Il linguaggio SQL

Il linguaggio SQL

- il linguaggio SQL è un linguaggio per la definizione e la manipolazione dei dati, sviluppato originariamente presso il laboratorio IBM a San Jose (California)
- è diventato standard ufficiale nel 1986 (SQL-1986)
- ultimo standard è SQL:2003
- due grosse revisioni precedenti SQL2 (o SQL-92) e SQL:1999 (o SQL3) , con caratteristiche object-relational

Il linguaggio SQL

- è il linguaggio oggi più 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)
  - MySQL
  - PostgreSQL
  - InterBase

SQL - Tipi di dato

- I tipi di dato in SQL:2003 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:2003
- i tipi predefiniti sono suddivisi in quattro categorie
  - tipi numerici
  - tipi carattere
  - tipi temporali
  - tipi booleani

SQL - Tipi di dato 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)

SQL - Tipi di dato numerici

Tipi numerici esatti:

- **INTEGER** rappresenta i valori interi
  - la precisione (numero totale di cifre) di questo tipo di dato è espressa in numero di bit o cifre, a seconda della specifica implementazione di SQL
- **SMALLINT** rappresenta i valori interi
  - l'unico requisito è che la precisione di questo tipo di dato sia non maggiore della precisione del tipo di dato INTEGER
  - usato per eventuali ottimizzazioni in quanto i valori richiedono minore spazio di memorizzazione
- **BIGINT** rappresenta i valori interi
  - l'unico requisito è che la precisione di questo tipo di dato sia non minore della precisione del tipo di dato INTEGER

SQL - Tipi di dato numerici

Tipi numerici esatti:

- **NUMERIC** rappresenta i valori decimali
  - è caratterizzato da una precisione (numero totale di cifre) e una scala (numero di cifre della parte frazionaria)
  - la specifica di questo tipo di dato ha la forma NUMERIC [(precisione[,scala])]
  - ad esempio, NUMERIC (4,1) corrisponde ai valori tra -999.9 a 999.9
  - sia precisione che scala possono essere omesse, il default per la precisione è 1, per la scala è 0

SQL - Tipi di dato numerici

Tipi numerici esatti:

- **DECIMAL** rappresenta i valori decimali
  - è simile al tipo NUMERIC
  - la specifica di questo tipo di dato ha la forma DECIMAL [(precisione[,scala])]
- La differenza tra NUMERIC e DECIMAL è che il primo deve essere implementato esattamente con la precisione richiesta, mentre il secondo può avere una precisione maggiore

SQL - Tipi di dato numerici

Tipi numerici approssimati:

- **REAL** rappresenta valori a singola precisione in virgola mobile
  - la precisione dipende dalla specifica implementazione di SQL
- **DOUBLE PRECISION** rappresenta valori a doppia precisione in virgola mobile
  - la precisione dipende dalla specifica implementazione di SQL
- **FLOAT** permette di richiedere la precisione che si desidera
  - formato FLOAT (precisione)

SQL - Tipi di dato carattere

- **CHARACTER** rappresenta stringhe di caratteri di lunghezza massima predefinita
  - è spesso abbreviato in CHAR
  - la specifica ha il formato CHAR [(length)] dove length è la lunghezza massima delle stringhe
  - se non viene specificata la lunghezza il default è 1
- **CHARACTER VARYING** rappresenta stringhe di caratteri di lunghezza massima predefinita
  - è spesso abbreviato in VARCHAR
  - la specifica ha il formato VARCHAR [(length)] dove length è la lunghezza massima delle stringhe

SQL - Tipi di dato carattere

- La differenza è che per il tipo CHAR si alloca per ogni stringa la lunghezza massima predefinita, mentre per VARCHAR si adottano strategie diverse
- **CHARACTER LARGE OBJECT (CLOB)**
  - permette di definire sequenze di caratteri di elevate dimensioni (testo)
- **analogo binario: BINARY LARGE OBJECT (BLOB)**
  - permette di definire sequenze di bit di elevate dimensioni (immagini)

SQL - Tipi di dato temporali

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

  i valori di questo tipo hanno il formato DATE 'yyyy-mm-dd'
- **TIME [WITH TIME ZONE]** rappresenta i tempi espressi come ora (2 cifre), minuto (2 cifre) e secondo (2 cifre)

  i valori di questo tipo hanno il formato
  TIME 'hh:mm:ss[.[nnnnnn]] [TimeZone]'
  dove TimeZone è espresso nel formato {+|-} hh:mm

