ni SQL:1999 si suddividono in
efiniti
turati
-defined
remo sui tipi predefiniti (i tipi strutturati e
verranno considerati nelle caratteristiche
onal di SQL:1999)

predefiniti sono suddivisi in 5 categorie: tipi
pi binari, tipi carattere, tipi temporali e tipi

no suddivisi in sottocategorie

Tipi numerici

esatti
esentano valori interi e valori decimali in
la fissa (es. 75, -6.2)
approssimati
esentano valori numerici approssimati con
esentazione in virgola mobile (es. 1256E-4)

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

- e' diventato standard ufficiale nel 1986 (SQL-86)

- ultimo standard e' SQL:1999 (noto anche come SQL3), con caratteristiche object-relational, lo standard precedente e' SQL2 (o SQL-92)

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

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

approssimati

rappresenta valori a singola precisione
in virgola mobile
one dipende dalla specifica implementazione

E PRECISION rappresenta valori a
doppia precisione in virgola mobile
one dipende dalla specifica implementazione

permette di richiedere la precisione
che si desidera
LOAT (p)

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

o in CHAR) permette di definire stringhe di
i lunghezza massima predefinita
a di questo tipo di dato ha la forma
dove n e' la lunghezza massima delle
se non viene specificata alcuna lunghezza,
e' 1)

CTER VARYING questo tipo di dato
obreviato in VARCHAR) permette di
ringhe di caratteri di lunghezza massima
a
a di questo tipo di dato ha la forma
AR(n) dove n e' la lunghezza massima delle

nza con il tipo CHAR e' che per questo tipo
er ogni stringa la lunghezza massima
a, mentre per VARCHAR si usano
diverse

CTER LARGE OBJECT (CLOB)
o di dato permette di definire sequenze di
i elevate dimensioni (testo)
usso in dettaglio relativamente alle
iche object-relational di SQL:1999

sociare al tipo carattere il CHARACTER
ento e la relativa COLLATION (ordine dei
t)
ei tipi carattere esiste inoltre la variante
predefinita
la specifica di questo tipo di dato ha la forma
BIT(n) dove n e' la lunghezza massima delle
stringhe (se non viene specificata alcuna lunghezza,
il default e' 1)

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

la differenza con il tipo BIT e' che per questo tipo
si alloca per ogni stringa la lunghezza massima
predefinita, mentre per BIT VARYING si usano
strategie diverse

per entrambi i tipi possono essere utilizzate le
rappresentazioni binaria o esadecimale
(es. B'01111' o X'44')

BINARY LARGE OBJECT (BLOB) questo tipo
di dato permette di definire sequenze di bit di elevate
dimensioni
verra' discusso in dettaglio relativamente alle
caratteristiche object-relational di SQL:1999

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

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 timestamp che consistono di: anno, mese, giorno, ora, minuto, secondo e microsecondo

Tipi booleani

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

o e' un nuovo nome per un tipo di dato, a cui
associare altre informazioni (valori di
vincoli sui valori)

DOMAIN nome-dominio
zione-tipo
LT valoredefault];

MAIN DominioMansione

piegato';

BLE Impiegati
Decimal(4),
Char(20),
e DominioMansione,
Date,
Decimal(7,2),
P Decimal(7,2) DEFAULT 0,
Decimal(2));

CREATE TABLE

- sintassi:

CREATE TABLE nome-tabella
(spec_col$_1$
[, ...,
spec_col$_n$]);

dove:

- nome-tabella e' il nome della relazione che viene creata
- spec_col$_i$ (i=1,....,n) e' una specifica di colonna 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) DEFAULT 0,
       Dip#          Decimal(2));
```

QUE garantisce che non esistano due tuple
ono gli stessi valori non nulli per gli attributi
IQUE possono contenere valori nulli)

MARY KEY impone che per ogni tupla i
li attributi specificati siano non nulli e diversi
di ogni altra tupla

ella e' possibile specificare piu' chiavi
ma una sola PRIMARY KEY

ve e' formata da un solo attributo e' sufficiente
e la specifica dell'attributo da UNIQUE o
Y KEY

amente si puo' far seguire la definizione della

MARY KEY (column-name-list)

QUE (column-name-list)

- un vincolo puo' essere associato a una tabella, a un attributo, ad un dominio

- in SQL e' possibile specificare diversi tipi di vincoli

  - chiavi (UNIQUE e PRIMARY KEY)
  - obbligatorieta' di attributi (NOT NULL)
  - chiavi esterne (FOREIGN KEY)
  - vincoli CHECK (su attributo o su tupla) e asserzioni (vincoli CHECK su piu' tuple o tabelle)

- e' inoltre possibile specificare se il controllo del vincolo debba essere eseguito non appena si esegue un'operazione che ne puo' causare la violazione (NON DEFERRABLE) o se possa essere rimandato alla fine della transazione (DEFERRABLE)

- per i vincoli differibili si puo' specificare un check time iniziale: INITIALLY DEFERRED (default) o INITIALLY IMMEDIATE

- i vincoli non possono contenere condizioni la cui valutazione puo' dare risultati differenti a seconda di quando e' valutata (es. riferimenti al tempo di sistema)

fficiente includere il vincolo NOT NULL
ifica della colonna

e dichiarate PRIMARY KEY non possono
valori nulli

