAR(data_assunzione, EUR)
  - la funzione DAYS che converte una data in un intero che rappresenta il numero di giorni a partire dall'anno 0

    esempio: DAYS('1/18/19)
- <u>funzioni per la determinazione del valore di registri speciali</u>
  - CURRENT_DATE
  - CURRENT_TIME
  - CURRENT_TIMESTAMP
- date e tempi possono essere usati in espressioni aritmetiche; in tali espressioni e' possibile usare diverse *unita' temporali* quali:
  YEAR[S], MONTH[S], DAY[S], HOUR[S], MINUTE[S], SECOND[S]

SQL - Date e tempi

- esempio: si vuole avere un colloquio con tutti i nuovi impiegati dopo 90 giorni dalla loro assunzione. In particolare, per tutti gli impiegati del dipartimento 10 per cui si deve ancora effettuare il colloquio, si vuole determinare il nome, la data di assunzione, la data del colloquio ed, inoltre, la data corrente di esecuzione dell'interrogazione;

```
SELECT Nome, CHAR(Data_A, EUR),
       CHAR(CURRENT_DATE, EUR),
       CHAR(Data_A + 90 DAYS, EUR)
       FROM Impiegati WHERE
       Data_A + 90 DAYS > CURRENT_DATE AND
       Dip#=10;
```

risultato

Nome	Data_A	CURRENT_DATE	Data_A + 90
Rossi	23/01/82	04/03/82	25/03/82

SQL - Funzioni di gruppo

- una funzione di gruppo permette di estrarre informazioni da gruppi di tuple di una relazione

- le funzioni di gruppo si basano su due concetti fondamentali

  - partizionamento delle tuple di una relazione in base al valore di una o piu' colonne della relazione
    le colonne da usare sono specificate tramite la clausola GROUP BY

  - calcolo della funzione di gruppo per ogni gruppo ottenuto dal partizionamento
    una funzione di gruppo ha come argomento una colonna e si applica all'insieme di valori di questa colonna, estratti dalle tuple che appartengono allo stesso gruppo

- le funzioni di gruppo comunemente presenti sono:

  MAX, MIN, SUM, AVG, COUNT
  (alcuni sistemi includono anche STDEV e VARIANCE)

- tutte le funzioni di gruppo, ad accezione di COUNT, possono essere applicate solo su insiemi che consistono di valori semplici e non su insiemi di tuple

- la funzione COUNT puo' avere due tipi di argomenti

  - un nome di colonna - in tal caso nello standard SQL2 e' obbligatorio l'uso del qualificatore DISTINCT

    esempio: COUNT (DISTINCT Stipendio)

  - il carattere speciale '*' - in tal caso la funzione restituisce il numero di tuple presenti in un dato gruppo

    esempio: COUNT(*)

SQL - Funzioni di gruppo

- esempio: si vuole raggruppare gli impiegati in base al numero di dipartimento e si vuole determinare il massimo stipendio di ogni gruppo

```
SELECT Dip#, MAX(Stipendio)
  FROM Impiegati
  GROUP BY Dip#;
```

risultato

Dip#	MAX(Stipendio)
10	3000,00
20	2975,00
30	2850,00

- in una query contenente una clausola GROUP BY, ogni tupla della relazione risultato rappresenta un gruppo di tuple della relazione su cui la query e' eseguita

- nell'esempio visto i gruppi sono tre: uno per ogni valore di DIP#
  ad ognuno di questi gruppi e' applicata la funzione MAX sulla colonna Stipendio

SQL - Funzioni di gruppo

- piu' colonne possono essere usate per definire gruppi
- le funzioni di gruppo possono essere usate anche in presenza di join
- esempio: supponiamo di voler raggruppare gli impiegati sulla base del dipartimento e della mansione;
  per ogni gruppo si vuole determinare il nome del dipartimento, la somma degli stipendi, quanti impiegati appartengono ad ogni gruppo, e la media degli stipendi

```
SELECT Nome_Dip, Mansione, SUM(Stipendio),
       COUNT(*), AVG(Stipendio)
       FROM Dipartimenti, Impiegati
       WHERE Dipartimenti.Dip#=Impiegati.Dip#
       GROUP BYNome_Dip, Mansione;
```

risultato

Nome_Dip	Mansione	SUM(Stipendio)	COUNT(*)	AVG(Stipendio)
Edilizia Civile	dirigente	3000,00	1	3000,00
Edilizia Civile	ingegnere	5750,00	3	1916,66
Edilizia Stradale	dirigente	2850,00	1	100,00
Edilizia Stradale	ingegnere	1950,00	1	1950,00
Edilizia Stradale	segretaria	800,00	1	800,00
Edilizia Stradale	tecnico	3100,00	3	1033,33
Ricerche	dirigente	2975,00	1	2975,00
Ricerche	ingegnere	2700,00	2	1350,00
Ricerche	segretaria	1800,00	2	900,00

- le colonne della relazione risultato ottenute dall'applicazione delle funzioni di gruppo sono colonne virtuali

SQL - Funzioni di gruppo

- importante restrizione: una clausola di proiezione di una query contenente la clausola GROUP BY puo' solo includere:
  - una o piu' colonne tra le colonne che compaiono nella clausola GROUP BY
  - funzioni di gruppo
- le funzioni di gruppo possono apparire in espressioni aritmetiche

  esempio: SUM(Stipendio) + SUM(Premio_p)

SQL - Funzioni di gruppo

clausola HAVING

- e' possibile specificare condizioni di ricerca su gruppi di tuple

- esempio: supponiamo di voler eseguire una query come la precedente ma di essere interessati solo ai gruppi che contengono almento due impiegati

```
SELECT Nome_Dip, Mansione, SUM(Stipendio),
       COUNT(*), AVG(Stipendio)
       FROM Dipartimenti, Impiegati
       WHERE Dipartimenti.Dip#=Impiegati.Dip#
       GROUP BYNome_Dip, Mansione
       HAVING COUNT(*) ≥ 2;
```

risultato

Nome_Dip	Mansione	SUM(Stipendio)	COUNT(*)	AVG(Stipendio)
Edilizia Civile	ingegnere	5750,00	3	1916,66
Edilizia Stradale	tecnico	3100,00	3	1033,33
Ricerche	ingegnere	2700,00	2	1350,00
Ricerche	segretaria	1800,00	2	900,00

- la clausola HAVING puo' essere una combinazione Booleana di predicati; tali predicati tuttavia possono essere solo predicati che coinvolgono funzioni di gruppo

SQL - Funzioni di gruppo

Un modello di "esecuzione"

1. si applica la condizione di ricerca specificata nella clausola WHERE a tutte le tuple della relazione oggetto della query
   la valutazione avviene tupla per tupla

2. alle tuple ottenute al passo precedente, si applica il partizionamento specificato dalla clausola GROUP BY

3. ad ogni gruppo di tuple ottenuto al passo precedente, si applica la condizione di ricerca specificata dalla clausola HAVING

4. i gruppi ottenuti al passo precedente sono i gruppi di tuple che verificano la query
   per tali gruppi, vengono calcolate le funzioni di gruppo specificate nella clausola di proiezione della query
   i valori restituiti da tali funzioni costituiscono il risultato della query

SQL - null values

- SQL usa una logica a tre valori per valutare il valore di verita' di una condizione di ricerca (clausola where)

  True (T), False (F), Unknown (?)

- un predicato semplice valutato su un attributo a valore nullo da' come risultato della valutazione ?

- il valore di verita' di un predicato complesso viene calcolato in base alle seguenti tabelle di verita'

AND

	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR

	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

NOT

T	F
F	T
?	?

- una tupla per cui il valore di verita' e' ? non viene restituita dalla query

SQL - Null values

- esempio: R

A	B	C	
a	?	c1	t1
a1	b	c2	t2
a2	?	?	t3

SELECT * FROM R WHERE A=a OR B=b;
il valore di verita' della condizione per ogni tupla e' il seguente:
t1 T OR ? --> T
t2 F OR T --> T
t3 F OR ? --> ?
le tuple che verificano la query sono t1 e t2

SELECT * FROM R WHERE A=a AND B=b;
il valore di verita' della condizione per ogni tupla e' il seguente:
t1 T AND ? --> ?
t2 F AND T --> F
t3 F AND ? --> F
nessuna tupla verifica la query

SELECT * FROM R WHERE NOT C=c1;
il valore di verita' della condizione per ogni tupla e' il seguente:
t1 NOT T --> F
t2 NOT F --> T
t3 NOT ? --> ?
la tupla che verifica la query e' t2

SQL - Null values

- predicati IS NULL e IS NOT NULL
- il predicato IS NULL applicato ad un dato attributo di una tupla restituisce True se la tupla ha valore nullo per l'attributo

  esempio:
  SELECT * FROM R WHERE B IS NULL;
  restituisce le tuple t1 e t3

  SELECT * FROM R WHERE B IS NULL
  AND C IS NULL;
  restituisce la tupla t3

  SELECT * FROM R WHERE B IS NULL
  OR C IS NULL;
  restituisce le tuple t1 e t3

- il predicato IS NOT NULL applicato ad un dato attributo di una tupla restituisce True se la tupla ha valore non nullo per l'attributo

  esempio:
  SELECT * FROM R WHERE B IS NOT NULL;
  restituisce la tuple t2

SQL - Subqueries

- una delle ragioni che rendono SQL un linguaggio potente e' la possibilita' di esprimere queries piu' complesse in termini di queries piu' semplici, tramite il meccanismo delle subqueries (sottointerrogazioni)

- la clausola WHERE di una query (detta query esterna) puo' infatti contenere un'altra query (detta subquery)

- la subquery viene usata per determinare uno o piu' valori da usare come valori di confronto in un predicato della query esterna

