formato FLOAT (p)

NUMERIC rappresenta i valor
e' caratterizzato da una precisione e u
(numero di cifre della parte frazionari
la specifica di questo tipo di dato ha l
NUMERIC (p, s)

DECIMAL e' simile al tipo NU
la specifica di questo tipo di dato ha l
DECIMAL (p, s)
La differenza tra NUMERIC e DECI
primo deve essere implementato esatt
precisione richiesta, mentre il secondo
una precisione maggiore

SQL - Tipi di Dato

- i tipi di dato in SQL:1999 si suddividono in
  - tipi predefiniti
  - tipi strutturati
  - tipi user-defined

  ci concentreremo sui tipi predefiniti (i tipi strutturati e user-defined verranno considerati nelle caratteristiche object-relational di SQL:1999)

- i tipi di dato predefiniti sono suddivisi in 5 categorie: tipi numerici, tipi binari, tipi carattere, tipi temporali e tipi booleani

- i vari tipi sono suddivisi in sottocategorie

Tipi numerici

- tipi numerici *esatti*
  rappresentano valori interi e valori decimali in virgola fissa (es. 75, -6.2)
- tipi numerici *approssimati*
  rappresentano valori numerici approssimati con rappresentazione in virgola mobile (es. 1256E-4)

Linguaggio SQL

- il linguaggio SQL *(Structured Query* linguaggio per la definizione e la manip sviluppato originariamente presso il labo Jose (Calif.)

- e' diventato standard ufficiale nel 1986 (

- ultimo standard e' SQL:1999 (noto anc con caratteristiche object-relational precedente e' SQL2 (o SQL-92)

- e' il linguaggio oggi piu' usato nei DBMS prodotti commerciali

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

CHAR(n)	stringa di lunghezza n, dove n ≤ 254 (default n=1)
VARCHAR(n)	stringa di lunghezza variabile con lunghezza massima pari ad n, dove n ≤ 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

Tipi booleani

BOOLEAN rappresenta i valori bool
i valori di tale tipo sono
TRUE, FALSE, UNKNOWN

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

CHARACTER LARGE OBJECT (CLOB)
questo tipo di dato permette di definire sequenze di caratteri di elevate dimensioni (testo)
verra' discusso in dettaglio relativamente alle caratteristiche object-relational di SQL:1999

e' possibile associare al tipo carattere il CHARACTER SET di riferimento e la relativa COLLATION (ordine dei caratteri nel set)
per ognuno dei tipi carattere esiste inoltre la variante NATIONAL

SQL - Tipi di Dato
Tipi binari

BIT rappresenta stringhe di bit di lung
predefinita
la specifica di questo tipo di dato ha l
BIT(n) dove n e' la lunghezza massim
stringhe (se non viene specificata alcu
il default e' 1)

BIT VARYING rappresenta strin
lunghezza massima predefinita
la specifica di questo tipo di dato ha l
BIT VARYING(n) dove n e' la lunghe
delle stringhe

la differenza con il tipo BIT e' che pe
si alloca per ogni stringa la lunghezza
predefinita, mentre per BIT VARYIN
strategie diverse

per entrambi i tipi possono essere util
rappresentazioni binaria o esadecimal
(es. B'01111' o X'44')

BINARY LARGE OBJECT (BLO
di dato permette di definire sequenze
dimensioni
verra' discusso in dettaglio relativame
caratteristiche object-relational di SQ

fa seguire la specifica dell'attributo da UNIQUE o PRIMARY KEY

- alternativamente si puo' far seguire la definizione della tabella da

  PRIMARY KEY (column-name-list)

  o

  UNIQUE (column-name-list)

- per i vincoli differibili si puo' spe

time iniziale: INITIALLY DEFER
INITIALLY IMMEDIATE

- i vincoli non possono contenere

valutazione puo' dare risultati differ
quando e' valutata (es. riferimenti al t

DEFERRABLE) o se possa essere ri
della transazione (DEFERRABLE)

SQL - Domini

- e' possibile definire domini ed utilizzarli nella definizione di tabelle
- un dominio e' un nuovo nome per un tipo di dato, a cui si possono associare altre informazioni (valori di default e/o vincoli sui valori)
- sintassi:

```
CREATE DOMAIN nome-dominio
 AS descrizione-tipo
 [DEFAULT valoredefault];
```