```
BLE Impiegati
            Decimal(4) PRIMARY KEY,
Fiscale     Char(16) UNIQUE,
            Char(20) NOT NULL,
e           DominioMansione,
            Date,
            Decimal(7,2) NOT NULL,
P           Decimal(7,2),
            Decimal(2) NOT NULL);
```

```
CREATE TABLE Impiegati
      (Imp#          Decimal(4) PRIMARY KEY,
       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) PRIMARY KEY,
       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));
```

SIMPLE: il vincolo di integrita' referenziale e' e per ogni tupla della tabella referente (a) delle colonne della chiave esterna e' NULL, essuna di tali colonne e' NULL ed esiste una abella riferita la cui chiave coincide con i colonne

FULL: il vincolo di integrita' referenziale e' e per ogni tupla della tabella referente (a) nne della chiave esterna sono NULL, oppure di tali colonne e' NULL ed esiste una tupla riferita la cui chiave coincide con i valori di

PARTIAL: il vincolo di integrita' referenziale o se per ogni tupla della tabella referente i colonne non nulle della chiave esterna o ai valori di chiave di una tupla della tabella

MATCH SIMPLE

CREATE TABLE

- sintassi:

```
[, FOREIGN KEY (listaNomiColonne)
      REFERENCES NomeTabellaRiferita
   [MATCH {FULL|PARTIAL|SIMPLE} ]
   [ ON DELETE { NO ACTION | RESTRICT |
      CASCADE | SET NULL | SET DEFAULT}]
   [ ON UPDATE { NO ACTION | RESTRICT |
      CASCADE | SET NULL | SET DEFAULT}]

[, FOREIGN KEY.....]])
```

dove listaNomiColonne e' un sottoinsieme delle colonne della tabella che corrisponde ad una chiave (UNIQUE o primaria) della tabella riferita
non e' necessario che i nomi siano gli stessi, ma i domini degli attributi corrispondenti devono essere compatibili

- nel caso di chiave esterna costituita da un solo attributo si puo' far seguire la specifica della colonna da REFERENCES NomeTabellaRiferita

chiave della tupla riferita tramite chiave

stesso significato delle opzioni viste per la
one

ASCADE
ffetto di assegnare alla chiave esterna il
ore della chiave della tupla riferita

NO ACTION sia per la cancellazione che

cui vengono considerate le varie opzioni
di piu' riferimenti):

ESTRICT
ASCADE, SET NULL, SET DEFAULT
O ACTION

i inserimento o modifica nella tabella
non e' possibile specificare alcuna opzione e
olicata e' sempre NO ACTION

cancellazione di una tupla riferita tramite chiave esterna:

(1) NO ACTION: 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

(2) RESTRICT: come per NO ACTION, con la differenza che questa condizione viene controllata subito, mentre NO ACTION viene considerata dopo che sono state esaminate tutte le altre specifiche relative all'integrita' referenziale

(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

ica della colonna viene affiancata la parola
CK seguita da una condizione, cioe' un
una combinazione booleana di predicati
WHERE di una query SQL)

zione puo' contenere sottointerrogazioini che
nento ad altre tabelle, ma il vincolo viene
solo quando viene modificato il valore
a cui e' associato

specifica di range di valori per gli attributi di

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

di constraint che puo' essere violato

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

u domini sono del tutto analoghi – esempio:

```
OMAIN DominioMansione AS Char(10)
ALUE IN
nte', 'ingegnere', 'tecnico', 'segretaria'))
```

```
CREATE TABLE Docente
      (Dno          Char(7),
       Dnome        Varchar(20),
       Residenza    Varchar(15),
       PRIMARY KEY (Dno));

CREATE TABLE Studente
      (Sno          Char(6),
       Sname        Varchar(20),
       Residenza    Varchar(15),
       Birthdate    Date,
       PRIMARY KEY (Sno));