- esempio: si vuole elencare tutti gli impiegati che hanno la stessa mansione dell'impiegato di nome Gianni

```
SELECT Nome, Mansione FROM Impiegati
WHERE Mansione = (SELECT Mansione
          FROM Impiegati WHERE
          Nome = 'Gianni');
```

  - la subquery restituisce come valore 'ingegnere'
  - la query esterna determina quindi tutti gli impiegati che sono ingegneri
  - il risultato e'

  {Rossi, Neri, Dare, Adami, Gianni, Milli}

SQL - Subqueries

- tramite il meccanismo delle subqueries e' possibile esprimere queries anche piu' complesse
- esempio: si vuole elencare tutti gli impiegati che hanno uno stipendio superiore alla media degli stipendi di tutti gli impiegati

```
SELECT Nome, Stipendio FROM Impiegati
WHERE Stipendio >
        (SELECT AVG(Stipendio)
              FROM Impiegati);
```

risultato

Nome	Stipendio
Rosi	2975,00
Blacchi	2850,00
Neri	2450,00
Dare	2000,00
Gianni	1950,00
Verdi	3000,00

- e' possibile per una subquery avere al suo interno un'altra subquery, predicati di join, e tutti i predicati visti
- le subqueries possono essere usate anche all'interno dei comandi di manipolazione dei dati (Insert, Delete, Update)

SQL - Subqueries

- negli esempi visti, le subqueries restituiscono un solo valore
- se la subquery restituisce piu' valori e' necessario specificare come i valori restituiti devono essere usati nella clausola WHERE
- a tale scopo vengono usati i *quantificatori* ANY ed ALL, che sono inseriti tra l'operatore di confronto e la subquery
- esempio: determinare lo stipendio, la mansione, il nome e il numero di dipartimento degli impiegati che guadagnano piu' di *almeno un* impiegato del dipartimento 30

```
SELECT Stipendio, Mansione, Nome, Dip#
   FROM Impiegati WHERE
           Stipendio > ANY (SELECT Stipendio
                               FROM Impiegati
                               WHERE Dip#=30);
```

risultato

Stipendio	Mansione	Nome	Dip#
3000,00	dirigente	Verdi	10
2975,00	dirigente	Rosi	20
2850,00	dirigente	Blacchi	30
2450,00	ingegnere	Neri	10
2000,00	ingegnere	Dare	10
1950,00	ingegnere	Gianni	30
1600,00	ingegnere	Rossi	20
1500,00	tecnico	Turni	30
1300,00	ingegnere	Milli	10
1100,00	ingegnere	Adami	10

SQL - Subqueries

- se si usa il quantificatore ALL, la query restituisce gli impiegati il cui stipendio e' maggiore di *tutti* i valori restituiti dalla subquery

- esempio: determinare lo stipendio, la mansione, il nome e il numero di dipartimento degli impiegati che guadagnano piu' di *tutti gli* impiegati del dipartimento 30

```
SELECT Stipendio, Mansione, Nome, Dip#
   FROM Impiegati WHERE
          Stipendio > ALL (SELECT Stipendio
                              FROM Impiegati
                              WHERE Dip#=30);
```

risultato

Stipendio	Mansione	Nome	Dip#
3000,00	dirigente	Verdi	10
2975,00	dirigente	Rosi	20

SQL - Subqueries

- sono definite le seguenti abbreviazioni per ANY ed ALL
  - IN equivalente ad =ANY
  - NOT IN equivalente ad ≠ALL

- esempio: elencare il nome e la mansione degli impiegati del dipartimento 10 che hanno la stessa mansione di un qualche impiegato del dipartimento 30

```
SELECT Nome, Mansione FROM Impiegati
   WHERE Dip#=10 AND Mansione IN
   (SELECT Mansione FROM Impiegati
        WHERE Dip#=30);
```

risultato

Mansione	Nome
dirigente	Verdi
ingegnere	Dare
ingegnere	Milli

- esempio: elencare il nome e la mansione degli impiegati del dipartimento 10 che hanno una qualche mansione non presente nel dipartimento 30

```
SELECT Nome, Mansione FROM Impiegati
   WHERE Dip#=10 AND Mansione NOT IN
          (SELECT Mansione FROM Impiegati
               WHERE Dip#=30);
```

risultato

Mansione	Nome
tecnico	Andrei
tecnico	Bianchi
tecnico	Turni
segretaria	Martini

SQL - Subqueries

- e' inoltre possibile selezionare piu' di una colonna tramite una sottointerrogazione; in tal caso e' necessario apporre delle parentesi alla lista delle colonne a sinistra dell'operatore di confronto
- esempio: si vuole elencare gli impiegati con la stessa mansione e stipendio di Martini

```
SELECT Nome FROM Impiegati
   WHERE (Mansione,Stipendio) =
             (SELECT Mansione, Stipendio
                  FROM Impiegati
                  WHERE Nome = 'Martini');
```


SQL - Subqueries multiple

- la clausola di WHERE di una query puo' contenere una qualsiasi combinazione di condizioni normali e condizioni con subqueries
- esempio: si vuole elencare gli impiegati con la stessa mansione di Gianni o con uno stipendio maggiore o uguale a quello di Fordi. Inoltre si vuole ordinare il risultato in base alla mansione e allo stipendio.

```
SELECT Nome, Mansione, Stipendio
FROM Impiegati WHERE
        Mansione = (SELECT Mansione FROM Impiegati
                        WHERE Nome ='Gianni')
OR
        Stipendio ≥ (SELECT Stipendio FROM Impiegati
                        WHERE Nome = 'Fordi')
ORDER BY Mansione, Stipendio;
```

- una subquery puo' contenere a sua volta un'altra subquery
- esempio: elencare il nome e la mansione degli impiegati del dipartimento 10 con la stessa mansione di un qualche impiegato del dipartimento Ricerche

```
SELECT Nome, Mansione FROM Impiegati
   WHERE Dip#=10 AND Mansione IN
        (SELECT Mansione FROM Impiegati
             WHERE Dip# =
                  (SELECT Dip# FROM Dipartimenti
                       WHERE Nome_Dip = 'Ricerche'));
```

SQL - Subqueries correlate

- negli esempi visti ogni subquery viene eseguita una volta per tutte ed il valore (o insieme di valori) e' usato nella clausola WHERE della query esterna

- e' possibile definire subqueries che sono eseguite ripetutamente per ogni *tupla candidata* considerata nella valutazione della query esterna

- esempio: si vogliono determinare gli impiegati che guadagnano piu' dello stipendio medio del proprio dipartimento

- e' pertanto necessaria una query esterna che selezioni gli impiegati dalla relazione Impiegati in base ad un predicato su stipendio; tale query avrebbe la forma:

  SELECT Nome FROM Impiegati WHERE
  Stipendio >
  (*media degli stipendi nel dipartimento dell'impiegato candidato*);

- e' inoltre necessaria una subquery che calcoli la media degli stipendi del dipartimento di ogni tupla candidata della relazione Impiegati; tale subquery avrebbe la forma:

  (SELECT AVG(Stipendio) FROM Impiegati
  WHERE Dip#=(*valore di Dip# nella tupla candidata*));

SQL - Subqueries correlate

- ogni volta che la query esterna considera una tupla candidata, deve invocare la subquery e "passare" il numero di dipartimento dell'impiegato in esame

- la subquery calcola quindi la media degli stipendi nel dipartimento che e' stato passato e restituisce tale valore alla query esterna

- la query esterna puo' quindi confrontare lo stipendio dell'impiegato in esame con il valore restituito dalla subquery

- questo tipo di queries e' chiamato correlato, perche' ogni esecuzione della subquery e' correlata al valore di uno o piu' attributi delle tuple candidate nella interrogazione principale

- per poter riferire le colonne delle tuple candidate nella query esterna si fa uso degli alias di relazione; un alias di relazione e' definito nella query esterna e riferito nella query interna

SQL - Subqueries correlate

- esempio: la seguente query formula in SQL la query discussa precedentemente; si richiede inoltre l'ordinamento delle tuple del risultato

```
SELECT Dip#, Nome, Stipendio FROM Impiegati X
    WHERE Stipendio > (SELECT AVG(Stipendio)
                              FROM Impiegati WHERE
                              X.Dip# = Dip#)
    ORDER BY Dip#;
```

nella query X e' un alias di relazione (ovvero una variabile che denota una tupla e varia nella relazione Impiegati)

- due sono i concetti principali che sono alla base della correlazione

a) una subquery correlata fa riferimento ad un attributo selezionato dalla query esterna

b) se una subquery seleziona tuple dalla stessa relazione riferita dalla query esterna, e' necessario definire un alias per tale relazione della query esterna;
la subquery deve usare l'alias per riferire i valori di attributo nelle tuple candidate nella query principale

SQL - Subqueries correlate ALL ed ANY

- esempio: determinare i dipartimenti tali che tutti gli impiegati di tali dipartimenti guadagnino piu' di 1000

```
SELECT * FROM Dipartimenti X
   WHERE 1000 < ALL
        (SELECT Stipendio FROM Impiegati
             WHERE Dip#=X.Dip#);
```

- esempio: determinare i dipartimenti che abbiano almeno un impiegato che guadagna piu' di 1000

```
SELECT * FROM Dipartimenti X
   WHERE 1000 < ANY
        (SELECT Stipendio FROM Impiegati
             WHERE Dip#=X.Dip#);
```

nota: tale interrogazione si puo' semplicemente esprimere con un join

SQL - Subqueries correlate operatori EXISTS e NOT EXISTS

- il predicato EXISTS(sq) restituisce il valore Booleano True se la subquery restituisce almeno una tupla; restituisce il valore Booleano False altrimenti;

- il predicato NOT EXISTS(sq) restituisce il valore Booleano True se la subquery non restituisce alcuna tupla;
  restituisce il valore Booleano False altrimenti;