- Esempio:

```
CREATE DOMAIN DominioMansione
AS CHAR(10)
DEFAULT 'impiegato';

CREATE TABLE Impiegati
      (Imp#        Decimal(4),
       Nome        Char(20),
       Mansione    DominioMansione,
       Data_A      Date,
       Stipendio   Decimal(7,2),
       Premio_P    Decimal(7,2) DEFAULT 0,
       Dip#        Decimal(2));
```

SQL – DDL: Creazione di t

- La creazione avviene tramite il comando **CREATE TABLE**
- sintassi:

```
CREATE TABLE nome-tabella
   (spec_col1
   [,  ....,
   spec_colN]);
```

dove:

- nome-tabella e' il nome della relazione creata
- $spec_col_i$ (i=1,....,n) e' una specifica il cui formato e' il seguente

  $NomeColonna_i$ $Dominio_i$
  [DEFAULT valoredefault]

- Esempio:

```
CREATE TABLE Impiegati
      (Imp#        Decimal(4),
       Nome        Char(20),
       Mansione    Char(10),
       Data_A      Date,
       Stipendio   Decimal(7,2),
       Premio_P    Decimal(7,2) DEFAUL
       Dip#        Decimal(2));
```

(b) nessuna di tali colonne e' NULL ed esiste una tupla nella tabella riferita la cui chiave coincide con i valori di tali colonne

• MATCH PARTIAL: il vincolo di integrita' referenziale e' soddisfatto se per ogni tupla della tabella referente i valori delle colonne non nulle della chiave esterna corrispondono ai valori di chiave di una tupla della tabella riferita

il default e' MATCH SIMPLE

primaria) della tabella riferita
non e' necessario che i nomi siano gli st
degli attributi corrispondenti devono essere

• nel caso di chiave esterna costituita da

si puo' far seguire la specifica d
REFERENCES NomeTabellaRiferita

SQL - NOT NULL

- per specificare che un colonna non puo' assumere valori nulli e' sufficiente includere il vincolo NOT NULL nella specifica della colonna
- le colonne dichiarate PRIMARY KEY non possono assumere valori nulli
- Esempio:

```
CREATE TABLE Impiegati
      (Imp#           Decimal(4) PRIMARY KEY,
       Codice_Fiscale Char(16) UNIQUE,
       Nome           Char(20) NOT NULL,
       Mansione       DominioMansione,
       Data_A         Date,
       Stipendio      Decimal(7,2) NOT NULL,
       Premio_P       Decimal(7,2),
       Dip#           Decimal(2) NOT NULL);
```

SQL - Chiavi

Esempi

```
CREATE TABLE Impiegati
      (Imp#           Decimal(4) PRIMARY
       Nome           Char(20),
       Mansione       DominioMansione,
       Data_A         Date,
       Stipendio      Decimal(7,2),
       Premio_P       Decimal(7,2),
       Dip#           Decimal(2));
```

```
CREATE TABLE Impiegati
      (Imp#           Decimal(4) PRIM
       Codice_Fiscale Char(16) UNIQUE
       Nome           Char(20),
       Mansione       DominioMansione
       Data_A         Date,
       Stipendio      Decimal(7,2),
       Premio_P       Decimal(7,2),
       Dip#           Decimal(2));
```

```
CREATE TABLE Film
      (Titolo   Char(20),
       Anno     Integer,
       Studio   Char(20),
       Colore   Boolean,
       PRIMARY KEY (Titolo,Anno));
```

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

- Esempio di constraint che puo' essere violato

CHECK (Stipendio < (SELECT MAX(Stipendio)
FROM Impiegato
WHERE Mansione = 'dirigente'))

- i vincoli su domini sono del tutto analoghi – esempio:

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

Sno Char(6),
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 - Chiavi esterne

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

- differenza
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
- ordine in cui vengono considerate le varie opzioni (nel caso di piu' riferimenti):
  - RESTRICT
  - CASCADE, SET NULL, SET DEFAULT
  - NO ACTION
- nel caso di inserimento o modifica nella tabella referente non e' possibile specificare alcuna opzione e quella applicata e' sempre NO ACTION

SQL - Chiavi estern

Opzioni riguardanti le azioni da esegui
cancellazione di una tupla riferita tramite c

(1) NO ACTION: la cancellazione
dalla tabella riferita e' eseguita solo se n
tupla nella tabella referente che ha come
la chiave della tupla da cancellare

(2) RESTRICT: come per NO AC
differenza che questa condizione vi
subito, mentre NO ACTION viene co
che sono state esaminate tutte le a
relative all'integrita' referenziale