CREATE TABLE Relatore
      (Dno          Char(7),
       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);
```

nenti dello schema, servono per scrivere
oinvolgono piu' tuple o piu' tabelle

SSERTION Nome
ndizione)

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

SSERTION UnSoloDirig
T EXISTS (SELECT * FROM Impiegati
WHERE Mansione = 'dirigente'
GROUP BY Dip#
HAVING COUNT(*) > 1))

i DBMS supportano tutti i tipi di vincoli, in
e asserzioni non sono generalmente supporate
si limitano a gestire vincoli che possono
cati esaminando una sola tupla (motivazione:
lla valutazione)

- quando si definisce una tabella e' possibile aggiungere alla specifica delle colonne, delle chiavi e delle chiavi esterne la parola chiave CHECK seguita da una condizione, cioe' un predicato o una combinazione booleana di predicati (componente WHERE di una query SQL)

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

- Esempio:

CHECK (Stipendio > PremioP)

- Esempio di constraint che puo' essere violato

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

ABLE Esame
Char(6) NOT NULL,
Char(6) NOT NULL,
Varchar (20) NOT NULL,
Integer NOT NULL,
Y KEY (Sno,Cno),
N KEY (Sno) REFERENCES Studente
DELETE CASCADE,
AINT Cname-constr
CK Cname IN ('Documentazione Automatica',
'SEI I', 'SEI II'),
AINT Voto-constr
CK Voto ≥ 18 AND Voto ≤ 30);

constraints:

raint da una tupla e' violato se la condizione
a tupla da valore False

raint la cui valutazione su una data tupla da'
o Unknown (a causa di valori nulli) non e'

l'inserzione o la modifica di un insieme di
azione di un vincolo per una delle tuple causa
zione dell'intero insieme;
a tuttavia continua la sua esecuzione

precedere la specifica del vincolo dalla parola chiave CONSTRAINT e dal nome

- Esempi:

Imp# Char(6) CONSTRAINT Chiave PRIMARY KEY

CONSTRAINT StipOK CHECK(Stipendio > PremioP)

- specificare un nome per i vincoli e' utile per potervisi poi riferire (ad esempio per modificarli)

- ogni vincolo ha associato un descrittore costituito da
  - nome (se non specificato assegnato dal sistema)
  - differibile o meno
  - checking time iniziale

- per modificare il checking time (solo per i vincoli DEFERRABLE) si usa il comando

SET CONSTRAINTS {listaNomiConstraints|ALL}
{IMMEDIATE|DEFERRED}

OMAIN D SET DEFAULT valore_default
fica il valore di default di un dominio

OMAIN D{RESTRICT|CASCADE}
ove un dominio
iene specificato RESTRICT: il dominio viene
rimosso solo se nessuna tabella lo utilizza
iene 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)

vincoli

LE R DROP CONSTRAINT C
AIN D DROP CONSTRAINT C
incolo C da tabella R o dominio D

LE R ADD CONSTRAINT C …
AIN D ADD CONSTRAINT C …
vincolo C a tabella R o dominio D

TION A
sserzione A

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;

re gli impiegati che hanno uno stipendio
li 2000

o>2000(Impiegati)

* FROM Impiegati
ERE Stipendio>2000;

mbolo * nella clausola di proiezione indica
le colonne delle relazione devono essere

l

Mansione	Data_A	Stipendio	Premio_P	Dip#
dirigente	02-Apr-81	2975,00	?	20
dirigente	01-Mag-81	2850,00	?	30
ingegnere	01-Giu-81	2450,00	200,00	10
ingegnere	17-Nov-81	2600,00	300,00	10
dirigente	10-Dic-80	3000,00	?	10

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

$$\pi_{Ri1.C1,\ Ri2.C2,\ .....,\ Rin.Cn}\ (\sigma_{F}(R1 \times R2 \times ... \times Rk))$$

BETWEEN permette di determinare le tuple
ono in un dato attributo valori in un intervallo

EEN v1 AND v2

a

ETWEEN v1 AND v2

Nome, Stipendio FROM Impiegati
Stipendio BETWEEN 1100 AND 1400;

Stipendio
1100,00
1300,00

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

pattern-matching su colonne di tipo stringa

di confronto espresso con l'operatore LIKE
e formato

pattern

e' una stringa di caratteri che puo' contenere
eciali % e _

% denota una sequenza di caratteri arbitrari di
ualsiasi (anche zero)

denota esattamente un carattere

erminare tutti gli impiegati che hanno 'R'
come terza lettera del cognome

Nome FROM Impiegati
Nome LIKE '_ _ R%';

ormati con gli operatori BETWEEN, IN e
no essere connessi con AND ed OR nella
condizioni complesse

contengono, in un dato attributo, uno tra i valori in un insieme specificato;

- formato

  C IN (v1, v2,....,vn) C IN *sq*
  (dove *sq* e' una sottoquery)

  forma negata

  C NOT IN (v1, v2,....,vn) 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

escente

vuole elencare mansione, nome e stipendio di
li impiegati ordinando le tuple in base alla
one in ordine crescente, ed in base allo
dio in ordine decrescente

Mansione, Stipendio, Nome
M Impiegati
ER BY Mansione, Stipendio DESC;

ne	Stipendio	Nome
	3000,00	Verdi
	2975,00	Rosi
	2850,00	Blacchi
e	2450,00	Neri
e	2000,00	Dare
e	1950,00	Gianni
e	1600,00	Rossi
e	1300,00	Milli
e	1100,00	Adami
a	1000,00	Fordi
a	800,00	Martini
a	800,00	Scotti
	1500,00	Turni
	800,00	Andrei
	800,00	Bianchi

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

rmette di correlare dati rappresentati da
erse

vede un'operazione di join esplicita;
esso in SQL tramite un prodotto Cartesiano a
licati uno o piu' *predicati di join*

di join esprime una relazione che deve essere
lle tuple risultato dell'interrogazione

terminare il nome del dipartimento in cui
l'impiegato Rossi

```
ome_Dip
 Impiegati, Dipartimenti
RE Nome = 'Rossi' AND
ati.Dip# = Dipartimenti.Dip#;
```

di join e'