- nota: la valutazione di predicati con questi due operatori non restituisce mai il valore Unknown

- esempio: si supponga che la relazione Impiegati abbia una colonna addizionale, Risponde_a, che contiene per ogni tupla di Impiegati un numero di impiegato. Dato un impiegato E, tale numero indica l'impiegato a cui E risponde

  si vuole determinare il nome di tutti gli impiegati che hanno almeno un impiegato che risponde loro

```
SELECT Nome FROM Impiegati X
WHERE EXISTS (SELECT * FROM Impiegati
                 WHERE X.Imp# = Risponde_a);
```

  si vuole determinare il nome di tutti gli impiegati che non hanno alcun impiegato che risponda loro

```
SELECT Nome FROM Impiegati X
  WHERE NOT EXISTS
        (SELECT * FROM Impiegati
        WHERE X.Imp# = Risponde_a);
```


SQL - Divisione

- le subqueries correlate e l'operatore di NOT EXISTS sono particolarmente utili per implementare l'operazione di divisione in SQL

- la specifica della divisione in SQL richiede di ragionare in base al concetto di controesempio

- esempio: siano date le relazioni
  Progetti(Prog#, Pnome, Budget)
  Partecipanti(Dip#, Prog#)

  determinare i nomi dei dipartimenti che partecipano a tutti i progetti con un budget maggiore di 50,000

  un dipartimento X verifica l'interrogazione se non e' possibile determinare un progetto che ha un budget maggiore di 50,000 ed a cui il dipartimento X non partecipa

```
SELECT Nome_Dip FROM Dipartimenti X
WHERE NOT EXISTS
    (SELECT * FROM Progetti Y
    WHERE Budget > 50,000 AND
    NOT EXISTS (SELECT * FROM Partecipanti
                WHERE X.Dip# = Dip# AND
                Y.Prog#=Prog#));
```

SQL - Operazioni insiemistiche
Union

- una interrogazione o sottointerrogazione puo' essere costituita da una o piu' interrogazioni connesse dall'operatore UNION

- l'operatore UNION restituisce tutte <u>le tuple distinte</u> restituite da almeno una delle sottointerrogazioni a cui e' applicata

- esempio: si considerino le seguenti relazioni con lo stesso schema della relazione Impiegati
  Impiegati_Tempo_Parziale
  Impiegati_In_Congedo

  determinare gli impiegati che abbiano lo stesso stipendio di alcuni impiegati, opportunamente selezionati, contenuti nelle altre relazioni

```
SELECT Nome, Imp# FROM Impiegati
    WHERE Stipendio IN
    (SELECT Stipendio FROM Impiegati_Tempo_Parziale
          WHERE Mansione = 'tecnico'
    UNION
    SELECT Stipendio FROM Impiegati_In_Congedo
          WHERE Nome ='Viola');
```


SQL - Operazioni insiemistiche
Union

- l'operatore UNION impone alcune restrizioni sulle interrogazioni si cui opera

- le interrogazioni devono restituire lo stesso numero di colonne, e le colonne corrispondenti devono avere lo stesso dominio (non e' richiesto che abbiano la stessa lunghezza) o domini compatibili

- se si usa una clausola di ORDER BY questa deve essere usata una sola alla fine dell'interrogazione e *non alla fine di ogni* SELECT

- quando si specificano le colonne su cui eseguire l'ordinamento non si possono usare i nomi di colonna, poiche' questi potrebbero essere differenti nelle varie relazioni
  occorre indicarle specificandone la *posizione relativa* all'interno della clausola di proiezione

- esempio:

```
SELECT Nome,Imp# FROM Impiegati_Tempo_Parziale
    UNION
SELECT Nome,Imp# FROM Impiegati_In_Congedo
ORDER BY 2;
```

SQL - Operazioni insiemistiche Union

- l'operatore UNION elimina i duplicati dal risultato
- l'operatore UNION ALL non elimina i duplicati
- la ragione per includere UNION ALL e' che in molti casi l'utente sa che non ci saranno duplicati nel risultato
  in tal caso il tentativo da parte del sistema di eliminare i duplicati causa un peggioramento delle prestazioni

SQL - Operazioni insiemistiche Intersect ed Except

- gli operatori INTERSECT ed EXCEPT (detto anche MINUS) eseguono l'intersezione e la differenza
- per questi operatori valgono le stesse condizioni di applicabilita' viste per l'unione
- esempi:
  determinare i nomi degli impiegati a tempo parziale che sono in congedo

```
SELECT Nome FROM Impiegati_Tempo_Parziale
   INTERSECT
SELECT Nome FROM Impiegati_In_Congedo;
```

determinare i nomi degli impiegati a tempo parziale che non sono in congedo

```
SELECT Nome FROM Impiegati_Tempo_Parziale
   EXCEPT
SELECT Nome FROM Impiegati_In_Congedo;
```

SQL - Operazioni insiemistiche Intersect ed Except

- qualora i linguaggio SQL che si ha a disposizione non supporta gli operatori INTERSECT ed EXCEPT, tali operazioni possono essere eseguite mediante l'uso di EXISTS e NOT EXISTS

- esempio
  determinare i nomi degli impiegati a tempo parziale che sono in congedo

```
SELECT Nome FROM Impiegati_Tempo_Parziale X
   WHERE EXISTS
            (SELECT * FROM Impiegati_In_Congedo
            WHERE Imp#=X.Imp#);
```

determinare i nomi degli impiegati a tempo parziale che non sono in congedo

```
SELECT Nome FROM Impiegati_Tempo_Parziale X
   WHERE NOT EXISTS
            (SELECT * FROM Impiegati_In_Congedo
            WHERE Imp#=X.Imp#);
```


SQL - DML Inserzione

il comando di Insert ha il seguente formato

```
INSERT INTO R  [(C1,C2,....,Cn)]
       {VALUES (e1,e2,....,en) | sq};
```

dove:

- R e' il nome della relazione su cui si esegue l'inserzione
- $C_1,C_2,....,C_n$ e' la lista delle colonne dalla nuova tupla (o delle nuove tuple) a cui si assegnano valori

  tutte le colonne non esplicitamente elencate ricevono il valore nullo o il valore di default (se specificato nel comando di creazione di R)

  la mancata specifica di una lista di colonne equivale ad una lista che include tutte le colonne di R
- $e_1,e_2,....,e_n$ e' la lista di valori da assegnare alla nuova tupla

  i valori sono assegnati in base ad una corrispondenza posizionale; il valore e_i (i=1,...,n) e' assegnato alla colonna C_i
- sq e' una subquery (mutuamente esclusiva rispetto alla clausola VALUES)

  le tuple generate come risposta alla subquery vengono inserite nella relazione R

  la clausola di proiezione di sq deve contenere colonne (o piu' in generale espressioni) compatibili con le colonne di R a cui si assegnano valori

  pertanto il dominio della colonna C_i (i=1,...,n) deve essere compatibile con il dominio della colonna (o espressiione) i-esima contenuta nella clausola di proiezione di sq

SQL - DML Inserzione

- esempio di inserzione con valori espliciti

  si vuole inserire un nuovo dipartimento; il numero del dipartimento e' 40, il nome e' Edilizia Industriale; la divisione e' D2; il dirigente e' Blacchi (Imp# 7698); l'ufficio e' il numero 6100

  ```
  INSERT INTO Dipartimenti
     VALUES (40, 'Edilizia Industriale', 6100, 'D2',7698);
  ```

- esempio di inserzione con tuple i cui valori sono ottenuti tramite una subquery

  si vuole creare una relazione Promozioni
  tale relazione contiene alcune delle colonne della relazione Impiegati: Nome, Sipendio, Premio_P

  si vuole inserire in questa relazione tutti gli ingegneri il cui premio di produzione e' superiore al 25% del loro stipendio; le informazioni sugli ingegneri devono essere estratte dalla relazione Impiegati

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

  In questo caso gli ingegneri inseriti sono Rossi ed Adami

SQL - DML Cancellazione

il comando di Delete ha il seguente formato

```
DELETE FROM R [alias] [WHERE F];
```

dove:

- R e' il nome della relazione su cui si esegue la cancellazione
  il nome della relazione puo' avere associato un alias se e' necessario riferire a tuple di tale relazione in una qualche sottointerrogazione presente in F
- F e' la clausola di qualificazione che specifica le tuple da cancellare
- se non e' specificata alcuna clausola di qualificazione, vengono cancellate tutte le tuple

SQL - DML Cancellazione

- esempio

  si vuole cancellare il dipartimento 40

```
DELETE FROM Dipartimenti
   WHERE Dip# = 40;
```

- esempio di cancellazione con uso di subqueries

  si supponga che esista una relazione, di nome Bonus, che contenga tutti gli impiegati che hanno diritto ad aumenti di stipendio
  si vuole cancellare dalla relazione Bonus tutti gli impiegati che hanno la stessa mansione di Gianni

```
DELETE FROM Bonus
   WHERE Mansione IN
         (SELECT Mansione FROM Impiegati
               WHERE Nome = 'Gianni');
```


SQL - DML Modifica

il comando di UPDATE ha il seguente formato

```
UPDATE R [alias]
      SET C1={e1 | NULL}, ...., Cn={en | NULL}
[WHERE F];
```

dove:

- R e' il nome della relazione su cui si esegue la modifica
  il nome della relazione puo' avere associato un alias se e' necessario riferire a tuple di tale relazione in una qualche sottointerrogazione presente in F
- C_i = {e_i | NULL} (i=1,...,n) e' un'espressione di assegnamento che specifica che alla colonna C_i deve essere assegnato il valore dell'espressione e_i
  tale espressione puo' essere una costante, oppure un'espressione aritmetica o di stringa, spesso funzione dei valori correnti delle tuple da modificare, oppure una subquery
  alternativamente si puo' specificare che alla colonna sia assegnato il valore nullo
- F e' la clausola di qualificazione che specifica le tuple da modificare

  se non e' specificata alcuna clausola di qualificazione, vengono modificate tutte le tuple