(3) CASCADE: la cancellazione di
tabella riferita implica la cancellazione
della tabella referente che hanno come c
chiave della tupla da cancellare

(4) SET NULL: la cancellazione di
tabella riferita implica che in tutte le tu
referente che hanno come chiave esterna
tupla da cancellare, la chiave esterna
valore NULL (se ammesso)

(5) SET DEFAULT: la cancellazion
dalla tabella riferita implica che in tutt
tabella referente che hanno come ch
chiave della tupla da cancellare, il valo
viene posto uguale al valore di default
le colonne che costituiscono la chiave es

- 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

- ogni vincolo ha associato un descritto
  - nome (se non specificato asseg
  - differibile o meno
  - checking time iniziale

- per modificare il checking time (so DEFERRABLE) si usa il comando

  SET CONSTRAINTS {listaNomiCon
  {IMMEDIATE|DEFERRED}

SQL – Asserzioni

- sono elementi dello schema, servono per scrivere vincoli che coinvolgono piu' tuple o piu' tabelle

- sintassi:

```
CREATE ASSERTION Nome
CHECK (Condizione)
```

la condizione, che si valuta in vero o falso, e' un predicato o una combinazione booleana di predicati (componente WHERE di una query SQL)

- Esempio:

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

- Non tutti i DBMS supportano tutti i tipi di vincoli, in particolare le asserzioni non sono generalmente supporate ed i DBMS si limitano a gestire vincoli che possono essere verificati esaminando una sola tupla (motivazione: efficienza della valutazione)

SQL – Vincoli CHECK su

- quando si definisce una tabella e' po alla specifica delle colonne, delle chia esterne la parola chiave CHECK condizione, cioe' un predicato o u booleana di predicati (componente WH SQL)

- tale condizione puo' contenere sottoi fanno riferimento ad altre tabelle, ma controllato solo quando viene inserita tabella o viene modificata una tupla della

- Esempio:

CHECK (Stipendio > PremioP)

- Esempio di constraint che puo' essere

CHECK (Stipendio < (SELECT MAX(S
FROM Impiegato
WHERE Mansion

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

relazione R

- se una sola relazione nella lista di relaz FROM ha una colonna di nome C, si pu di R.C
- F e' un predicato analogo ai predica dell'operazione relazionale σ
- il significato di una query SQL puo' es mezzo della seguente espressione algebr

$\pi_{Ri1.C1,\ Ri2.C2,\ .....,\ Rin.Cn}\ (\sigma_{F(R1}$

SQL - DDL
Cancellazione e modifica di domini e vincoli

- ALTER DOMAIN D SET DEFAULT valore_default
  modifica il valore di default di un dominio
- DROP DOMAIN D{RESTRICT|CASCADE}
  rimuove un dominio
  - se viene specificato RESTRICT: il dominio viene rimosso solo se nessuna tabella lo utilizza
  - se viene specificato CASCADE: in ogni tabella che lo utilizza il nome del dominio viene sostituito dalla sua definizione (la modifica non influenza i dati presenti nella tabella)
- Modifica di vincoli

ALTER TABLE R DROP CONSTRAINT C
ALTER DOMAIN D DROP CONSTRAINT C
rimuove vincolo C da tabella R o dominio D

ALTER TABLE R ADD CONSTRAINT C …
ALTER DOMAIN D ADD CONSTRAINT C …
aggiunge vincolo C a tabella R o dominio D

DROP ASSERTION A
rimuove asserzione A

SQL - DDL
Cancellazione e modifiche a schem

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

  esempio: DROP TABLE Im
- RENAME R_V TO R_n
  dove R_V e R_n sono, rispettivame nuovo nome della relazione

  esempio: RENAME Impieg
- ALTER TABLE R ADD COLUMN s
  aggiunge una nuova colonna ad u

  esempio:
  ALTER TABLE Impiegati
  ADD COLUMN (Prog# D
- ALTER TABLE R ALTER COLUM
  modifica una colonna di una rela

  esempio:
  ALTER TABLE Impiegati
  ALTER COLUMN (Prog#
- ALTER TABLE R DROP COLUMN
  rimuove una colonna da una rela

  esempio:
  ALTER TABLE Impiegati
  DROP COLUMN

• 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

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

risultato

Dip#	Nome_Dip	Ufficio	Divisio
10	Edilizia Civile	1100	D1
30	Edilizia Stradale	5100	D2