```
ati.Dip# = Dipartimenti.Dip#
```

seguire il join tra una tupla di una relazione e
un'altra relazione

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

di colonna, anche espressioni aritmetiche

oni sono formulate applicando gli operatori -, *, /) ai valori delle colonne delle tuple

oni artimetiche possono comparire nella roiezione e nelle espressioni di assegnamenti di UPDATE

ovare il nome, lo stipendio, il premio di ione, e la somma dello stipendio e del premio uzione di tutti gli ingegneri

e, Stipendio, Premio_P, Stipendio+Premio_P
Impiegati
E Mansione = 'ingegnere';

pendio	Premio_P	Stipendio + Premio_P
00,00	500,00	2100,00
50,00	200,00	2650,00
00,00	300,00	2300,00
00,00	500,00	1600,00
50,00	?	1950,00
00,00	150,00	1450,00

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

oggetto di standardizzazione)

la il valore assoluto del valore numerico

b)
la il resto intero della divisione di n per b

la la radice quadrata

S supportano anche funzioni trigonometriche
er il calcolo della parte intera superiore ed

possono essere usate nella clausola di
nella clausola WHERE

d(A,5) > 3
un nome di colonna

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

i di conversione le piu' rilevanti includono

i DATE, TIME, e TIMESTAMP
o rispettivamente un valore scalare in una
empo, un timestamp

DATE('6/20/1997')

e CHAR che converte un valore di
o in una stringa di caratteri; tale funzione
re ricevere una specifica di formato

CHAR(data_assunzione, EUR)

e DAYS che converte una data in un intero
esenta il numero di giorni a partire dall'anno

DAYS('1/18/19)

la determinazione del valore di registri

NT_DATE
NT_TIME
NT_TIMESTAMP

mpi possono essere usati in espressioni
n tali espressioni e' possibile usare diverse
ali quali:
, MONTH[S], DAY[S], HOUR[S],
[S], SECOND[S]

- 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

tuple di una relazione

di gruppo si basano su due concetti

mento delle tuple di una relazione in base al
una o piu' colonne della relazione
e da usare sono specificate tramite la
GROUP BY

lla funzione di gruppo per ogni gruppo
al partizionamento
one di gruppo ha come argomento una
si applica all'insieme di valori di questa
estratti dalle tuple che appartengono allo
ppo

i gruppo comunemente presenti sono:

N, SUM, AVG, COUNT
mi includono anche STDEV e VARIANCE)

zioni di gruppo, ad accezione di COUNT,
re applicate solo su insiemi che consistono di
ci e non su insiemi di tuple

COUNT puo' avere due tipi di argomenti

di colonna - in tal caso nello standard SQL2
torio l'uso del qualificatore DISTINCT

COUNT (DISTINCT Stipendio)

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

COUNT(*)

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

di gruppo possono essere usate anche in
oin

pponiamo di voler raggruppare gli impiegati
ase del dipartimento e della mansione;
gni gruppo si vuole determinare il nome del
mento, la somma degli stipendi, quanti
gati appartengono ad ogni gruppo, e la media
tipendi

me_Dip, Mansione, SUM(Stipendio),
NT(*), AVG(Stipendio)
M Dipartimenti, Impiegati
ERE Dipartimenti.Dip#=Impiegati.Dip#
UP BYNome_Dip, Mansione;

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

e della relazione risultato ottenute
ione delle funzioni di gruppo sono colonne

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

specificare condizioni di ricerca su gruppi di

pponiamo di voler eseguire una query come la ente ma di essere interessati solo ai gruppi che gono almento due impiegati

```
me_Dip, Mansione, SUM(Stipendio),
NT(*), AVG(Stipendio)
M Dipartimenti, Impiegati
ERE Dipartimenti.Dip#=Impiegati.Dip#
UP BYNome_Dip, Mansione
ING COUNT(*) ≥ 2;
```

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

HAVING puo' essere una combinazione predicati; tali predicati tuttavia possono essere i che coinvolgono 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)

False (F), Unknown (?)

semplice valutato su un attributo a valore
ne risultato della valutazione ?

erita' di un predicato complesso viene
base alle seguenti tabelle di verita'

	?
	?
	F
	?

OR

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

cui il valore di verita' e' ? non viene restituita

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

S NULL applicato ad un dato attributo di una
sce True se la tupla ha valore nullo per

FROM R WHERE B IS NULL;
 tuple t1 e t3

FROM R WHERE B IS NULL
AND C IS NULL;
 tupla t3

FROM R WHERE B IS NULL
OR C IS NULL;
 tuple t1 e t3

S NOT NULL applicato ad un dato attributo
 restituisce True se la tupla ha valore non
tributo

FROM R WHERE B IS NOT NULL;
 tupla t2

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

o ad alcuna tupla di S

empre e' quello che si desidera

i OUTER JOIN aggiunge al risultato le tuple non hanno partecipato al join, completandole

i join originario, per contrasto, e' anche detto N

erse varianti dell'outer join: R OUTER JOIN S
a le tuple di R che quelle di S che non
no al join vengono completate e inserite nel

tuple di R che non partecipano al join
completate e inserite nel risultato
e tuple di S che non partecipano al join
completate e inserite nel risultato

OUTER puo' essere utilizzata sia per join per theta-join

egati LEFT OUTER JOIN Dipartimenti
ON Imp# = Dirigente

NATURAL RIGHT OUTER JOIN Studente

ratore e' l'UNION JOIN, che contiene ogni gni riga delle tabelle di partenza, completate n tutti gli attributi non presenti nella tupla

interrogazioni SELECT-FROM-WHERE

- in realta' SQL:1999 prevede diversi tipi di operatori di join

- questi operatori poiche' producono relazioni possono essere usati nella clausola FROM

- la forma di operatore join piu' semplice, che corrisponde al prodotto Cartesiano, e' il CROSS JOIN

  es.: Impegati CROSS JOIN Dipartimenti

- il theta-join si ottiene come operatore JOIN ON

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

- il join naturale corrisponde all'operatore NATURAL JOIN

  es.: Dipartimenti NATURAL JOIN Impiegati

- sintassi alternativa: JOIN USING (listaNomiColonne)

  es.: Dipartimenti JOIN Impiegati USING (Dip#)

eries anche piu' complesse

vuole elencare tutti gli impiegati che hanno pendio superiore alla media degli stipendi di impiegati

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

Stipendio
2975,00
2850,00
2450,00
2000,00
1950,00
3000,00

er una subquery avere al suo interno un'altra edicati di join, e tutti i predicati visti

s possono essere usate anche all'interno dei manipolazione dei dati (Insert, Delete,

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}

cui stipendio e' maggiore di *tutti* i valori
a subquery

terminare lo stipendio, la mansione, il nome e
nero di dipartimento degli impiegati che
nano piu' di *tutti gli* impiegati del
mento 30

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

ndio	Mansione	Nome	Dip#
00	dirigente	Verdi	10
00	dirigente	Rosi	20

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

errogazione; in tal caso è necessario apporre
tesi alla lista delle colonne a sinistra
e di confronto

vuole elencare gli impiegati con la stessa
ne e stipendio di Martini

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

- 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

il valore (o insieme di valori) e' usato nella
ERE della query esterna

definire subqueries che sono eseguite
e per ogni *tupla candidata* considerata nella
della query esterna

vogliono determinare gli impiegati che
nano piu' dello stipendio medio del proprio
mento

ecessaria una query esterna che selezioni gli
lla relazione Impiegati in base ad un predicato
tale query avrebbe la forma:

Nome FROM Impiegati WHERE
ndio >
ia degli stipendi nel dipartimento
mpiegato candidato);

cessaria una subquery che calcoli la media
di del dipartimento di ogni tupla candidata
ne Impiegati; tale subquery avrebbe la forma:

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

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'));
```