SQL - DML Modifica

- esempio: si vuole aumentare di 100 lo stipendio di tutti i tecnici

```
UPDATE Impiegati
  SET Stipendio = Stipendio + 100
  WHERE Mansione = 'tecnico';
```

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

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


SQL - DML Modifica

nel comando di Update le subqueries possono essere usate per

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

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

```
UPDATE Impiegati
  SET Stipendio = 1.05*Stipendio
  WHERE Imp# IN
       (SELECT Imp# FROM Bonus);
```

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

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

SQL - Updates and Null values

- assegnamento di valori nulli nel comando di Update

- si usa la parola chiave NULL (da non confondere con la stringa 'NULL')

- UPDATE R
    SET B=NULL WHERE C=C2;
  risultato

R	A	B	C	
	a	?	c1	t1
	a1	?	c2	t2
	a2	?	?	t3

SQL - DML esempio complessivo

- supponiamo di creare una relazione Progetti con il seguente comando

```
CREATE TABLE Progetti (Prog# Decimal(3) NOT NULL,
                       Pnome Varchar(5),
                       Budget Decimal(7,2));
```

- supponiamo di inserire alcune tuple nella relazione Progetti

```
INSERT INTO Progetti VALUES (101,'Alpha', 96000);
INSERT INTO Progetti VALUES (102,'Beta', 82000);
INSERT INTO Progetti VALUES (103,'Gamma', 15000);
```

- poiche' ogni impiegato e' assegnato ad un progetto, si vuole estendere la relazione Impiegati con una nuova colonna che indichi il numero del progetto

```
ALTER TABLE Impiegati
      ADD COLUMN (Prog# Decimal(3));
```

- supponiamo di assegnare tutti i tecnici (di qualsiasi dipartimento) e tutti gli impiegati del dipartimento 20 al progetto 101

```
UPDATE Impiegati   SET Prog#=101
   WHERE Dip#=20 OR Mansione = 'tecnico';
```

SQL - DML esempio complessivo
(continua)

- supponiamo di assegnare tutti gli impiegati non assegnati ad alcun progetto al progetto 102

```
UPDATE Impiegati   SET Prog#=102
   WHERE Prog# IS NULL;
```

Impiegati

Imp#	Nome	Mansione	Data_A	Stipendio	Premio_P	Dip#	Proj#
7369	Rossi	ingegnere	17-Dic-80	1600,00	500,00	20	101
7499	Andrei	tecnico	20-Feb-81	800,00	?	30	101
7521	Bianchi	tecnico	20-Feb-81	800,00	100,00	30	101
7566	Rosi	dirigente	02-Apr-81	2975,00	?	20	101
7654	Martini	segretaria	28-Set-81	800,00	?	30	102
7698	Blacchi	dirigente	01-Mag-81	2850,00	?	30	102
7782	Neri	ingegnere	01-Giu-81	2450,00	200,00	10	102
7788	Scotti	segretaria	09-Nov-81	800,00	?	20	101
7839	Dare	ingegnere	17-Nov-81	2000,00	300,00	10	102
7844	Turni	tecnico	08-Set-81	1500,00	?	30	101
7876	Adami	ingegnere	28-Set-81	1100,00	500,00	20	101
7900	Gianni	ingegnere	03-Dic-81	1950,00	?	30	102
7902	Fordi	segretaria	03-Dic-81	1000,00	?	20	101
7934	Milli	ingegnere	23-Gen-82	1300,00	150,00	10	102
7977	Verdi	dirigente	10-Dic-80	3000,00	?	10	102

- supponiamo infine di voler aumentare la lunghezza della colonna Budget

```
ALTER TABLE Progetti
     ALTER COLUMN (Budget Decimal(9,2));
```


SQL - Viste

in SQL e' possibile definire viste alternative degli stessi dati

- una vista (**view**) e' una relazione virtuale (simile ad una window) attraverso cui e' possibile vedere i dati memorizzati nelle relazioni reali (dette **di base**)

- una view non contiene tuple, ma puo' essere usata a quasi tutti gli effetti come una relazione di base

- una view e' definita da una query su una o piu' relazioni di base o altre view

- una view e' materializzata eseguendo la query che la definisce

- il meccanismo delle views e' utile per

  - semplificare l'accesso ai dati
  - fornire indipendenza logica
  - garantire la privatezza dei dati

SQL - Viste

il comando di creazione di viste ha il seguente formato

```
CREATE VIEW V [(Lista Nomi Colonne)] AS Q
   [WITH [LOCAL| CASCADED] CHECK OPTION];
```

dove:

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

- Q e' l'*interrogazione di definizione* della vista
  una vista ha lo stesso numero di colonne pari alle colonne (di base o virtuali) specificate nella clausola di proiezione di Q

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

SQL - Viste

- esempio: si vuole creare una vista costituita da un sottoinsieme delle tuple della relazione Impiegati;
  piu' precisamente la vista deve elencare le colonne
  Imp#, Nome e Mansione
  degli impiegati del dipartimento 10

```
CREATE VIEW Imp10 AS
   SELECT Imp#, Nome, Mansione
      FROM Impiegati WHERE Dip#=10;
```

- i nomi delle colonne della vista Imp10 sono rispettivamente: Imp#, Nome, Mansione

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

- esempio: selezionare le tuple della vista Imp10

  SELECT * FROM Imp10;

  risultato:

Imp#	**Nome**	**Mansione**
7782	Neri	ingegnere
7839	Dare	ingegnere
7934	Milli	ingegnere
7977	Verdi	dirigente

SQL - Viste: uso di joins

- puo' essere facile per alcuni utenti lavorare con una sola relazione piuttosto che eseguire joins tra relazioni diverse

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

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


SQL - Viste: uso di espressioni e funzioni

- e' possibile definire viste tramite interrogazioni che contengono espressioni o funzioni

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

- tali colonne sono spesso chimate colonne virtuali; e' obbligatorio specificare un nome nella vista per tali colonne

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

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

se si esegue l'interrogazione

```
SELECT * FROM V1 WHERE Dip#=30;
```

si ottiene il seguente risultato

Nome	Stipendio_Mensile	Sipendio_Annuale	Dip#
Andrei	800,00	9600,00	30
Bianchi	800,00	9600,00	30
Martini	800,00	9600,00	30
Blacchi	2850,00	34200,00	30
Turni	1500,00	18000,00	30
Gianni	1950,00	23400,00	30

SQL - Viste: uso di espressioni e funzioni

- e' possibile definire viste tramite interrogazioni che contengono funzioni di gruppo e clausole di GROUP BY

- esempio: si vuole definire una vista che calcoli per ogni dipartimento alcune statistiche riguardanti lo stipendio degli impiegati

```
CREATE VIEW Dip_S
   (Dip#,Min_S,Med_S, Max_S,Totale) AS
        SELECT Dip#, MIN(Stipendio), AVG(Stipendio),
                MAX(Stipendio), SUM(Stipendio)
            FROM Impiegati GROUP BY Dip#;
```

se si esegue l'interrogazione

```
SELECT Dip#,Min_S,Max_S,Totale FROM Dip_S;
```

si ottiene il seguente risultato

Dip#	**Min_S**	**Max_S**	**Totale**
10	1300,00	3000,00	8750,00
20	800,00	2975,00	7445,00
30	800,00	2850,00	8700,00

SQL - Uso di viste

- una volta creata, una vista puo' essere considerata (quasi) una relazione di base

- e' possibile eseguire qualsiasi interrogazione su una vista con una importante restrizione:

  non e' possibile l'uso di funzioni di gruppo su colonne di viste definite tramite funzioni di gruppo

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

  tale operazione e' detta *view composition*

  esempio: si consideri la vista Personale e la query

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

  la query che si ottiene dopo la composizione e' la seguente

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

- una vista puo' essere definita su una o piu' relazioni e/o viste

SQL - Check option per views

- una view essendo definita da una query contiene condizioni sul contenuto delle tuple

- solo le tuple che verificano una certa condizione sono restituite dalla view

- un problema e' cosa succede se in una view in cui si puo' eseguire l'insert, si esegue l'inserzione di una tupla che non verifica la condizione specificata dalla query nella view

- per esempio; si consideri la seguente view

```
CREATE VIEW imp-r AS
SELECT Imp#, Nome, Stipendio FROM Impiegati
WHERE Stipendio > 3000;
```

- supponiamo di inserire nella view la seguente tupla
  (200, Haas, 2000)

  la condizione specificata nella query non e' verificata dalla nuova tupla; pertanto la tupla viene inserita ma non e' ritrovata da una query sulla view

- per assicurare che le tuple inserite tramite una view (o modificate tramite una view) siano accettate solo se verificano la condizione nella query della view, si usa la CHECK OPTION

- il formato della definizione di una view e' pertanto il seguente:

```
CREATE VIEW view-name [(colum-names)] AS
query WITH [LOCAL|CASCADED] CHECK OPTION]
```


SQL - Check option per views

- se la view precedente e' definita come segue

```
CREATE VIEW imp-r AS
SELECT Imp#, Nome, Stipendio FROM Impiegati
WHERE Stipendio > 3000
WITH CHECK OPTION;
```

  l'inserzione sulla view della tupla
  (200, Haas, 2000)
  non viene eseguita

- la differenza tra LOCAL e CASCADE e' rilevante nei casi in cui una view e' definita in termini di un'altra view

- sia View1 una view definita in termini di un'altra view View2 (View2 e' detta view sottostante a View1)