SQL - Tipi di dato temporali

- **TIMESTAMP [WITH TIME ZONE]** rappresenta una "concatenazione" dei due tipi di dato precedenti
  - permette di rappresentare timestamp che consistono in: anno, mese, giorno, ora, minuto, secondo e microsecondo
  - i valori di questo tipo hanno il formato
    TIMESTAMP 'yyyy-mm-dd hh:mm:ss[.[nnnnnn]] [TimeZone]'
    dove TimeZone è espresso nel formato {+|-} hh:mm
- **INTERVAL** rappresenta una durata temporale in riferimento a uno o più dei qualificatori YEAR, MONTH, DAY, HOUR, MINUTE, SECOND
  - es. INTERVAL '3' YEAR
    INTERVAL '-36 22:30' DAY TO MINUTE

SQL - Tipi di dato booleani

- **BOOLEAN** rappresenta i valori booleani
  - i valori di tali tipo sono
    TRUE, FALSE, UNKNOWN
  - il valore UNKNOWN viene introdotto per la gestione dei confronti con valori nulli
  - nel seguito vedremo come gli operatori booleani AND, OR, NOT vengano estesi al valore UNKNOWN

SQL - Domini

- È possibile definire domini ed utilizzarli nella definizione di tabelle
- un dominio è 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 NomeDominio
  AS DescrizioneTipo
  [DEFAULT ValoreDefault];
- Esempio
  CREATE DOMAIN DominioMansione
  AS CHAR(10)
  DEFAULT 'impiegato';

SQL – Sequence

- È possibile utilizzare un meccanismo per la generazione automatica di valori sequenziali
- sintassi:
  CREATE SEQUENCE NomeSeq AS Tipo
  [START WITH Valore] [INCREMENT BY Valore]
  [MINVALUE Valore] [MAXVALUE Valore]
  {CYCLE|NO CYCLE};
- il tipo deve essere numerico con scala 0
- l'incremento di default è 1
- genera numeri tra MAXVALUE e MINVALUE, partendo da START e con l'incremento specificato (che può essere negativo, ma non 0)

SQL – Sequence

- Se si esce dal range dei valori (<MINVALUE o >MAXVALUE), con CYCLE si rigenera un valore nell'intervallo, con NO CYCLE si solleva un errore
- Esempio
  CREATE SEQUENCE ImpSeq AS INTEGER
  START WITH 1 INCREMENT BY 1 MINVALUE 1
  MAXVALUE 9999 CYCLE;
- per generare nuovi valori si usa il comando NEXTVALUE FOR NomeSeq;
- utili per realizzare codici, matricole, …

SQL - DDL: Creazione di tabelle

- La creazione avviene tramite il comando di
  CREATE TABLE
- sintassi:
  CREATE TABLE NomeTabella
  ($spec_col_1$ [, …, $spec_col_n$])
  dove:
  - NomeTabella è il nome della relazione che viene creata
  - $spec_col_i$ (i=1, …, n) è una specifica di colonna, il cui formato è il seguente
    $NomeColonna_i$ $Dominio_i$ [DEFAULT ValoreDefault]

SQL - DDL: Creazione di tabelle

- 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));
```


SQL - DDL: Creazione di tabelle

- Esempio con dominio:

```
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));
```

- i valori di Imp# possono essere generati attraverso una SEQUENCE

SQL - Vincoli di integrità

- Un vincolo è una regola che specifica delle condizioni sui valori
- un vincolo può essere associato ad una tabella, ad un attributo, ad un dominio
- in SQL è possibile specificare diversi tipi di vincoli:
  - chiavi (UNIQUE e PRIMARY KEY)
  - obbligatorietà di attributi (NOT NULL)
  - chiavi esterne (FOREIGN KEY)
  - vincoli CHECK (su attributo o su tupla) e asserzioni (vincoli CHECK su più tuple o tabelle) – li vedremo dopo il linguaggio di interrogazione

SQL - Chiavi

- La specifica delle chiavi si effettua in SQL mediante le parole chiave UNIQUE o PRIMARY KEY
  - UNIQUE garantisce che non esistano due tuple che condividono gli stessi valori non nulli per gli attributi (le colonne UNIQUE possono contenere valori nulli)
  - PRIMARY KEY impone che per ogni tupla i valori degli attributi specificati siano non nulli e diversi da quelli di ogni altra tupla
- in una tabella è possibile specificare più chiavi UNIQUE ma una sola PRIMARY KEY

SQL - Chiavi

- Se la chiave è formata da un solo attributo è sufficiente far seguire la specifica dell'attributo da UNIQUE o PRIMARY KEY
- alternativamente si può far seguire la definizione della tabella da

  PRIMARY KEY (ListaNomiColonne)

  o

  UNIQUE(ListaNomiColonne)

SQL - Chiavi

- Esempio:

```
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) DEFAULT 0,
     Dip#        Decimal(2));
```

SQL - Chiavi

- Esempio:

```
CREATE TABLE Impiegati
    (Imp#           Decimal(4) PRIMARY KEY,
     CodiceFiscale  Char(16) UNIQUE,
     Nome           Char(20),
     Mansione       DominioMansione,
     Data_A         Date,
     Stipendio      Decimal(7,2),
     Premio_P       Decimal(7,2) DEFAULT 0,
     Dip#           Decimal(2));
```


SQL - Chiavi

- Esempio:

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


SQL - NOT NULL

- Per specificare che una colonna non può assumere valori nulli è sufficiente includere il vincolo NOT NULL nella specifica della colonna
- le colonne dichiarate PRIMARY KEY non possono assumere valori nulli

SQL - NOT NULL

- Esempio:

```
CREATE TABLE Impiegati
    (Imp#           Decimal(4) PRIMARY KEY,
     CodiceFiscale  Char(16) UNIQUE,
     Nome           Char(20) NOT NULL,
     Mansione       DominioMansione,
     Data_A         Date,
     Stipendio      Decimal(7,2) NOT NULL,
     Premio_P       Decimal(7,2) DEFAULT 0,
     Dip#           Decimal(2) NOT NULL);
```


SQL - Chiavi esterne

- La specifica di chiavi esterne avviene mediante la clausola (opzionale) FOREIGN KEY del comando CREATE TABLE
- sintassi:

```
[, FOREIGN KEY (ListaNomiColonne)
   REFERENCES NomeTabellaRiferita
                        [(ListaNomiColonneRiferite)]
  [ 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.....]]);
```


SQL - Chiavi esterne

- ListaNomiColonne è un sottoinsieme delle colonne della tabella che corrisponde ad una chiave (UNIQUE o primaria) della tabella riferita
- non è necessario che i nomi siano gli stessi, ma i domini degli attributi corrispondenti devono essere compatibili
- è possibile (obbligatorio in caso di ambiguità) indicare esplicitamente quali sono le colonne riferite
- nel caso di chiave esterna costituita da un solo attributo si può far seguire la specifica della colonna da REFERENCES NomeTabellaRiferita

SQL - Chiavi esterne

- Il tipo di match è significativo nel caso di chiavi esterne costituite da più di un attributo e in presenza di valori nulli
  - MATCH SIMPLE: il vincolo di integrità referenziale è soddisfatto se per ogni tupla della tabella referente
    - almeno una delle colonne della chiave esterna è NULL, oppure
    - nessuna di tali colonne è NULL ed esiste una tupla della tabella riferita la cui chiave coincide con i valori di tali colonne

    opzione di default

SQL - Chiavi esterne

  - MATCH FULL: il vincolo di integrità referenziale è soddisfatto se per ogni tupla della tabella referente
    - tutte le colonne della chiave esterna sono NULL, oppure
    - nessuna di tali colonne è NULL ed esiste una tupla della tabella riferita la cui chiave coincide con i valori di tali colonne
  - MATCH PARTIAL: il vincolo di integrità referenziale è 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

SQL - Chiavi esterne

- Esempio
  - tabella riferita con tuple con valore di chiave

    10, 'mario'

    20, 'giuseppe'
  - se consideriamo MATCH SIMPLE
    - i seguenti valori sono valori legali per le chiavi esterne

      10, 'mario'

      NULL, 'giuseppe'

      10, NULL

      NULL, 'luca'

      30, NULL
    - ovviamente non è valore legale 10, 'giuseppe' né 10, 'luca'

SQL - Chiavi esterne

- Esempio (segue)
  - se consideriamo MATCH FULL
    - i seguenti valori sono valori legali per le chiavi esterne

      10, 'mario'

      NULL, NULL
    - non sono valori legali

      10, NULL

      NULL, 'giuseppe'

      nè

      NULL, 'luca'

      30, NULL

      10, 'giuseppe'

      10, 'luca'

SQL - Chiavi esterne

- Esempio (segue)
  - se consideriamo MATCH PARTIAL
    - i seguenti valori sono valori legali per le chiavi esterne

      10, 'mario'

      NULL, NULL

      10, NULL

      NULL, 'giuseppe'
    - ma non sono valori legali

      NULL, 'luca'

      30, NULL

      10, 'giuseppe'

      10, 'luca'

SQL - Chiavi esterne

- Opzioni riguardanti le azioni da eseguire nel caso di cancellazione di una tupla riferita tramite chiave esterna (ON DELETE):
  - NO ACTION: la cancellazione di una tupla dalla tabella riferita è eseguita solo se non esiste alcuna tupla nella tabella referente che ha come chiave esterna la chiave della tupla da cancellare
  - RESTRICT: come per NO ACTION, con la differenza che questa opzione viene controllata subito, mentre NO ACTION viene considerata dopo che sono state esaminate tutte le specifiche relative all'integrità referenziale

SQL - Chiavi esterne

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

SQL - Chiavi esterne

- Opzioni riguardanti le azioni da eseguire nel caso di modifica della chiave della tupla riferita tramite chiave esterna (ON UPDATE):
  - hanno lo stesso significato delle opzioni viste per la cancellazione
  - differenza: l'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 per l'aggiornamento

SQL - Chiavi esterne

- In caso di più riferimenti l'ordine in cui vengono considerate le opzioni è
  - RESTRICT
  - CASCADE, SET NULL, SET DEFAULT
  - NO ACTION
- nel caso di inserimento o modifica nella tabella referente non è possibile specificare alcuna opzione e viene applicata sempre NO ACTION

SQL - Chiavi esterne

- Esempio

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

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


SQL - Chiavi esterne

- Esempio (segue)

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


SQL - Chiavi esterne

- Esempio Impiegati-Dipartimenti