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 v1 AND v2

forma negata

C NOT BETWEEN v1 AND v2

• esempio

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

risultato

Nome	Stipendio
Adami	1100,00
Milli	1300,00

SQL - Interrogazioni
esempi dalla base di dati impiegati

Q2: selezionare il nome e il numero di c impiegati che hanno uno stipendio ma hanno mansione di ingegnere

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

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

risultato Q2

Nome	Dip#
Neri	10
Dare	10

Q3: selezionare il numero degli impiegati dipartimento 30 e sono ingegneri o tec

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

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

risultato Q3

Imp#
7499
7521
7844
7900

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

ingegnere
ingegnere
segretaria
ingegnere
dirigente

- e' possibile richiedere l'eliminazione de la clausola DISTINCT

SELECT DISTINCT Mansione FROM

risultato

Mansione
ingegnere
tecnico
dirigente
segretaria

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
ordinamento del risultato di u

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

- e' possibile specificare un ordin aggiungendo alla fine dell'interrogaz ORDER BY

- esempio: elencare lo stipendio, la mans tutti gli impiegati del dipartimento tuple in ordine crescente in base al

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

risultato - le tuple sono ordinate in ordine

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

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

- una espressione aritmetica usata n proiezione di un'interrogazione da' luog detta *virtuale*, non presente nella rel effettua l'interrogazione

- le colonne virtuali non sono fisicame sono solo *materializzate* come interrogazioni

- nel calcolo delle espressioni aritmetich viene considerato uguale al valore zero

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 - Operazione di jo

- il risultato di un'operazione di join e' pertanto possibile richiedere che tale rela anche in base a valori di colonne di relaz le tuple duplicate siano eliminate

- si vuole ritrovare per ogni impiega stipendio, la mansione, e il nome del d l'impiegato lavora. Inoltre si vuole or ordine crescente in base al nome del di ordine decrescente in base allo stipendio

```
SELECT Nome_Dip, Nome, Mansione,
        FROM Impiegati, Dipartimenti
        WHERE Impiegati.Dip# = Dipar
        ORDER BY Nome_Dip, Stipend
```

risultato

Nome_Dip	Nome	Mansione
Edilizia Civile	Verdi	dirigente
Edilizia Civile	Neri	ingegnere
Edilizia Civile	Dare	ingegnere
Edilizia Civile	Milli	ingegnere
Edilizia Stradale	Blacchi	dirigente
Edilizia Stradale	Gianni	ingegnere
Edilizia Stradale	Turni	tecnico
Edilizia Stradale	Andrei	tecnico
Edilizia Stradale	Bianchi	tecnico
Edilizia Stradale	Martini	segretaria
Ricerche	Rosi	dirigente
Ricerche	Rossi	ingegnere
Ricerche	Adami	ingegnere
Ricerche	Fordi	segretaria
Ricerche	Scotti	segretaria

- 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(*)

Nome	Data_A	CURRENT_DATE	D
Rossi	23/01/82	04/03/82	25

SQL - Date e tempi

- funzioni per conversione tra date/tempi ed altri tipi di dato
  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)

- funzioni per la determinazione del valore di registri speciali
  - 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 - Espressioni e Funzioni pe

- espressioni di tipo stringa possono esse
  l'operatore di concatenazione denotato d

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

  restituisce un'unica stringa che contiene
  ed indirizzo separati da uno spazio bianc

- 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 sottostr
    di posizione m per una lunghezza
    specificato) oppure fino all'ultim

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

- 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 grupp

- esempio: si vuole raggruppare gli imp
  numero di dipartimento e si vuo
  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 G
  tupla della relazione risultato rappresen
  tuple della relazione su cui la query e' es
- nell'esempio visto i gruppi sono tre: uno
  DIP#
  ad ognuno di questi gruppi e' applicata
  sulla colonna Stipendio

- 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 tupla t2

t3 F AND ? --> F
nessuna tupla verifica la query

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

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 - Funzioni di grupp

Un modello di "esecuzione

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

2.alle tuple ottenute al passo preceden
partizionamento specificato dalla clausol

3.ad ogni gruppo di tuple ottenuto al pas
applica la condizione di ricerca specific
HAVING

4. i gruppi ottenuti al passo precedente
tuple che verificano la query
per tali gruppi, vengono calcolate le fu
specificate nella clausola di proiezione d
i valori restituiti da tali funzioni costitu
della query