- se View1 e' definita con una **check option locale**, le inserzioni eseguite su View1 devono verificare
  (1) la definizione di View1
  (2) la definizione di View2 solo se View2 e' a sua volta definita con la check option
  (se View2 non e' definita con la check option allora la definizione di View2 non deve essere verificata dalla tupla inserita)

- se View1 e' definita con una **check option cascaded**, le inserzioni eseguite su View1 devono verificare
  (1) la definizione di View1
  (2) la definizione di View2 (indipendentemente dal fatto che View2 sia definita con check option o no)

SQL - Modifiche su viste

Problema 1: inserimento

-

```
CREATE VIEW V3 AS
SELECT Imp#, Nome, Mansione,
Data_A, Dip# FROM Impiegati
WHERE Mansione < > 'dirigente';
```

- lo schema di V3 e'
  (Imp#, Nome, Mansione, Data_A, Dip#)

- supponiamo di inserire nella view la seguente tupla
  (8001, 'Smith', 'Tecnico', '13-Dic-91', 20)

  questa insert viene trasformata in una operazione di insert nella relazione Impiegati

- il problema e' quale valore assegnare agli attributi Stipendio e Premio_P

- una soluzione e' assegnare il valore nullo

- questa soluzione crea problemi se il valore di Stipendio e Premio_P deve essere diverso dal valore nullo

- questo problema giustifica la restrizione sull'inserzione

SQL - Modifiche su viste

Problema 2: cancellazione

-

```
CREATE VIEW V4 AS
    SELECT Nome, Ufficio
    FROM Impiegati, Dipartimenti
    WHERE Impiegati.Dip# = Dipartimenti.Dip#;
```

- lo schema di V4 e'
  (Nome, Ufficio)

- supponiamo di eseguire la seguente cancellazione

```
DELETE FROM V4 WHERE Nome = 'Rossi';
```

- questa cancellazione potrebbe essere tradotta come segue

```
DELETE FROM Impiegati WHERE Nome = 'Rossi';
DELETE FROM Dipartimenti
            WHERE Ufficio = 2100;
```

SQL - Modifiche su viste

Problema 2: cancellazione

- stato della view prima della cancellazione

    SELECT * FROM V4;

Nome	Ufficio
Rossi	2100
Andrei	5100
Bianchi	5100
Rosi	2100
Martini	5100
Blacchi	5100
Neri	1100
Scotti	2100

- stato della view dopo la cancellazione

Nome	Ufficio
Andrei	5100
Bianchi	5100
Martini	5100
Blacchi	5100
Neri	1100

- la modifica precedente ha <u>side-effects</u> sulla view

- se l'utente esegue una select sulla view dopo la modifica tutte le tuple dello stesso ufficio di Rossi sono sparite dalla view

SQL - Modifiche su viste

Problema 2: cancellazione

- il problema e' che il mapping dell'operazione in termini di operazioni sulle relazioni di base puo' essere eseguito in modi diversi

  (a) una cancellazione su V4 equivale ad una cancellazione sia su Impiegato che su Dipartimento

  (b) una cancellazione su V4 equivale ad una cancellazione su Impiegato

  (c) una cancellazione su V4 equivale ad un update su Impiegato che assegna Null al valore di Dip#

- a causa dell'ambiguita' del mapping, un regola usata nei DBMS e' che non vengono ammesse modifiche su views definite tramite joins

- il problema dell'update nelle views e' molto studiato ed esistono le proposte piu' disparate

SQL - Modifiche su viste

- l'esecuzione di una operazione di modifica su una vista e' propagata alla relazione su cui la vista e' definita

- valgono le seguenti restrizioni:

1) e' possibile eseguire l'operazione di DELETE se l'interrogazione di definizione della vista soddisfa le seguenti condizioni:

   - e' su una sola relazione
   - non contiene la clausola GROUP BY, la clausola DISTINCT, o una funzione di gruppo

2) e' possibile eseguire l'operazione di UPDATE se l'interrogazione di definizione della vista soddisfa le due condizioni precedenti ed inoltre:

   - la colonna modificata non e' definita da un'espressione

3) e' possibile eseguire l'operazione di INSERT se l'interrogazione di definizione della vista soddisfa le tre condizioni precedenti ed inoltre:

   - qualsiasi colonna per cui valga il vincolo NOT NULL sia presente nella vista

SQL - Cancellazione e renaming di viste

- formato comando di cancellazione

```
DROP VIEW V;
```

dove V e' il nome della vista da cancellare

- formato comandi di renaming

  RENAME V_v to V_n

- nota: non e' possibile, invece, modificare colonne di viste; l'unico modo e' ridefinire l'interrogazione di definizione

SQL - Viste inoperative

- una questione importante riguarda qual'e' lo stato di una view V quando una tabella (o view) usata nella query di definizione della view e' cancellata

- esistono due approcci:

  (a) V e' a sua volta cancellata
  (b) V e' resa **inoperativa**

- una view inoperativa e' una view tale che:
  i. la sua definizione viene mantenuta nei cataloghi di sistema (syscat.views in DB2)
  ii. non e' possibile eseguire alcuna operazione sulla view tranne l'operazione di DROP

- lo stato inoperativo di una view e' denotato da un particolare flag rappresentato dalla colonna STATUS del catalogo syscat.views

- una view inoperativa e' automaticamente eliminata se si esegue un comando di CREATE VIEW di una view con lo stesso nome

- la precedente e' l'unica eccezione alla restrizione che non si possono creare due views con lo stesso nome

SQL - Views inoperative

- esempio:

```
CREATE VIEW imp-r AS
   SELECT Imp#, Nome, Stipendio FROM Impiegati
   WHERE Stipendio > 3,000;
```

```
DROP TABLE Impiegati;
```

la view imp-r diventa inoperativa

supponiamo che si crei una nuova tabella di nome Employees

il comando

```
CREATE VIEW imp-r AS SELECT Imp#, Nome,
Stipendio FROM Employees
   WHERE Stipendio > 3,000;
```

cancella la vecchia definizione della view imp-r con la nuova

- nota: questo e' l'unico caso in cui si puo' definire una vista con il nome di un'altra vista gia' definita

SQL - Accesso da linguaggio di programmazione

concetti di base

- il linguaggio SQL puo' essere usato come linguaggio "ospitato" in un linguaggio di programmazione (detto *linguaggio ospite*)

- per poter usare SQL da un linguaggio di programmazione sono necessarie alcune funzionalita' aggiuntive ad SQL, ad esempio per trasferire risultati di interrogazioni in variabili di programma;
  questa variante di SQL e' detta SQL ospitato (*embedded SQL*)

- fasi nella compilazione di un programma P scritto in SQL ospitato e linguaggio ospite

  (1) pre-compilazione di P

  - il programma viene passato ad un pre-compilatore (specifico del linguaggio ospite);

  - il precompilatore elimina gli statements di SQL sostituendoli con chiamate di procedura (al DBMS) espresse nella sintassi del linguaggio ospite;

  Il risultato e' un programma scritto completamente nel linguaggio ospite

  (2) compilazione del programma risultato della fase (1)

SQL - Accesso da linguaggio di programmazione

concetti di base

1) ogni istruzione di SQL ospitato deve essere preceduta dalla stringa EXEC SQL; questa stringa permette al precompilatore di distinguere un'istruzione SQL dalle istruzioni del linguaggio ospite

2) ogni istruzione di SQL deve terminare con un carattere speciale che dipende dallo specifico linguaggio ospite
(esempi:

	C	;
	PL/I	;
	Cobol	END-EXEC)

3) un'istruzione SQL <u>eseguibile</u> puo' comparire dovunque possa comparire uno statement del linguaggio ospite; diversamente da SQL interattivo, l'SQL ospitato fornisce istruzioni <u>dichiarative</u> (ad es. DECLARE CURSOR)

4) le variabili del inguaggio ospite possono essere usate in istruzioni SQL in accordo a quanto segue:

- nella clausola INTO del comando di SELECT
  (in questo caso le variabili conterranno il risultato della query)
- nella clausola WHERE dei comandi di SELECT, UPDATE, DELETE
  (in questo caso le variabili contengono costanti da confrontare con gli attributi delle tuple)
- nella clausola SET del comando di UPDATE
  (in questo caso le variabili contengono i valori da assegnare alle tuple modificate)
- nella clausola VALUES del comando di INSERT
  (in questo caso le variabili contengono i valori da assegnare alle nuove tuple)
- nelle espressioni aritmetiche, di stringa e temporali

SQL - Accesso da linguaggio di programmazione

concetti di base (continua)

5) inizialmente un programma SQL ospitato deve eseguire una connessione al DBMS

```
EXEC SQL CONNECT user_name
         IDENTIFIED BYuser_psw
```

6) SQLCA: SQL Communication Area

- dopo l'esecuzione di un'istruzione SQL, il DBMS restituisce informazioni di "feedback"
- tali informazioni sono memorizzate dal DBMS nella SQLCA (deve essere inclusa tramite l'istruzione
  EXEC SQL INCLUDE SQLCA)
- un campo della SQLCA e' il SQLCODE (SQLSTATUS) a cui il DBMS assegna un codice di ritorno a seguito dell'esecuzione di un comando SQL; in particolare:
  - un valore 0 indica che il comando e' stato eseguito correttamente
  - un valore positivo (ad es. +100) indica che il comando e' stato eseguito correttamente ma si e' verificata qualche condizione eccezionale
    (+100 indica che non e' stata trovata alcuna tupla che verifica la query)
  - un valore negativo indica che il comando non e' stato terminato a causa di qualche errore
    (ad esempio una query su una relazione non esistente)

  in linea di principio, ogni comando SQL dovrebbe essere seguito da un test su SQLCODE, in modo da prendere le opportune decisioni