```
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) DEFAULT 0,
     Dip#       Decimal(2) REFERENCES Dipartimenti);
CREATE TABLE Dipartimenti
    (Dip#       Decimal(2) PRIMARY KEY,
     Nome_Dip   Char(20),
     Ufficio    Decimal(4),
     Divisione  Char(2),
     Dirigente  Decimal(4) REFERENCES Impiegati);
```

SQL - DDL
Cancellazioni e modifiche

- DROP TABLE R
  - cancella la relazione R
  - esempio:
    DROP TABLE Impiegati;
- RENAME R_v TO R_n
  - modifica il nome della relazione da R_v a R_n
  - esempio:
    RENAME Impiegati TO Imp;

SQL - DDL
Cancellazioni e modifiche

- 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));

SQL - DDL
Cancellazioni e modifiche

- ALTER TABLE R DROP COLUMN nome_col
  - rimuove una colonna da una relazione
  - esempio:
    ALTER TABLE Impiegati
    DROP COLUMN Premio_P;

SQL - DDL
Cancellazioni e modifiche

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

SQL - DDL
Cancellazioni e modifiche

- ALTER TABLE R DROP CONSTRAINT C
- ALTER DOMAIN D DROP CONSTRAINT C
  - rimuove il 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 - Interrogazioni

- Il formato di base di un'interrogazione in SQL è:
  SELECT {DISTINCT $R_{i1}.C_1$, $R_{i2}.C_2$, ..., $R_{in}.C_n$ | *}
  FROM R_1, R_2, ..., R_k
  WHERE F;

dove

- R_1, R_2, ..., R_k è una lista di nomi distinti di relazioni
- $R_{i1}.C_1$, $R_{i2}.C_2$, ..., $R_{in}.C_n$ è 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 può usare C invece di R.C
- il simbolo * indica che tutte le colonne delle relazione devono essere ritrovate
- F è un predicato analogo ai predicati visti nel caso dell'operazione relazionale σ

SQL - Interrogazioni

- Il significato dell'interrogazione SQL:

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

  FROM R_1, R_2, ..., R_k

  WHERE F;

  può essere espresso per mezzo della seguente espressione algebrica

  $$\pi_{R_{i1}.C_1,\ R_{i2}.C_2,\ .....,\ R_{in}.C_n}(\sigma_F(R_1 \times R_2 \times ... \times R_k))$$

SQL - Interrogazioni

- Il significato dell'interrogazione SQL:

  SELECT *

  FROM R_1, R_2, ..., R_k

  WHERE F;

  può essere espresso per mezzo della seguente espressione algebrica

  $$\sigma_F(R_1 \times R_2 \times ... \times R_k)$$

  in cui non viene effettuata alcuna proiezione

SQL - Interrogazioni

- **esempi dalla base di dati impiegati e dipartimenti**
  - **Q1:** selezionare gli impiegati che hanno uno stipendio maggiore di 2000

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

  SELECT * FROM Impiegati
  WHERE Stipendio>2000;

SQL - Interrogazioni

- Risultato Q1:

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

SQL - Interrogazioni

- **esempi dalla base di dati impiegati e dipartimenti**
  - **Q2:** selezionare il nome e il numero di dipartimento degli impiegati che hanno uno stipendio maggiore di 2000 e hanno mansione di ingegnere

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

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

- risultato Q2:

Nome	Dip#
Neri	10
Dare	10

SQL - Interrogazioni

- **esempi dalla base di dati impiegati e dipartimenti**
  - **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 DISTINCT Imp# FROM Impiegati
  WHERE Dip#=30 AND
  (Mansione = 'ingegnere' OR Mansione = 'tecnico');

  - risultato Q3:

Imp#
7499
7521
7844
7900

SQL - Interrogazioni

Eliminazione dei duplicati

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

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

- risultato

Mansione
ingegnere
tecnico
dirigente
segretaria

SQL - Interrogazioni

Eliminazione dei duplicati

- è possibile non richiedere l'eliminazione dei duplicati omettendo la clausola DISTINCT