Biacchi 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)

FROM Impiegati WHE
Nome = 'Gianni');

- la subquery restituisce come valo
'ingegnere'
- la query esterna determina quindi
che sono ingegneri
- il risultato e'

{Rossi, Neri, Dare, Adami, Giann

SQL - Join e outerjoin

- in R |X| S non si ha traccia delle tuple di R che non corrispondono ad alcuna tupla di S
- questo non sempre e' quello che si desidera
- l'operatore di OUTER JOIN aggiunge al risultato le tuple di R e S che non hanno partecipato al join, completandole con NULL
- l'operatore di join originario, per contrasto, e' anche detto INNER JOIN
- esistono diverse varianti dell'outer join: R OUTER JOIN S
  - FULL: sia le tuple di R che quelle di S che non partecipano al join vengono completate e inserite nel risultato
  - LEFT: le tuple di R che non partecipano al join vengono completate e inserite nel risultato
  - RIGHT: le tuple di S che non partecipano al join vengono completate e inserite nel risultato
- la variante OUTER puo' essere utilizzata sia per join naturale che per theta-join
- esempi:

Impiegati LEFT OUTER JOIN Dipartimenti
ON Imp# = Dirigente

Relatore NATURAL RIGHT OUTER JOIN Studente

- l'ultimo operatore e' l'UNION JOIN, che contiene ogni colonna e ogni riga delle tabelle di partenza, completate con NULL in tutti gli attributi non presenti nella tupla originaria

SQL - Join e outerjoin

- abbiamo visto come e' possibile esprim
interrogazioni SELECT-FROM-WHERE
- in realta' SQL:1999 prevede diversi ti
join
- questi operatori poiche' producono r
essere usati nella clausola FROM
- la forma di operatore join piu' semplice
al prodotto Cartesiano, e' il CROSS JOI

es.: Impegati CROSS JOIN Dipartime

- il theta-join si ottiene come operatore JO

es.: Impiegati JOIN Dipartimenti
ON Impiegati.Nome > Dipa

- il join naturale corrisponde all'operatore

es.: Dipartimenti NATURAL JOIN Ir

- sintassi alternativa: JOIN USING (listaN

es.: Dipartimenti JOIN Impiegati USI

ingegnere	Milli

• esempio: elencare il nome e la mansion
del dipartimento 10 che han
mansione non presente nel dipartin

```
SELECT Nome, Mansione FROM Impi
  WHERE Dip#=10 AND Mansione NO
        (SELECT Mansione FROM
          WHERE Dip#=30);
```

risultato

Mansione	Nome
tecnico	Andrei
tecnico	Bianchi
tecnico	Turni
segretaria	Martini

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

• negli esempi visti, le subqueries resti
valore

• se la subquery restituisce piu' valo
specificare come i valori restituiti dev
nella clausola WHERE

• a tale scopo vengono usati i *quantificato*
che sono inseriti tra l'operatore di confro

• esempio: determinare lo stipendio, la ma
il numero di dipartimento deg
guadagnano piu' di *almeno u*
dipartimento 30

```
SELECT Stipendio, Mansione, Nome, D
  FROM Impiegati WHERE
        Stipendio > ANY (SELECT
                         FROM I
                         WHERE
```

risultato

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

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

- per poter riferire le colonne delle tuple
query esterna si fa uso degli alias di rel
di relazione e' definito nella query es
nella query interna

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 multi

- la clausola di WHERE di una query p
qualsiasi combinazione di condizioni no
con subqueries

- esempio: si vuole elencare gli impieg
mansione di Gianni o con uno stipe
uguale a quello di Fordi. Inoltre si
risultato in base alla mansione e allo stip

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

- una subquery puo' contenere a sua volta

- esempio: elencare il nome e la mansion
del dipartimento 10 con la stessa mansi
impiegato del dipartimento Ricerche

```
SELECT Nome, Mansione FROM Impi
  WHERE Dip#=10 AND Mansione IN
      (SELECT Mansione FROM Imp
          WHERE Dip# =
              (SELECT Dip# FROM
                  WHERE Nome_D
```

```
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');
```

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

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 - Subqueries correla ALL ed ANY

- esempio: determinare i dipartimenti ta impiegati di tali dipartimenti gua 1000

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

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

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

nota: tale interrogazione si puo' esprimere con un join

non solo in congedo

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

```
SELECT Nome FROM Impiegati_Temp
   EXCEPT
SELECT Nome FROM Impiegati_In_Co
```