sa precedentemente, si richiede inoltre
amento delle tuple del risultato

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

X e' un alias di relazione (ovvero una
denota una tupla e varia nella relazione

oncetti principali che sono alla base della

y correlata fa riferimento ad un attributo
alla query esterna

bquery seleziona tuple dalla stessa relazione
query esterna, e' necessario definire un
relazione della query esterna;
deve usare l'alias per riferire i valori di
e tuple candidate nella query principale

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

bquery restituisce almeno una tupla;
valore Booleano False altrimenti;

NOT EXISTS(sq) restituisce il valore
rue se la subquery non restituisce alcuna

valore Booleano False altrimenti;

tazione di predicati con questi due operatori
ce mai il valore Unknown

supponga che la relazione Impiegati abbia
colonna addizionale, Risponde_a, che
ne per ogni tupla di Impiegati un numero di
ato. Dato un impiegato E, tale numero
l'impiegato a cui E risponde

e determinare il nome di tutti gli impiegati
nno almeno un impiegato che risponde loro

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

e determinare il nome di tutti gli impiegati
n hanno alcun impiegato che risponda loro

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

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

gazione o sottointerrogazione puo' essere
da una o piu' interrogazioni connesse
e UNION

UNION restituisce tutte le tuple distinte
almeno una delle sottointerrogazioni a cui e'

onsiderino le seguenti relazioni con lo stesso
relazione Impiegati
empo_Parziale
_Congedo

gli impiegati che abbiano lo stesso stipendio
piegati, opportunamente selezionati, contenuti
lazioni

me, Imp# FROM Impiegati
Stipendio IN
Stipendio FROM Impiegati_Tempo_Parziale
ERE Mansione = 'tecnico'

Stipendio FROM Impiegati_In_Congedo
ERE Nome ='Viola');

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#));
```

UNION elimina i duplicati dal risultato

UNION ALL non elimina i duplicati

er includere UNION ALL e' che in molti casi
ne non ci saranno duplicati nel risultato
l tentativo da parte del sistema di eliminare i
usa un peggioramento delle prestazioni

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;
```

nguaggio SQL che si ha a disposizione non
operatori INTERSECT ed EXCEPT, tali
possono essere eseguite mediante l'uso di
OT EXISTS

i nomi degli impiegati a tempo parziale che
gedo

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

i nomi degli impiegati a tempo parziale che
congedo

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

- 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;
```

erire un nuovo dipartimento; il numero del
e' 40, il nome e' Edilizia Industriale; la
D2; il dirigente e' Blacchi (Imp# 7698);
numero 6100

TO Dipartimenti
(40, 'Edilizia Industriale', 6100, 'D2',7698);

inserzione con tuple i cui valori sono ottenuti
subquery

re una relazione Promozioni
ne contiene alcune delle colonne della
piegati: Nome, Sipendio, Premio_P

erire in questa relazione tutti gli ingegneri il
di produzione e' superiore al 25% del loro
informazioni sugli ingegneri devono essere
relazione Impiegati

TO Promozioni (Nome, Sipendio, Premio_P)
Nome, Sipendio, Premio_P
M Impiegati WHERE
io_P > 0.25*Stipendio AND
sione = 'ingegnere';

so gli ingegneri inseriti sono Rossi ed Adami

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 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

cellare il dipartimento 40

ROM Dipartimenti
Dip# = 40;

ancellazione con uso di subqueries

che esista una relazione, di nome Bonus, che
ti gli impiegati che hanno diritto ad aumenti

ancellare dalla relazione Bonus tutti gli
e hanno la stessa mansione di Gianni

ROM Bonus
Mansione IN
ECT Mansione FROM Impiegati
WHERE Nome = 'Gianni');

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