```
SELECT Mansione FROM Impiegati;
```

- risultato

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

SQL – Interrogazioni

Operatori e funzioni

- SQL fornisce numerosi operatori di confronto e funzioni predefinite, che possono essere usati per specificare predicati ed espressioni
- vediamo i più comuni
- operatori di confronto:
  - quelli dell'algebra (=,>,<,>=,<=,<>)
  - BETWEEN, IN, LIKE, OVERLAPS
- funzioni:
  - aritmetiche
  - su stringhe
  - su date

SQL - Interrogazioni

Condizioni su intervalli di valori

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

  C BETWEEN v_1 AND v_2

- forma negata

  C NOT BETWEEN v_1 AND v_2

SQL - Interrogazioni

Condizioni su intervalli di valori

- esempio

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

- risultato

Nome	Stipendio
Adami	1100,00
Milli	1300,00

SQL - Interrogazioni

Ricerca di valori in un insieme

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

  C IN (v_1, v_2,....,v_n)

  C IN *sq* (dove *sq* è una sottoquery)

- forma negata

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

  C NOT IN *sq*

SQL - Interrogazioni

Ricerca di valori in un insieme

- esempio

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

- risultato

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

SQL - Interrogazioni

Confronto tra stringhe di caratteri

- l'operatore LIKE permette di eseguire alcune semplici operazioni di *pattern-matching* su colonne di tipo stringa
- un predicato di confronto espresso con l'operatore LIKE ha il seguente formato

  C [NOT] LIKE *pattern*

  dove *pattern* è una stringa di caratteri che può contenere i caratteri speciali % e _
  - il carattere % denota una sequenza di caratteri arbitrari di lunghezza qualsiasi (anche zero)
  - il carattere _ denota esattamente un carattere

SQL - Interrogazioni

Confronto tra stringhe di caratteri

- 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

SQL - Interrogazioni

Confronto tra stringhe di caratteri

- l'operatore SIMILAR TO permette di esprimere condizioni di *matching* più complesse ma è più inefficiente
- per questo motivo è usato più frequentemente nella specifica di vincoli CHECK che nelle interrogazioni
- un predicato di confronto espresso con l'operatore SIMILAR TO ha il seguente formato

  C [NOT] SIMILAR TO *pattern*

  dove *pattern* è una stringa di caratteri che può contenere, oltre ai caratteri % e _, gli operatori:

SQL - Interrogazioni

Confronto tra stringhe di caratteri

- *, ripetizione da 0 a n volte
- +, ripetizione da 1 a n volte
- [InsiemediCaratteri], un carattere nell'insieme
- [Carattere1-Carattere2], un carattere nel range
- [^InsiemeoRange], negazione
- [:ALPHA:], [:UPPER:], [:LOWER:], [:DIGIT:], [:ALNUM:], caratteri alfabetici, maiuscoli, minuscoli, numerici, alfanumerici
- |, or
- ||, concatenazione

- esempio: '[A-C]|| [:DIGIT:]||[:ALPHA:]*'

SQL - Interrogazioni

Confronto tra periodi temporali

- è presente un predicato OVERLAPS che restituisce il valore Booleano true se due periodi temporali hanno intersezione non vuota
- si applica tra coppie di valori temporali (inizio,fine) o tra coppie valore temporale-intervallo (inizio,durata)
- esempi:
  - (DATE '1994-01-01', DATE '1994-05-01') OVERLAPS (DATE '1993-07-01', DATE '1994-03-01')
  - (DATE '1994-01-01', INTERVAL '05' MONTH) OVERLAPS (DATE '1993-07-01', INTERVAL '08' MONTH)

SQL - Interrogazioni

Espressioni e funzioni

- i predicati usati nelle interrogazioni possono coinvolgere, oltre a nomi di colonna, anche espressioni
- tali espressioni sono formulate applicando operatori ai valori delle colonne delle tuple
- esempi di espressioni e funzioni sono quelle aritmetiche, su stringhe, su date e tempi
- le espressioni possono comparire nella clausola di proiezione, nelle clausole WHERE e nelle espressioni di assegnamento del comando di UPDATE

SQL - Interrogazioni

Espressioni e funzioni

- una espressione usata nella clausola di proiezione di un'interrogazione dà luogo ad una colonna, detta *virtuale*, non presente nella relazione su cui si effettua l'interrogazione
- le colonne virtuali non sono fisicamente memorizzate, ma sono *calcolate* come risultato delle interrogazioni
- è possibile assegnare un nome ad una colonna virtuale con la sintassi

  AS NomeColonna
- poichè la clausola di proiezione è l'ultima ad essere valutata, non è possibile utilizzare tale nome nelle altre clausole dell'interrogazione

SQL - Interrogazioni

Espressioni e funzioni aritmetiche

- semplici espressioni possono essere formulate applicando gli operatori aritmetici (+, -, *, /) ai valori delle colonne delle tuple
- 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';
```