SQL - Accesso da linguaggio di programmazione

concetti di base (continua)

7) uso del *cursore*

- l'esecuzione di comandi di SELECT puo' restituire piu' di una tupla
- e' necessario usare un meccanismo che permette di accedere al risultato "una-tupla-alla-volta"
- tale meccanismo e' fornito dal cursore, un nuovo oggetto SQL non presente in SQL interattivo
- consiste di un puntatore che scorre lungo un insieme di tuple e che permette l'accesso di una tupla alla volta
- non tutti i comandi SQL richiedono l'uso del cursore; questi includono i seguenti domandi:
  - "singleton" SELECT
    (SELECT che restituisce al piu' una tupla)
  - UPDATE (eccetto nella forma CURRENT OF)
  - DELETE (eccetto nella forma CURRENT OF)
  - INSERT
  - tutti i comandi di DDL
  - tutti i comandi di autorizzazione

SQL - Accesso da linguaggio di programmazione

esempio

base di dati che contiene informazioni su prodotti, i fornitori che li forniscono, ed i progetti a cui i prodotti sono forniti

Schema della base di dati

Fornitori(Forn#,NomeF, Volume, Citta)

Parti(P#, Nome, Colore, Peso)

Progetti(Pr#, Nome, Citta)

Invii(Forn#, P#, Pr#, Quantita)

SQL - Accesso da linguaggio di programmazione

esempio

```
#include <stdio.h>
#include <ctype.h>

EXEC SQL INCLUDE SQLCA; /* inclusione della communication area */
int prompt1(char[~~], char[~~], int);
char prompt[  ] = "IMMETTERE NUMERO FORNITORE:";

main ( )
{
        EXEC SQL BEGIN DECLARE SECTION
                                        /* dichiarazione variabili ospiti */
                VARCHAR dato_f[5], citta[15];
                short volume;
                VARCHAR nome_utente[20], pass_utente[10];
        EXEC SQL END DECLARE SECTION;
                                /* inizializza nome e password utente
                                per la connessione alla base di dati */
        strcpy(nome_utente.arr, "bertino");
        nome_utente.len = strlen("bertino");
        strcpy(pass_utente.arr, "corsodb*");
        pass_utente.len = strlen("corsodb*");

        EXEC SQL CONNECT :nome_utente IDENTIFIED BY pass_utente;
        while((prompt1(prompt, dato_f.arr, 5)) > = 0)
        {
                dato_f.len = strlen(dato_f.arr);
                EXEC SQL SELECT Volume, Citta FROM Fornitori
                        INTO :volume, :citta
                        WHERE Forn#=:dato_f;
                printf("Il volume del fornitore e' %d e la citta' e' %s",
                        volume, citta_arr)};
        }
        EXEC SQL COMMIT RELEASE;
}
```

SQL - Accesso da linguaggio di programmazione

Nota sul passaggio di valori di tipo stringa tra SQL e linguaggio di programmazione

- in C le stringhe sono implementate come array di caratteri terminanti con \0 (rappresentazione *null-terminated*)

- nei DBMS le stringhe vengono rappresentate in modo diverso
  - per il tipo CHAR, si alloca una stringa comunque pari alla lunghezza massima, inizializzando a ' ' gli ultimi caratteri se la stringa e' piu' corta
    es: CHAR(4)   stringa c9   'c', '9', ' ', ' '
  - per il tipo VARCHAR si usa una rappresentazione detta *counted string* in cui all'inizio della stringa sono allocati due bytes che contengono la lunghezza della stringa

- alcuni DBMS (ad es. Ingres e DB2) effettuano automaticamente la conversione tra i due formati delle stringhe

- Oracle non esegue la conversione che deve essere eseguita dal programma applicativo quando deve passare stringhe ad Oracle

- Oracle richiede che nella DECLARE SECTION le variabili ospiti di tipo stringa siano dichiarate di tipo VARCHAR dove VARCHAR e' una struct del C che consiste di un array di carattere e di un intero

- esempio: VARCHAR dato_f[5] viene espansa in

```
struct {
        unsigned short len;
        unsigned char arr[5];
} dato_f;
```

- nota: una variabile deve avere lunghezza maggiore almeno di 1 rispetto alla colonna corrispondente

SQL - Operazioni senza cursore

"Singleton SQL"

```
EXEC SQL SELECT Volume, Citta
      FROM Fornitori
      INTO :volume, :citta
      WHERE Forn# = Dato_f#;
```

- il termine "singleton SQL" indica una interrogazione che restituisce una singola tupla

- se esiste una ed una sola tupla che verifica l'interrogazione, allora le due variabili :volume, e :citta ricevono i valori corrispondenti degli attributi Volume e Citta

- se non ci sono tuple, SQLCODE e' settato a +100

- se c'e' piu' di una tupla che verifica l'interrogazione, il programma e' considerato in errore e SQLCODE ha un valore negativo

- nelle ultime due situazioni le variabili di programma :volume, e :citta non vengono modificate

SQL - Operazioni senza cursore: valori nulli

- i linguaggi di progammazione non hanno la nozione di valore nullo per i propri tipi di dato

- pertanto se uno degli attributi restituiti dalla interrogazione ha valore nullo, si ha una condizione di errore (SQLCODE < 0) e le variabili di programma restano inalterate

- per evitare situazioni di errore, occorre usare le variabili indicatore; una variabile indicatore viene associata ad una variabile di programma nella clausola INTO del comando di SELECT

- le variabili indicatore devono essere dichiarate come variabili intere nella DECLARE SECTION e non possono essere usate nella clausola di qualificazione delle interrogazioni

- la variabile indicatore riceve:
  - valore uguale a -1 se la corrispondente colonna ha valore nullo
    (la corrispondente variabile non viene inizializzata)
  - valore uguale a 0 se la corrispondente colonna ha valore diverso da nullo
    (la corrispondente variabile viene inizializzata)

- esempio:

```
EXEC SQL SELECT Volume, Citta FROM Fornitori
      INTO :volume :volumeind, :citta :cittaind
      WHERE Forn# = :dato_f;

if (volumeind < 0) {.........}  /* volume e' nullo*/
if (cittaind < 0) {.........}       /* citta e' nullo*/
```


SQL - Operazioni senza cursore

- UPDATE: supponiamo di incrementare il volume delle forniture dei fornitori di Londra di un ammontare contenuto nella variabile di programma aumento

```
EXEC SQL UPDATE Fornitori
      SET Volume = Volume +:aumento
      WHERE Citta = 'Londra';
```

le variabili indicatore possono essere usate per indicare che ad una certa colonna deve essere assegnato un valore nullo

```
EXEC SQL UPDATE Fornitori
      SET Citta = :citta :cittaind
      WHERE Forn#=200;
```

se prima dell'esecuzione del comando di UPDATE :cittaind riceve valore negativo, allora alla colonna Citta viene assegnato valore nullo,
se :cittaind riceve 0, allora alla colonna Citta viene assegnato il valore contenuto nella variabile di programma :citta

SQL - Operazioni senza cursore

- DELETE: cancellare tuti gli invii dei fornitori la cui citta' e' data dalla variabile :citta

```
EXEC SQL DELETE FROM Invii X
        WHERE :citta = (SELECT DISTINCT Citta
                        FROM Fornitori
                        WHERE X.Forn#=Forn#);
```

- INSERT: si vuole inserire una nuova tupla nella relazione PARTI;
  i valori delle colonne P#, Pnome, e Peso sono contenuti rispettivamente nella variabili di programma
  pno, pnome, ps
  nessun valore e' assegnato alla colonna Colore che riceve pertanto un valore nullo

```
EXEC SQL INSERT INTO Parti (P#, Pnome, Peso)
   VALUES (:pno, :pnome, :ps);
```

- siano inoltre pcolore e pcoloreind una variabile ospite e una variabile indicatore, rispettivamente, il seguente frammento di programma ha lo stesso effetto del comando precedente

```
 pcoloreind = -1;
EXEC SQL INSERT INTO Parti
VALUES (:pno, :pnome, :pcolore :pcoloreind, :ps);
```

- l'uso di variabili indicatore nel comando di INSERT e' particolarmente utile se non e' noto a priori quali sono le colonne delle nuove tuple che ricevono valori nulli

SQL - Operazioni con cursore

- il cursore deve essere usato per queries che restituiscono piu' di una tupla come risultato

- un cursore e' associato ad una query ed e' dichiarato tramite un'istruzione il cui formato e' il seguente:

  EXEC SQL DECLARE CURSOR NomeC FOR Q;

  dove: NomeC e' il nome del cursore
  Q e' un comando di SELECT

- la dichiarazione di un cursore non ha l'effetto di eseguirlo; l'interrogazione e' "eseguita" tramite un'apposita istruzione, il cui formato e':

  EXEC SQL OPEN NomeC;

  tale comando associa un insieme attivo di tuple ad X

- le tuple sono quindi prelevate e copiate nelle variabili di programma tramite l'istruzione

  EXEC SQL FETCH NomeC INTO ListaVariabili;

  la variabili possono avere associate delle variabili indicatori per la segnalazione dei valori nulli

- un cursore e' disattivato tramite il comando

  EXEC SQL CLOSE NomeC;

  uno stesso cursore puo' essere attivato una seconda volta (o piu' volte) durante l'esecuzione di uno stesso programma, dopo essere stato pero' disattivato