```
TE Impiegati
Stipendio = Stipendio + 100
ERE Mansione = 'tecnico';
```

vuole promuovere Gianni a dirigente e
poraneamente aumentare il suo stipendio del

```
TE Impiegati
Mansione = 'dirigente',
Stipendio = 1.10*Stipendio
ERE Nome = 'Gianni';
```

```
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

ola chiave NULL (da non confondere con la
L')

ULL WHERE C=C2;

C
c1
c2
?

t1
t2
t3

(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';
```

getto al progetto 102

npiegati SET Prog#=102
Prog# IS NULL;

Mansione	Data_A	Stipendio	Premio_P	Dip#	Proj#
ngegnere	17-Dic-80	1600,00	500,00	20	101
ecnico	20-Feb-81	800,00	?	30	101
ecnico	20-Feb-81	800,00	100,00	30	101
dirigente	02-Apr-81	2975,00	?	20	101
segretaria	28-Set-81	800,00	?	30	102
dirigente	01-Mag-81	2850,00	?	30	102
ngegnere	01-Giu-81	2450,00	200,00	10	102
segretaria	09-Nov-81	800,00	?	20	101
ngegnere	17-Nov-81	2000,00	300,00	10	102
ecnico	08-Set-81	1500,00	?	30	101
ngegnere	28-Set-81	1100,00	500,00	20	101
ngegnere	03-Dic-81	1950,00	?	30	102
segretaria	03-Dic-81	1000,00	?	20	101
ngegnere	23-Gen-82	1300,00	150,00	10	102
dirigente	10-Dic-80	3000,00	?	10	102

infine di voler aumentare la lunghezza della
get

'ABLE Progetti
ER COLUMN (Budget Decimal(9,2));

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';

IEW V [(Lista Nomi Colonne)] AS Q
OCAL| CASCADED] CHECK OPTION];

della vista che viene creata; tale nome deve
rispetto a tutti i nomi di relazioni e di viste
stesso utente che definisce V

gazione di definizione della vista
lo stesso numero di colonne pari alle colonne
rtuali) specificate nella clausola di proiezione

Colonne e' una lista di nomi da assegnare alle
a vista; tale specifica non e' obbligatoria,
caso in cui l'interrogazione contenga nella
roiezione funzioni di gruppo e/o espressioni

- 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

acile per alcuni utenti lavorare con una sola
ttosto che eseguire joins tra relazioni diverse

uole creare una vista Personale che contiene
onne Nome e Mansione della relazione
ati e il nome del progetto a cui ogni
ato lavora

VIEW Personale AS
ECT Nome, Mansione, Pnome
M Impiegati, Progetti
ERE Impiegati.Prog# = Progetti.Prog#;

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

vuole definire una vista che calcoli per ogni mento alcune statistiche riguardanti lo lio degli impiegati

```
IEW Dip_S
n_S,Med_S, Max_S,Totale) AS
ECT Dip#, MIN(Stipendio), AVG(Stipendio),
      MAX(Stipendio), SUM(Stipendio)
FROM Impiegati GROUP BY Dip#;
```

l'interrogazione

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

eguente risultato

	Max_S	Totale
	3000,00	8750,00
	2975,00	7445,00
	2850,00	8700,00

- 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

e che verificano una certa condizione sono
la view

e' cosa succede se in una view in cui si puo'
sert, si esegue l'inserzione di una tupla che
la condizione specificata dalla query nella

si consideri la seguente view

VIEW imp-r AS
Imp#, Nome, Stipendio FROM Impiegati
Stipendio > 3000;

di inserire nella view la seguente tupla
s, 2000)

e specificata nella query non e' verificata dalla
pertanto la tupla viene inserita ma non e'
una query sulla view

re che le tuple inserite tramite una view (o
tramite una view) siano accettate solo se
condizione nella query della view, si usa la
TION

della definizione di una view e' pertanto il

EW view-name [(colum-names)] AS
[LOCAL|CASCADED] CHECK OPTION]

- 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

VIEW V3 AS
Imp#, Nome, Mansione,
Dip# FROM Impiegati
Mansione < > 'dirigente';

V3 e'
ome, Mansione, Data_A, Dip#)

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

viene trasformata in una operazione di insert
ne Impiegati

e' quale valore assegnare agli attributi
e Premio_P

e e' assegnare il valore nullo

ione crea problemi se il valore di Stipendio e
ve essere diverso dal valore nullo

ema giustifica la restrizione sull'inserzione

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)

ncellazione

ew prima della cancellazione

* FROM V4;

Ufficio
2100
5100
5100
2100
5100
5100
1100
2100

ew dopo la cancellazione

Ufficio
5100
5100
5100
5100
1100

precedente ha side-effects sulla view

segue una select sulla view dopo la modifica
e dello stesso ufficio di Rossi sono sparite

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;

a relazione su cui la vista e' definita

eguenti restrizioni:

bile eseguire l'operazione di DELETE se
azione di definizione della vista soddisfa le
condizioni:

a sola relazione
ntiene la clausola GROUP BY, la clausola
INCT, o una funzione di gruppo

bile eseguire l'operazione di UPDATE se
azione di definizione della vista soddisfa le
zioni precedenti ed inoltre:

olonna modificata non e' definita da
pressione

bile eseguire l'operazione di INSERT se
azione di definizione della vista soddisfa le
zioni precedenti ed inoltre:

si colonna per cui valga il vincolo NOT
L sia presente nella vista

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

lella view e' cancellata

approcci:

volta cancellata
inoperativa

perativa e' una view tale che:
nizione viene mantenuta nei cataloghi di
yscat.views in DB2)
sibile eseguire alcuna operazione sulla view
erazione di DROP

perativo di una view e' denotato da un
ag rappresentato dalla colonna STATUS del
cat.views

operativa e' automaticamente eliminata se si
mando di CREATE VIEW di una view con
ne

e e' l'unica eccezione alla restrizione che non
reare due views con lo stesso nome

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

o SQL puo' essere usato come linguaggio
n un linguaggio di programmazione (detto
spite)

are SQL da un linguaggio di programmazione
arie alcune funzionalita' aggiuntive ad SQL,
per trasferire risultati di interrogazioni in
rogramma;
nte di SQL e' detta SQL ospitato (*embedded*

mpilazione di un programma P scritto in SQL
guaggio ospite

pilazione di P

ramma viene passato ad un pre-compilatore
o del linguaggio ospite);

compilatore elimina gli statements di SQL
doli con chiamate di procedura (al DBMS)
nella sintassi del linguaggio ospite;

o e' un programma scritto completamente nel
io ospite

zione del programma risultato della fase (1)