SQL - Interrogazioni

Espressioni e funzioni aritmetiche

- 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	?	?
Milli	1300,00	150,00	1450,00

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

SQL - Interrogazioni

Espressioni e funzioni aritmetiche

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

- risultato

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

- in questo esempio si è assegnato il nome StipendioTot alla colonna virtuale

SQL - Interrogazioni

Espressioni e funzioni aritmetiche

- funzioni:
  - ABS(n) calcola il valore assoluto del valore numerico n
  - MOD(n,b) calcola il resto intero della divisione di n per b
- altre funzioni: per il calcolo della radice quadrata, della parte intera superiore ed inferiore, funzioni trigonometriche
- le funzioni possono essere usate nella clausola di proiezione e nella clausola WHERE

  es. WHERE MOD(A,5) > 3 dove A è un nome di colonna

SQL - Interrogazioni

Espressioni e funzioni per stringhe

- operatore di concatenazione denotato da ||

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

  restituisce un'unica stringa che contiene cognome, nome ed indirizzo separati da uno spazio bianco
- funzioni:
  - SUBSTRING (str FROM m [FOR n]) estrae dalla stringa str la sottostringa dal carattere di posizione m per una lunghezza n (se n è specificato) oppure fino all'ultimo carattere

SQL - Interrogazioni

Espressioni e funzioni per stringhe

- funzioni:
  - OVERLAY (str_1 PLACING str_2 FROM m [FOR n]) estrae la sottostringa dalla posizione m alla n da str_1 e la sostituisce con str_2
  - TRIM ([[{LEADING|TRAILING|BOTH} [str_1] FROM] str_2 elimina dalla stringa str_2 i caratteri in str_1 (se non specificata gli spazi), dalle posizioni iniziali/seguenti o entrambe (default: BOTH)
  - {UPPER | LOWER} (str) trasformano la stringa str in caratteri tutti maiuscoli o tutti minuscoli, rispettivamente

SQL - Interrogazioni

Espressioni e funzioni per stringhe

- funzioni:
  - POSITION (str_1 IN str_2) restituisce la posizione del primo carattere in cui str_1 appare come sottostringa in str_2
  - {BIT|CHAR|OCTET}_LENGTH(str) restituisce la lunghezza della stringa str, rispettivamente in numero di bit, caratteri, byte
  - inoltre funzioni per convertire in un altro character set

SQL - Interrogazioni

Espressioni e funzioni per date e tempi

- è possibile applicare le funzioni aritmetiche + e - tra tra intervalli e tra valori temporali e intervalli
- è inoltre possibile applicare * e / tra un intervallo e un numero e la funzione ABS ad intervalli
- funzioni:
  - zerarie: CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME,LOCALTIMESTAMP
  - EXTRACT (campo FROM expr) estrae il campo corrispondente al qualificatore temporale specificato dall'espressione
    es. EXTRACT (DAY FROM DATE '1969-10-08')

SQL - Interrogazioni

Espressioni e funzioni per date e tempi

- esempio: si vuole avere un colloquio con tutti i nuovi impiegati dopo 90 giorni dalla loro assunzione. Per tutti gli impiegati del dipartimento 10 per cui si deve ancora effettuare il colloquio, si vuole determinare il nome, la data di assunzione e la data del colloquio

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


SQL - Interrogazioni

Espressione di CAST

- è possibile convertire un valore ad un altro tipo mediante l'operatore di CAST

  CAST (EspressioneScalare) AS Target
- esempi di conversioni possibili:
  - tra numeri/stringhe di caratteri/valori temporali
  - tra numero (esatto o approssimato) e stringa (lunghezza fissa o variabile)
  - tra valori temporali e stringhe di caratteri
  - tra numeri esatti e intervalli

SQL - Interrogazioni

Espressione di CAST

- sono inoltre possibili diversi cast tra tipi temporali
  - da DATE a TIMESTAMP:ai campi di TIME viene assegnato 00:00:00
  - da TIME a TIMESTAMP: ai campi di DATE viene assegnato CURRENT_DATE
  - da TIMESTAMP a DATE/TIME: si proietta sui campi di interesse
  - tra intervalli a granularità diverse, quanto è possibile effettuare la conversione
    es. CAST(INTERVAL '3' YEAR TO INTERVAL MONTH) è INTERVAL '36' MONTH