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 insiem
Union

- l'operatore UNION impone alcune interrogazioni si cui opera
- le interrogazioni devono restituire lo colonne, e le colonne corrispondenti stesso dominio (non e' richiesto che lunghezza) o domini compatibili
- se si usa una clausola di ORDER BY c usata una sola alla fine dell'interrogazio *di ogni* SELECT
- quando si specificano le colonne l'ordinamento non si possono usare i poiche' questi potrebbero essere diffe relazioni
  occorre indicarle specificandone la *p* all'interno della clausola di proiezione
- esempio:

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

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

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 Inserzio

il comando di Insert ha il seguente formato

INSERT INTO R [(C_1,C_2,....,C_n)]
{VALUES (e_1,e_2,....,e_n) | sq};

dove:

- R e' il nome della relazione su cui si e
- C_1,C_2,....,C_n e' la lista delle colonne (o delle nuove tuple) a cui si assegnan

  tutte le colonne non esplicitamente el valore nullo o il valore di default (comando di creazione di R)

  la mancata specifica di una lista di co una lista che include tutte le colonne d

- e_1,e_2,....,e_n e' la lista di valori da ass tupla

  i valori sono assegnati in base ad u posizionale; il valore e_i (i=1,...,n) colonna C_i

- sq e' una subquery (mutuamente escl clausola VALUES)

  le tuple generate come risposta alla s inserite nella relazione R

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

  pertanto il dominio della colonna C essere compatibile con il dominio espressiione) i-esima contenuta n proiezione di sq

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';
```

```
UPDATE Impiegati
  SET Stipendio =
      (SELECT 1.1*AVG(Stipend
          FROM Impiegati
          WHERE Mansione = 't
WHERE Mansione = 'tecnico';
```

SQL - DML Modifica

il comando di UPDATE ha il seguente for

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

dove:

- R e' il nome della relazione su cui si e
  il nome della relazione puo' avere ass
  e' necessario riferire a tuple di tale
  qualche sottointerrogazione presente i
- C_i = {e_i | NULL} (i=1,...,n) e'
  assegnamento che specifica che alla
  essere assegnato il valore dell'espressi
  tale espressione puo' essere una
  un'espressione aritmetica o di stringa
  dei valori correnti delle tuple da modi
  subquery
  alternativamente si puo' specificare ch
  assegnato il valore nullo
- F e' la clausola di qualificazione che
  da modificare

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

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

- il meccanismo delle views e' utile per

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

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 - DML esempio compl

- supponiamo di creare una relazione seguente comando

```
CREATE TABLE Progetti (Prog# Decima
                       Pnome Varch
                       Budget Decin
```

- supponiamo di inserire alcune tuple Progetti

```
INSERT INTO Progetti VALUES (101,
INSERT INTO Progetti VALUES (102,
INSERT INTO Progetti VALUES (103,
```

- poiche' ogni impiegato e' assegnato a vuole estendere la relazione Impiegat colonna che indichi il numero del proge

```
ALTER TABLE Impiegati
      ADD COLUMN (Prog# Decimal
```

- supponiamo di assegnare tutti i tecn dipartimento) e tutti gli impiegati del d progetto 101

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

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

se si esegue l'interrogazione

SELECT * FROM V1 WHERE Dip#

si ottiene il seguente risultato

Nome	Stipendio_Mensile	Sipendio_Annua
Andrei	800,00	9600,00
Bianchi	800,00	9600,00
Martini	800,00	9600,00
Blacchi	2850,00	34200,00
Turni	1500,00	18000,00
Gianni	1950,00	23400,00

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

- esempio: si vuole creare una vista
  sottoinsieme delle tuple della relazi
  piu' precisamente la vista deve elen
  Imp#, Nome e Mansione
  degli impiegati del dipartimento 10

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

- i nomi delle colonne della vist
  rispettivamente: Imp#, Nome, Mansion

- su una vista si possono eseguire (con
  restrizioni) sia interrogazioni che modifi

- esempio: selezionare le tuple della vista

  SELECT * FROM Imp10;

  risultato:

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

- 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

- se View1 e' definita con una **check opt**
inserzioni eseguite su View1 devono ver
(1) la definizione di View1
(2) la definizione di View2 (indipender
che View2 sia definita con check option

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]
```