```
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

one al DBMS

L CONNECT user_name
IDENTIFIED BYuser_psw

QL Communication Area

ecuzione di un'istruzione SQL, il DBMS
informazioni di "feedback"

azioni sono memorizzate dal DBMS nella
eve essere inclusa tramite l'istruzione
C SQL INCLUDE SQLCA)

della SQLCA e' il SQLCODE (SQLSTATUS)
BMS assegna un codice di ritorno a seguito
ione di un comando SQL; in particolare:

0 indica che il comando e' stato eseguito
ente
positivo (ad es. +100) indica che il comando
eguito correttamente ma si e' verificata
ondizione eccezionale
ica che non e' stata trovata alcuna tupla che
query)
negativo indica che il comando non e' stato
a causa di qualche errore
pio una query su una relazione non esistente)

i principio, ogni comando SQL dovrebbe
uito da un test su SQLCODE, in modo da
opportune decisioni

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 eseguibile puo' comparire dovunque possa comparire uno statement del linguaggio ospite;
diversamente da SQL interattivo, l'SQL ospitato fornisce istruzioni dichiarative (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

no, ed i progetti a cui i prodotti sono forniti

base di dati

rn#,NomeF, Volume, Citta)

me, Colore, Peso)

Nome, Citta)

P#, Pr#, Quantita)

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

he sono implementate come array di caratteri
n \0 (rappresentazione *null-terminated*)

le stringhe vengono rappresentate in modo

CHAR, si alloca una stringa comunque pari
ezza massima, inizializzando a ' ' gli ultimi
la stringa e' piu' corta
(4) stringa c9 'c', '9', ' ', ' '
VARCHAR si usa una rappresentazione detta
ing in cui all'inizio della stringa sono allocati
he contengono la lunghezza della stringa

MS (ad es. Ingres e DB2) effettuano
ente la conversione tra i due formati delle

segue la conversione che deve essere eseguita
ma applicativo quando deve passare stringhe

ede che nella DECLARE SECTION le
iti di tipo stringa siano dichiarate di tipo
dove VARCHAR e' una struct del C che
n array di carattere e di un intero

RCHAR dato_f[5] viene espansa in

gned short len;
gned char arr[5];

variabile deve avere lunghezza maggiore
rispetto alla colonna corrispondente

```
#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;
}
```

e uno degli attributi restituiti dalla
ne ha valore nullo, si ha una condizione di
CODE < 0) e le variabili di programma
erate

situazioni di errore, occorre usare le variabili
na variabile indicatore viene associata ad una
programma nella clausola INTO del comando

indicatore devono essere dichiarate come
re nella DECLARE SECTION e non possono
e nella clausola di qualificazione delle
ni

ndicatore riceve:
uale a -1 se la corrispondente colonna ha
lo
rispondente variabile non viene inizializzata)
uale a 0 se la corrispondente colonna ha
erso da nullo
rispondente variabile viene inizializzata)

QL SELECT Volume, Citta FROM Fornitori
O :volume :volumeind, :citta :cittaind
ERE Forn# = :dato_f;

eind < 0) {.....} /* volume e' nullo*/
d < 0) {.....} /* citta e' nullo*/

```
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

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

vuole inserire una nuova tupla nella relazione

e colonne P#, Pnome, e Peso sono contenuti
nte nella variabili di programma
ne, ps
e e' assegnato alla colonna Colore che riceve
valore nullo

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

pcolore e pcoloreind una variabile ospite e
le indicatore, rispettivamente, il seguente
i programma ha lo stesso effetto del comando