SQL - Interrogazioni

Ordinamento del risultato di una query

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

SQL - Interrogazioni

Ordinamento del risultato di una query

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

- risultato - le tuple sono ordinate in ordine crescente

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

SQL - Interrogazioni

Ordinamento del risultato di una query

- l'ordinamento non è limitato ad una sola colonna, nè ad un ordine crescente
- esempio: si vuole elencare mansione, stipendio e nome 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;
```


SQL - Interrogazioni

Ordinamento del risultato di una query

- 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

Operazione di join

- l'operazione di join rappresenta un'importante operazione in quanto permette di correlare dati rappresentati da relazioni diverse
- tradizionalmente il join è espresso in SQL tramite un prodotto Cartesiano a cui sono applicati uno o più *predicati di join*
- un predicato di join esprime una relazione che deve essere verificata dalle tuple risultato dell'interrogazione
- vedremo che attualmente SQL prevede anche un'operazione di join esplicita

SQL - Interrogazioni

Operazione di join

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

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


SQL - Interrogazioni

Operazione di join

- SQL:2003 prevede anche operatori di join espliciti
- questi operatori poiché producono relazioni possono essere usati nella clausola FROM
- la forma di operatore di join più semplice, che corrisponde al prodotto Cartesiano, è il CROSS JOIN
  - es. Impiegati CROSS JOIN Dipartimenti
- il theta-join è espresso dall'operatore JOIN ON
  - es. Impiegati JOIN Dipartimenti ON Dipartimenti.Dip# > Imp#

SQL - Interrogazioni

Operazione di join

- il join naturale corrisponde all'operatore NATURAL JOIN
  - es. Impiegati NATURAL JOIN Dipartimenti
  - viene richiesta l'uguaglianza dei valori degli attributi che hanno lo stesso nome nelle due relazioni
- sintassi alternativa: JOIN USING (ListaNomiColonne)
  - es. Impiegati JOIN Dipartimenti USING (Dip#)
  - viene richiesta l'uguaglianza dei valori degli attributi specificati nella clausola USING

SQL - Interrogazioni

Operazione di join

- esempio sintassi alternative
  -
```
SELECT  Nome_Dip
FROM Impiegati JOIN Dipartimenti
        ON Impiegati.Dip# = Dipartimenti.Dip#
WHERE Nome = 'Rossi';
```
  -
```
SELECT  Nome_Dip
FROM Impiegati NATURAL JOIN Dipartimenti
WHERE Nome = 'Rossi';
```
  -
```
SELECT  Nome_Dip
FROM Impiegati JOIN Dipartimenti USING (Dip#)
WHERE Nome = 'Rossi';
```


SQL - Interrogazioni

Operazione di join

- si noti che l'operazione di NATURAL JOIN non corrisponde completamente al join natural algebrico: non si esegue alcuna proiezione, e lo schema risultante è quello del prodotto Cartesiano
- il risultato di un'operazione di join è una relazione; è pertanto possibile richiedere che tale relazione sia ordinata anche in base a valori di colonne di relazioni diverse o che le tuple duplicate siano eliminate

SQL - Interrogazioni

Operazione di join

- esempio: 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;
```

SQL - Interrogazioni

Operazione di join

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

SQL - Interrogazioni

Outer join

- In R JOIN S non si ha traccia delle tuple di R che non corrispondono ad alcuna tupla di S
- questo non è sempre 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, è anche detto INNER JOIN

SQL - Interrogazioni

Outer join

- Esistono diverse varianti dell'outer join, consideriamo R OUTER JOIN S:
  - FULL: sia le tuple di R che quelle di S che non partecipano al join vengono completate ed inserite nel risultato
  - LEFT: le tuple di R che non partecipano al join vengono completate ed inserite nel risultato
  - RIGHT: le tuple di S che non partecipano al join vengono completate ed inserite nel risultato

SQL - Interrogazioni

Outer join

- la variante OUTER può essere utilizzata sia per join naturale che per theta-join
- esempi
  - Impiegati LEFT OUTER JOIN Dipartimenti
    ON Imp# = Dirigente
    se un impiegato è dirigente di un dipartimento, avrà associate le informazioni del dipartimento, altrimenti NULL
  - Relatore NATURAL RIGHT OUTER JOIN Studente
    se uno studente ha un relatore, avrà associate le sue informazioni, altrimenti NULL

SQL - Interrogazioni

Outer join - esempio

R

A	B
a	1
b	2

S

B	C
1	x
3	v

- SELECT A, R.B, C
  FROM R NATURAL LEFT OUTER JOIN S;

A	B	C
a	1	x
b	2	NULL

SQL - Interrogazioni

Outer join - esempio

- SELECT A, S.B, C
  FROM R NATURAL RIGHT OUTER JOIN S;

A	B	C
a	1	x
NULL	3	v

- SELECT A, R.B, S.B, C
  FROM R NATURAL FULL OUTER JOIN S;

A	R.B	S.B	C
a	1	1	x
b	2	NULL	NULL
NULL	NULL	3	v

SQL - Interrogazioni

Union join

- esiste un ulteriore operatore di join, UNION JOIN, tale che R UNION JOIN S contiene ogni colonna e ogni riga di R e di S, completate con NULL in tutti gli attributi non presenti nella tupla originaria
- esempio: R UNION JOIN S