SQL - Operazioni con cursore esempio

```
#include <stdio.h>
#include <ctype.h>

EXEC SQL INCLUDE SQLCA; /* inclusione della communication area */
int prompt1(char[~~], char[~~], int);
char prompt[  ] = "IMMETTERE NUMERO FORNITORE:";

main ( )
{
        EXEC SQL BEGIN DECLARE SECTION
                                        /* dichiarazione variabili ospiti */
                VARCHAR f[5], citta_data[15], nomef[20];
                short volume;
                VARCHAR nome_utente[20], pass_utente[10];
        EXEC SQL END DECLARE SECTION;
        EXEC SQL DECLARE X CURSOR FOR
                SELECT Forn#, Nomef, Volume FROM Fornitori
                                WHERE Citta = :citta_data;
                                        /* inizializza nome e password utente
                                        per la connessione alla base di dati */
        strcpy(nome_utente.arr, "bertino");
        nome_utente.len = strlen("bertino");
        strcpy(pass_utente.arr, "corsodb*");
        pass_utente.len = strlen("corsodb*");

        EXEC SQL CONNECT :nome_utente IDENTIFIED BY pass_utente;
        while((prompt1(prompt, citta_data.arr, 15)) > = 0)
        {
                citta_data.len = strlen(citta_data.arr);
                EXEC SQL OPEN X;
                        while "ci sono tuple"
                        {
                                /* preleva una tupla del risultato */
                                EXEC SQL FETCH X
                                        INTO :f, :nomef, :volume;
                                "stampa i valori della tupla";
                                /* fine ciclo while interno */
                        }
                EXEC SQL CLOSE X;
        }       /* fine ciclo while esterno */
        EXEC SQL COMMIT RELEASE;
}
```


SQL - Operazioni con cursore

- e' possibile avere un riferimento ad un cursore nei comandi di UPDATE e DELETE
- in particolare se un cursore X e' posizionato su una determinata tupla e' possibili modificare o cancellare "la tupla corrente di X"
- i comandi di UPDATE e DELETE con riferimento ad un cursore hanno il seguente formato:

```
EXEC SQL UPDATE R [alias]
        SET C1={e1 | NULL}, ...., Cn={en | NULL}
        WHERE CURRENT OF NomeC;

EXEC SQL DELETE FROM R [alias]
        WHERE CURRENT OF NomeC;
```

- sia X il nome di un cursore; si vuole modificare di un ammontare contenuto nella variabile di programma aumento, la tupla corrente del cursore X

```
EXEC SQL UPDATE Fornitori
        SET Volume = Volume +:aumento
    WHERE CURRENT OF X;
```

SQL - Operazioni con cursore esempio

- un programma che riceve in ingresso quattro parametri:

  (i) un numero di parte, contenuto nella variabile di programma p#

  (ii) un nome di citta, contenuto nella variabile di programma citta_data

  (iii) un incremento, contenuto nella variabile di programma incr

  (iv) un livello di volume, contenuto nella variabile di programma liv_volume

- il programma esegue le seguenti attivita':

  - esamina tutti i fornitori della parte il cui numero e' dato;
  - se un fornitore fornisce tale parte e la sua citta' e' uguale alla citta' data, il suo volume e' incrementato del livello dato;
  - altrimenti, se il suo volume e' minore del livello dato, il fornitore con tutti i suoi invii e' cancellato;
  - infine il programma stampa tutti i fornitori indicando in che modo ogni fornitore e' stato trattato

SQL - Operazioni con cursore: esempio

```
#include <stdio.h>
#include <ctype.h>
EXEC SQL INCLUDE SQLCA;                /* inclusione della communication area */
main ( )
{
  EXEC SQL BEGIN DECLARE SECTION;     /* dichiarazione variabili ospiti */
      VARCHAR nomef[20], p#[5], forn#[5], cittaf[15];
      short volume, incr;
      VARCHAR nome_utente[20], pass_utente[10];
  EXEC SQL END DECLARE SECTION;
  char citta_data[15], disp[10];
  short liv_voume;
  EXEC SQL DECLARE Z CURSOR FOR
      SELECT Forn#, Nomef, Volume, Citta FROM Fornitori
      WHERE EXISTS (SELECT * FROM Invii WHERE
                    Fornitori.Forn#= Invii.Forn# AND P# = :p#)
      FOR UPDATE OF Volume;
  EXEC SQL WHENEVER NOTFOUND CONTNUE;
  EXEC SQL WHENEVER SQLERROR GOTO errpt;
  EXEC SQL WHENEVER SQLWARNING CONTNUE;
          /* inizializza nome e password utente per la connessione alla base di dati */
  strcpy(nome_utente.arr, "bertino");
  nome_utente.len = strlen("bertino");
  strcpy(pass_utente.arr, "corsodb*");
  pass_utente.len = strlen("corsodb*");

  EXEC SQL CONNECT :nome_utente IDENTIFIED BY :pass_utente;
                        /* lettura dei parametri di ingresso del programma */
  getlist(p#, citta_data, incr, liv_volume);
  EXEC SQL OPEN Z;
  if (SQLCODE ==100) goto notfound;
  save = SQLCA.SQLERRD[2];
  while (save > 0)
      {     EXEC SQL FETCH Z INTO :form#, :nomef, :volume, :cittaf;
            save = save -1;
            strcpy(disp, "inalterato");
            if (strcmp(cittaf, citta_data))
                  { EXEC SQL UPDATE Fornitori
                          SET Volume = Volume + :incr WHERE CURRENT OF Z;
                    strcpy(disp, "aggiornato");}
            else
            { if (volume <  liv_volume)
                  { EXEC SQL DELETE FROM Fornitori WHERE  CURRENT OF Z;
                    EXEC SQL DELETE FROM Invii WHERE Forn# = :forn#;
                    strcpy(disp, "cancellato");
                  }
            }
  print ("\n, \%s, \%s, \%d, \%s, \%s", forn#, nomef, volume, cittaf, disp);
  }
  EXEC SQL CLOSE Z;
  EXEC SQL COMMIT RELEASE;
  return(OK);
  notfound: return(NOT_FOUND);
  errpt: EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL ROLLBACK WORK RELEASE;  }
```

SQL - Operazioni con cursore

il programma precedente illustra alcuni aspetti addizionali

- il comando WHENEVER permette di semplificare la gestione dell'errore; ha il formato
  EXEC SQL WHENEVER Cond Azione
  dove
  - Cond indica la condizione controllata dal comando WHENEVER; puo' avere uno dei seguenti valori

    NOTFOUND significa SQLCODE = 100
    SQLWARNING significa
    SQLCODE > 0 e SQLCODE ≠ 100
    SQLERROR significa SQLCODE < 0

  - Azione e' l'azione da eseguire al seguito del verificarsi della condizione; puo' avere uno dei seguenti valori

    CONTINUE che richiede di continuare l'esecuzione
    GOTO label che richiede di eseguire l'istruzione la cui etichetta e' label

    (alcuni sistemi includono STOP e CALL)

- il comando WHENEVER non e' un comando eseguibile; e' solo una direttiva al precompilatore che genera del codice opportuno che viene inserito dopo ogni istruzione SQL

- in particolare:

  WHENEVER condition GOTO
  causa l'inserzione di un comando di goto dopo il comando SQL

  WHENEVER condition CONTINUE
  il precompilatore non inserisce alcun comando

SQL - Operazioni con cursore

- un programma puo' avere piu' comandi di WHENEVER
  un comando di WHENEVER per una condizione esegue l'overriding del comando precedente di WHENEVER relativo alla stessa condizione

- nota: il pre-compilatore non analizza il flusso del programma
  quindi l'overriding e' basato sulla "ordinamento fisico" dei comandi come linee di codice nel programma e non sull'ordine effettivo di esecuzione

- esempio:

```
main ( )
{
        EXEC SQL WHENEVER SQLERROR STOP;
                        /* primo comando WHENEVER */
        ..........
        goto s1;
        ..........
        EXEC SQL WHENEVER SQLERROR CONTINUE;
                /* sovrascrive il primo comando WHENEVER */

s1:     EXEC SQL UPDATE R SET col1 = col1+5;
        ...........
}
```

quando si esegue il comando UPDATE, l'azione che viene eseguita a fronte di errori e' CONTINUE, anche se il comando di UPDATE e' stato raggiunto tramite un goto nello "scope" del primo comando di WHENEVER

SQL - Operazioni con cursore

- quando un programma esegue delle modifiche, tali modifiche sono tentative nel senso che se si hanno errori le modifiche sono cancellate dalla base di dati

- a fronte di errori lo stato della base di dati e' ripristinato allo stato precedente all'inizio della transazione
  ad esempio si ha un overflow ed il programma termina in modo anormale

- le modifiche sono tentative fino a che si verifica una delle due seguenti condizioni

  - e' eseguito un comando di COMMIT che rende definitive le modifiche

  - e' eseguito un comando di ROLLBACK che annulla tutte le modifiche eseguite

- nell'esempio, il programma esegue un COMMIT quando raggiunge la sua conclusione normale;
  esegue un ROLLBACK se si verificano errori

SQL Dinamico

- l'SQL dinamico consiste di un insieme di funzionalita' che hanno lo scopo di supportare la costruzione di applicazioni "on-line"

- un'applicazione "on-line" e' un programma che permette l'accesso alla base di dati da parte di utenti connessi tramite terminali

- una tipica applicazione "on-line" puo' essere schematizzata come segue:
  1. accetta un comando da terminale
  2. analizza il comando
  3. genera ed invia l'appropriato comando SQL al DBMS
  4. restituisce a terminale un messaggio e/o i risultati

- se l'insieme dei comandi che possono essere immessi a terminale e' piccolo, si puo' "cablare" nel programma applicativo l'insieme di tutti i possibili comandi di ingresso
  il programma applicativo sara' organizzato con una logica a casi, in cui si prevede un caso per ogni possibile comando in ingresso

- questo approccio non e' usabile quando e' difficile prevedere i comandi di ingresso

SQL Dinamico

- i comandi principali alla base di SQL dinamico sono:

  DECLARE STATEMENT
  PREPARE
  EXECUTE
  EXECUTE IMMEDIATE

- EXEC SQL DECLARE *Nome* STATEMENT;
  dove *Nome* e' un nome (assegnato dall'utente) di comando;
  tale nome e' usato dagli altri comandi di SQL dinamico

- EXEC SQL PREPARE *Nome* FROM *Expr*;
  dove:

  - *Nome* e' un nome di comando; deve essere stato dichiarato tramite il comando DECLARE STATEMENT
  - *Expr* e' un'espressione di tipo stringa che denota il comando SQL da eseguire; molto spesso tale espressione e' un nome di variabile di tipo stringa

  il comando PREPARE ha l'effetto di compilare il comando SQL denotato dall'espressione *Expr*

SQL Dinamico

- EXEC SQL EXECUTE *Nome* [USING *Args*];
  dove:

  - *Nome* e' un nome di comando che deve essere stato compilato tramite il comando PREPARE
  - *Args* e' una lista di variabili di programma
    i comandi SQL che sono generati dinamicamente non possono contenere variabili di programma; possono tuttavia contenere "parametri" indicati dal simbolo "?"
    (detto "dynamic parameter")
    al momento dell'esecuzione a tali parametri sono sostituiti i valori della variabili in *Args*

  l'effetto dell'istruzione EXECUTE e' di eseguire il comando denotato da *Nome*

- EXEC SQL EXECUTE IMMEDIATE *Nome* ;

  combina le funzioni dei comandi PREPARE ed EXECUTE

  conviene usare queste due nel caso in cui uno stesso comando si debba eseguire piu' volte

  inoltre il comando EXECUTE IMMEDIATE non ha la clausola USING

SQL Dinamico - Esempi

```
main ( )
{
      EXEC SQL BEGIN DECLARE SECTION;
            ...........
            VARCHAR sql_source[256];
            ...........
      EXEC SQL END DECLARE SECTION;

      EXEC SQL DECLARE sqolobj STATEMENT;
            ............
      strcpy(sqlsource, "DELETE FROM Invii WHERE quantita' <100");
      EXEC SQL PREPARE sqlobj FROM :sqlsource;
      EXEC SQL EXECUTE  sqlobj;
}
```

```
main ( )
{
      EXEC SQL BEGIN DECLARE SECTION;
            ...........
            VARCHAR sql_source[256];
            short minimo, massimo;
            ...........
      EXEC SQL END DECLARE SECTION;

      EXEC SQL DECLARE sqolobj STATEMENT;
            ............
      strcpy(sqlsource, "DELETE FROM Invii
                  WHERE quantita >? AND quantita <?");
      EXEC SQL PREPARE sqlobj FROM :sqlsource;
      EXEC SQL EXECUTE  sqlobj USING :minimo, :massimo;
}
```


SQL Dinamico - SQLDA

- quanto visto finora va bene per la preparazione e l'esecuzione di tutti i comandi di SQL, eccetto il SELECT
- il motivo e' che SELECT restituisce delle tuple, mentre tutti gli altri comandi restituiscono solo informazioni di feedback (nella SQLCA)
- in particolare un programma che usa SELECT ha bisogno di sapere che tipo di valori sono restituiti dall'esecuzione dell'interrogazione
- viene pertanto fornito un ulteriore comando, chiamato DESCRIPTOR che fornisce informazioni sul tipo di dati restituiti da una SELECT

  EXEC SQL DESCRIBE *Nome* INTO :descriptor;

  dove:

  Nome e' nome del comando di cui vogliamo descrivere il risultato

  descriptor deve essere un puntatore ad una struttura dati il cui formato e' dato SQLDA (descriptor area)

- campi contenuti nella SQLDA
  SQLN: N. max di colonne (intero)
  SQLD: N. effettivo di colonne (intero)
  SQLVAR: un array in cui c'e' un elemento per ogni colonna restituita dalla query;
  ogni elemento ha la seguente struttura
  - SQLTYPE: tipo della colonna (intero)
  - SQLLEN: lunghezza (intero)
  - SQLDATA: indirizzo dove restituire il valore della colonna (puntatore)
  - SQLIND: indirizzo dove restituire il valore della variabile indicatore (puntatore)
  - SQLNAME: nome della colonna (stringa)

SQL Dinamico - SQLDA

Passi nell'esecuzione di un programma:

1) si costruisce e si "PREPARA" il comando di SELECT (che non puo' contenere la clausola INTO)

2) si usa DESCRIBE per interrogare il sistema riguardo ai risultati; la descrizione di tali risultati e' restituita in un'apposita area chiamata SQL Descriptor Area

3) si alloca un'area di memoria per un insieme di variabili di programma a cui saranno assegnati i risultati
l'indirizzo di tale area deve essere memorizzato dal programma nella SQLDA

4) si ritrovano le tuple una alla volta usando il cursore

SQL Dinamico - Schema di programma

```
#include <stdio.h>
#include <ctype.h>
EXEC SQL INCLUDE SQLCA;                    /* inclusione della communication area */
EXEC SQL INCLUDE SQLDA;                    /* inclusione della description area */
main ( )
{
  EXEC SQL BEGIN DECLARE SECTION;          /* dichiarazione variabili ospiti */
      VARCHAR qstring[256];                /* buffer per la query */
      short volume, incr;
      VARCHAR nome_utente[20], pass_utente[10]
  EXEC SQL END DECLARE SECTION;

  EXEC SQL DECLARE sqlobj STATEMENT;
  EXEC SQL DECLARE Y CURSOR FOR sqlobj;

  struct sqlda *sqldaptr;                  /* puntatore al descrittore */

  sqldaptr = (struct sqlda*) malloc (SQLDASIZE(N));
                               /* alloca un'area che contenga al piu' N colonne */
  sqldaptr ->sqln = N;

  strcpy (qstring, "SELECT * FROM Invii WHERE quantita >100;");
                                           /* genera la query */

  EXEC SQL PREPARE sqlobj FROM :qstring;
  EXEC SQL DESCRIBE sqlobj INTO :*sqldaptr;   /* descrive la query */

      /* a questo punto SQLDA contiene (tra le altre informazioni):
              - il numero di colonne restituite dalla query (campo SQLD)
              - tipo e lunghezza della i-sima colonna (campo SQLVAR[i])

      usando tali informazioni si alloca area di memoria per ogni componente;
      l'indirizzo e' memorizzato in SQLVAR[i]   */

  EXEC SQL OPEN X;
      while (SQLCODE ==0)
              { EXEC SQL FETCH X USING DESCRIPTOR :*sqldaptr;
                     ....
              }
  EXEC SQL CLOSE X;
}
```

SQL Dinamico - Schema di programma

Note:

- l'istruzione EXEC SQL INCLUDE SQLDA ha come effetto l'inclusione nel programma di quanto segue:

1) definizione della struttura SQLDA (typedef)

2) definizione della struttura SQLVAR (che definisce il formato dell'entrata allocata ad ogni singola colonna)

3) macro SQLDASIZE(n) che calcola la lunghezza in bytes della struttura SQLDA con n entrate

- in alcuni sistemi non e' necessario usare il comando DECLARE STATEMENT

SQL Cataloghi - cenni

- il catalogo e' un database di sistema che contiene informazioni (descrittori) riguardanti i vari oggetti che sono di interesse al sistema

  tabelle, views, indici, diritti di accesso

- in un sistema come DB2 i cataloghi stessi sono organizzati in relazioni

- esempi:

  SYSTABLES contiene una tupla per ogni relazione o view

  data una relazione alcune delle informazioni contenute in una tupla del catalogo sono:
  il nome (TABNAME)
  il nome dell'utente creatore (DEFINER)
  il tipo (TYPE, T=table, V=view, A=alias)
  il numero di colonne (COLCOUNT)

  SYSCOLUMNS contiene una tupla per ogni colonna di ogni relazione o view nell'intero sistema

  data una colonna alcune delle informazioni contenute in una tupla del catalogo sono:
  il nome della colonna (COLNAME)
  il nome della tabella o view a cui la colonna appartiene (TABNAME)
  la posizione ordinale della colonna tra le colonne della stessa tabella o view (COLNO)
  il tipo della colonna (TYPENAME)

- nota: sull'insieme delle tabella che costituiscono i cataloghi sono spesso definite da parte del sistema viste diverse, quindi i cataloghi possono avere nomi diversi

SQL Cataloghi - cenni

- e' possibile interrogare i cataloghi come se fossero delle relazioni ordinarie

esempio: trovare tutte le tabelle che hanno una colonna che comincia con S

```
SELECT TABNAME
  FROM SYSTEM.SYSCOLUMNS WHERE
  COLNAME LIKE 'S%';
```

esempio: trovare tutte le colonne della relazione Impiegati;

```
SELECT COLNAME
  FROM SYSTEM.SYSCOLUMNS WHERE
       TABNAME = 'Impiegati';
```

esempio: determinare quante relazioni sono state create da Rossi

```
SELECT COUNT(*)
  FROM SYSTEM.SYSTABLES WHERE
       DEFINER = 'Rossi';
```

- in genere non e' possibile modificare i cataloghi (alcune eccezioni sono rappresentate dalla possibilita' per gli amministratori della base di dati di poter modificare direttamente informazioni sulle statistiche usate per l'ottimizzazione)

i cataloghi sono modificati automaticamente dal sistema a seguito dei comandi di DDL
CREATE TABLE
CREATE VIEW

SQL Cataloghi - cenni

- i cataloghi includono anche informazioni sulle relazioni stesse che li memorizzano

- ad esempio il catalogo SYSTABLES avra' un'entrata relativa a se' stesso

(SYSTABLES, SYSTEM, 20, T,...)

- le entrate relative alle relazioni di catalogo non sono create usando i comandi del DDL
tali entrate sono create automaticamente come parte della procedura di installazione del DBMS
(sono "hard-wired" nel sistema)