```
d = -1;
QL INSERT INTO Parti
 (:pno, :pnome, :pcolore :pcoloreind, :ps);
```

abili indicatore nel comando di INSERT e'
nte utile se non e' noto a priori quali sono le
e nuove tuple che ricevono valori nulli

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

```
UDE SQLCA; /* inclusione della communication area */
~], char[~~], int);
 IMMETTERE NUMERO FORNITORE:";



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

CONNECT :nome_utente IDENTIFIED BY pass_utente;
pt1(prompt, citta_data.arr, 15)) > = 0)

ata.len = strlen(citta_data.arr);
 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 */
}
 SQL CLOSE X;
 ciclo while esterno */
COMMIT RELEASE;
```

- 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

ro di parte, contenuto nella variabile di
a p#

e di citta, contenuto nella variabile di
a citta_data

cremento, contenuto nella variabile di
a incr

lo di volume, contenuto nella variabile di
a liv_volume

a esegue le seguenti attivita':

tti i fornitori della parte il cui numero e'

itore fornisce tale parte e la sua citta' e'
a citta' data, il suo volume e' incrementato
dato;

se il suo volume e' minore del livello dato,
con tutti i suoi invii e' cancellato;

ogramma stampa tutti i fornitori indicando
lo ogni fornitore e' stato trattato

- 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;
```

WHENEVER permette di semplificare la
'errore; ha il formato
EXEC SQL WHENEVER Cond Azione

ca la condizione controllata dal comando
ER; puo' avere uno dei seguenti valori

ND significa SQLCODE = 100
RNING significa
SQLCODE > 0 e SQLCODE ? 100
R significa SQLCODE < 0

l'azione da eseguire al seguito del verificarsi
zione; puo' avere uno dei seguenti valori

E che richiede di continuare l'esecuzione
el che richiede di eseguire l'istruzione la cui
label

emi includono STOP e CALL)

WHENEVER non e' un comando eseguibile; e'
ettiva al precompilatore che genera del codice
e viene inserito dopo ogni istruzione SQL

:

R condition GOTO
zione di un comando di goto dopo il comando

R condition CONTINUE
atore non inserisce alcun comando

QL - Operazioni con cursore

```
{
  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 :forn#, :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;  }
```

sono cancellate dalla base di dati

rrori lo stato della base di dati e' ripristinato
cedente all'inizio della transazione
si ha un overflow ed il programma termina in
ale

sono tentative fino a che si verifica una delle
condizioni

un comando di COMMIT che rende
e modifiche

un comando di ROLLBACK che annulla
difiche eseguite

il programma esegue un COMMIT quando
sua conclusione normale;
OLLBACK se si verificano errori

- 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

E STATEMENT
E
E
E IMMEDIATE

DECLARE *Nome* STATEMENT;
e' un nome (assegnato dall'utente) di

usato dagli altri comandi di SQL dinamico

PREPARE *Nome* FROM *Expr*;

un nome di comando; deve essere stato
tramite il comando DECLARE STATEMENT

n'espressione di tipo stringa che denota il
SQL da eseguire; molto spesso tale
e e' un nome di variabile di tipo stringa

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

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

```
BEGIN DECLARE SECTION;
..
HAR sql_source[256];
..
END DECLARE SECTION;

DECLARE sqolobj STATEMENT;
...
urce, "DELETE FROM Invii WHERE quantita' <100");
PREPARE sqlobj FROM :sqlsource;
EXECUTE  sqlobj;
```

```
BEGIN DECLARE SECTION;
..
HAR sql_source[256];
ninimo, massimo;
..
END DECLARE SECTION;

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

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

e e si "PREPARA" il comando di SELECT
' contenere la clausola INTO)

CRIBE per interrogare il sistema riguardo ai
descrizione di tali risultati e' restituita in
rea chiamata SQL Descriptor Area

area di memoria per un insieme di variabili di
cui saranno assegnati i risultati
i tale area deve essere memorizzato dal
nella SQLDA

le tuple una alla volta usando il cursore

- 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)

EXEC SQL INCLUDE SQLDA' ha come
ione nel programma di quanto segue:

della struttura SQLDA (typedef)

della struttura SQLVAR (che definisce il
'entrata allocata ad ogni singola colonna)

DASIZE(*n*) che calcola la lunghezza in bytes
ra SQLDA con *n* entrate

temi non e' necessario usare il comando
E STATEMENT

```
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;
   }
```

ovare tutte le tabelle che hanno una colonna
omincia con S

```
T TABNAME
M SYSTEM.SYSCOLUMNS WHERE
NAME LIKE 'S%';
```

trovare tutte le colonne della relazione
egati;

```
T COLNAME
M SYSTEM.SYSCOLUMNS WHERE
NAME = 'Impiegati';
```

erminare quante relazioni sono state create da

```
T COUNT(*)
M SYSTEM.SYSTABLES WHERE
NER = 'Rossi';
```

n e' possibile modificare i cataloghi
zioni sono rappresentate dalla possibilita' per
ratori della base di dati di poter modificare
informazioni sulle statistiche usate per
one)

ono modificati automaticamente dal sistema a
omandi di DDL
ATE TABLE
ATE VIEW

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

- 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)