- inserzioni eseguite su View1 devono ver
(1) la definizione di View1
(2) la definizione di View2 solo se Vie
definita con la check option
(se View2 non e' definita con la check
definizione di View2 non deve essere
tupla inserita)

SQL - Uso di viste

- una volta creata, una vista puo' essere c
una relazione di base

- e' possibile eseguire qualsiasi interrogaz
con una importante restrizione:

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

- quando si specifica una interrogazione
sistema sostituisce nell'interrogazione la
definizione

tale operazione e' detta *view composition*

esempio: si consideri la vista Personale e

```
SELECT Nome, Pnome FROM I
    WHERE Mansione = 'dirige
```

la query che si ottiene dopo la composizi

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

- una vista puo' essere definita su una o
viste

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

- a causa dell'ambiguita' del mapping, un DBMS e' che non vengono ammesse m definite tramite joins

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

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 side-effects 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 vis

Problema 2: cancellazione

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

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

- supponiamo di eseguire la seguente canc

  DELETE FROM V4 WHERE Nome

- questa cancellazione potrebbe essere trac

  DELETE FROM Impiegati WHERE
  DELETE FROM Dipartimenti
  WHERE Ufficio = 210

(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)

cancella la vecchia definizione della v
nuova

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

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 - Cancellazione e renamin

- formato comando di cancellazione

DROP VIEW V;

dove V e' il nome della vista da cancella

- formato comandi di renaming

RENAME V_V to V_n

- nota: non e' possibile, invece, modificar
l'unico modo e' ridefinire l'interrogazi

- (SELECT che restituisce al piu' una tu
- UPDATE (eccetto nella forma CURR
- DELETE (eccetto nella forma CURR
- INSERT
- tutti i comandi di DDL
- tutti i comandi di autorizzazione

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 pr

concetti di base

1) ogni istruzione di SQL ospitato deve dalla stringa EXEC SQL; questa str precompilatore di distinguere un'istru istruzioni del linguaggio ospite

2) ogni istruzione di SQL deve terminare speciale che dipende dallo specifico ling (esempi: C ;
PL/I ;
Cobol END-EXEC)

3) un'istruzione SQL eseguibile puo' co possa comparire uno statement del lingu diversamente da SQL interattivo, l'SQL istruzioni dichiarative (ad es. DECLARE

4) le variabili del inguaggio ospite posso istruzioni SQL in accordo a quanto segu

- nella clausola INTO del comando di S (in questo caso le variabili conterran query)
- nella clausola WHERE dei coman UPDATE, DELETE
  (in questo caso le variabili conten confrontare con gli attributi delle tuple
- nella clausola SET del comando di UP (in questo caso le variabili conten assegnare alle tuple modificate)
- nella clausola VALUES del comando (in questo caso le variabili conten assegnare alle nuove tuple)
- nelle espressioni aritmetiche, di stringa

- 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*/
```

- nelle ultime due situazioni le variabi
  :volume, e :citta non vengono modificat
  programma e' considerato in errore e S
  valore negativo

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 - Accesso da linguaggio di pr

esempio

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

EXEC SQL INCLUDE SQLCA; /* inclusione della co
int prompt1(char[~~], char[~~], int);
char prompt[  ] = "IMMETTERE NUMERO FORNIT

main ( )
{
      EXEC SQL BEGIN DECLARE SECTION
                                    /* dichiarazione v
            VARCHAR dato_f[5], citta[15];
            short volume;
            VARCHAR nome_utente[20], pass_utente[
      EXEC SQL END DECLARE SECTION;
                              /* inizializza nome e pa
                              per la connessione alla
      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 IDENTIF
      while((prompt1(prompt, dato_f.arr, 5)) > = 0)
      {
            dato_f.len = strlen(dato_f.arr);
            EXEC SQL SELECT Volume, Citta FRO
                  INTO :volume, :citta
                  WHERE Forn#=:dato_f;
            printf("Il volume del fornitore e' %d e la
                  volume, citta_arr)};
      }
      EXEC SQL COMMIT RELEASE;
}
```

```
    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;
}
```

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

  EXEC SQL FETCH NomeC INTO Lista

  la variabili possono avere associat indicatori per la segnalazione dei valori

- un cursore e' disattivato tramite il coman

  EXEC SQL CLOSE NomeC;

  uno stesso cursore puo' essere attivato (o piu' volte) durante l'esecuzione programma, dopo essere stato pero' disat

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 senza cu

- UPDATE: supponiamo di incrementar forniture dei fornitori di Londra d contenuto nella variabile di programma

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

  le variabili indicatore possono essere che ad una certa colonna deve essere as nullo

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

  se prima dell'esecuzione del coman :cittaind riceve valore negativo, allora viene assegnato valore nullo,
  se :cittaind riceve 0, allora alla col assegnato il valore contenuto nella varia :citta

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

```
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 :forn#, :nomef, :volu
            save = save -1;
            strcpy(disp, "inalterato");
            if (strcmp(cittaf, citta_data))
                    { EXEC SQL UPDATE Fornitori
                            SET Volume = Volume + :incr WH
                      strcpy(disp, "aggiornato");}
            else
            { if (volume < liv_volume)
                    { EXEC SQL DELETE FROM Fornitori W
                      EXEC SQL DELETE FROM Invii WHE
                      strcpy(disp, "cancellato");
                    }
            }
print ("\n, \%s, \%s, \%d, \%s, \%s", forn#, nomef, volume, c
}
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
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 cur

- e' possibile avere un riferimento ad comandi di UPDATE e DELETE

- in particolare se un cursore X e' po determinata tupla e' possibili modificare "la tupla corrente di X"

- i comandi di UPDATE e DELETE con cursore hanno il seguente formato:

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

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

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

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

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*

- se l'insieme dei comandi che possono terminale e' piccolo, si puo' "cablare applicativo l'insieme di tutti i poss ingresso
  il programma applicativo sara' organizza a casi, in cui si prevede un caso p comando in ingresso

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

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

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

- nota: il pre-compilatore non analizz programma
  quindi l'overriding e' basato sulla "ordin comandi come linee di codice nel p sull'ordine effettivo di esecuzione

- esempio:

```
main ( )
{
        EXEC SQL WHENEVER SQLERROR S
                                /* primo comando WH
        .....
        goto s1;
        .....
        EXEC SQL WHENEVER SQLERROR C
                        /* sovrascrive il primo coman

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

quando si esegue il comando UPDATE, eseguita a fronte di errori e' CONTIN comando di UPDATE e' stato raggiunt nello "scope" del primo comando di WH

Nome e' nome del comando di cui vog
il risultato

descriptor deve essere un puntatore ac
il cui formato e' dato SQLDA (descrip

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

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

- EXEC SQL EXECUTE *Nome* [USING
  dove:

  - *Nome* e' un nome di comando che
    compilato tramite il comando PREPAR

  - *Args* e' una lista di variabili di prograr
    i comandi SQL che sono generati di
    possono contenere variabili di prog
    tuttavia contenere "parametri" indicati
    (detto "dynamic parameter")
    al momento dell'esecuzione a tali
    sostituiti i valori della variabili in *Args*

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

- EXEC SQL EXECUTE IMMEDIATE N

  combina le funzioni dei comandi
  EXECUTE

  conviene usare queste due nel caso i
  comando si debba eseguire piu' volte

  inoltre il comando EXECUTE IMMEI
  clausola USING

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

il numero di colonne (COLCOUNT)

SYSCOLUMNS contiene una tupla per
ogni relazione o view n
data una colonna alcune delle informaz
una tupla del catalogo sono:
il nome della colonna (COLNAME)
il nome della tabella o view a cui la c
appartiene (TABNAME)
la posizione ordinale della colonna tra
della stessa tabella o view (COL
il tipo della colonna (TYPENAME)

- nota: sull'insieme delle tabella che cataloghi sono spesso definite da parte diverse, quindi i cataloghi possono aver

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 Dinamico - Schema di pr

```
#include <stdio.h>
#include <ctype.h>
EXEC SQL INCLUDE SQLCA;                    /* inclusione
EXEC SQL INCLUDE SQLDA;                    /* inclusione
main ( )
{
   EXEC SQL BEGIN DECLARE SECTION;         /* dichiarazio
        VARCHAR qstring[256];              /* buffer per l
        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 a

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

   strcpy (qstring, "SELECT * FROM Invii WHERE quantita >1
                                           /* genera la q

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

        /* a questo punto SQLDA contiene (tra le altre informazion
                - il numero di colonne restituite dalla query (campo
                - tipo e lunghezza della i-sima colonna (campo SQL

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

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

SQL Cataloghi - cenn

- i cataloghi includono anche informazio
  stesse che li memorizzano

- ad esempio il catalogo SYSTABLES
  relativa a se' stesso

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

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