A	R.B	S.B	C
a	1	NULL	NULL
b	2	NULL	NULL
NULL	NULL	1	x
NULL	NULL	3	v

SQL - Funzioni di gruppo

- le funzioni di gruppo permettono di estrarre informazioni aggregate da insiemi di valori
- le funzioni di gruppo comunemente presenti sono:
  - MAX, MIN, SUM, AVG, COUNT
    (alcuni sistemi includono anche STDEV e VARIANCE)
- le funzioni SUM e AVG sono definite solo per insiemi di valori numerici

SQL - Funzioni di gruppo

- tutte le funzioni di gruppo, ad accezione di COUNT, sono applicate su insiemi di valori semplici e non su insiemi di tuple, quindi hanno come argomento un nome di colonna, ed opzionalmente il qualificatore DISTINCT
- la funzione di gruppo si applica all'insieme di valori di colonna estratti dalle tuple che soddisfano la clausola WHERE dell'interrogazione (alle tuple che soddisfano la clausola WHERE dell'interrogazione nel caso di COUNT)
- l'insieme di valori è denotato nel caso più semplice da un nome di colonna, in generale da un'espressione contenente un nome di colonna

SQL - Funzioni di gruppo

- la funzione COUNT può avere tre tipi di argomenti
  - il carattere speciale '*' - in tal caso la funzione restituisce il numero di tuple presenti in un dato gruppo
    esempio: COUNT(*)
  - un nome di colonna - in tal caso la funzione restituisce il numero di valori non nulli per l'attributo
    esempio: COUNT (Stipendio)
  - un nome di colonna preceduto dal qualificatore DISTINCT - in tal caso la funzione restituisce il numero di valori non nulli e distinti per l'attributo
    esempio: COUNT (DISTINCT Stipendio)

SQL - Funzioni di gruppo

- per le funzioni MIN e MAX è indifferente utilizzare o meno il qualificatore DISTINCT
- per SUM e AVG, se l'argomento è
  - un nome di colonna - la funzione viene applicata a tutti i valori non nulli per l'attributo
    esempio: SUM (Stipendio)
  - un nome di colonna preceduto dal qualificatore DISTINCT - la funzione viene applicata all'insieme dei valori non nulli e distinti per l'attributo
    esempio: SUM (DISTINCT Stipendio)

SQL - Funzioni di gruppo

- le colonne ottenute dall'applicazione di funzioni di gruppo, come quelle ottenute dall'applicazione di funzioni aritmetiche, sono colonne virtuali, cui è possibile assegnare un nome con la clausola AS
- le funzioni di gruppo possono a loro volta essere utilizzate in espressioni aritmetiche
  es. SUM(Stipendio) + SUM(Premio_P)
- se l'insieme dei valori cui applicare la funzione di gruppo è vuoto
  - la funzione COUNT restituisce il valore numerico 0
  - le alttre funzioni restituiscono il valore NULL

SQL - Funzioni di gruppo

- esempio: si vuole determinare il massimo stipendio

```
SELECT MAX(Stipendio) FROM Impiegati;
```

- risultato

MAX(Stipendio)
3000,00

- esempio: si vuole determinare il massimo stipendio del dipartimento 20, chiamando MaxStip la colonna virtuale

```
SELECT MAX(Stipendio) AS MaxStip
FROM Impiegati WHERE Dip#=20;
```

- risultato

MaxStip
2975,00

SQL – Raggruppamento

- l'operatore di raggruppamento permette di partizionare le tuple di una relazione in base al valore di una o più colonne della relazione, specificate nella clausola GROUP BY del comando SELECT
- solo le tuple che soddisfano la clausola WHERE partecipano al partizionamento
- il risultato di una query che contiene una clausola GROUP BY contiene tante tuple quanti sono i gruppi di tuple risultanti dal partizionamento
- la clausola di proiezione di una query contenente la clausola GROUP BY può solo includere:
  - una o più colonne tra le colonne che compaiono nella clausola GROUP BY
  - funzioni di gruppo

SQL-Raggruppamento

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

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

- risultato

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

SQL - Raggruppamento

- in una query contenente una clausola GROUP BY, ogni tupla della relazione risultato rappresenta un gruppo di tuple della relazione su cui la query è eseguita
- nell'esempio visto i gruppi sono tre: uno per ogni valore di Dip#
- ad ognuno di questi gruppi è applicata la funzione MAX sulla colonna Stipendio
- più colonne possono essere usate per definire gruppi
- le funzioni di gruppo possono essere usate anche in presenza di join

SQL - Raggruppamento

- esempio: supponiamo di voler raggruppare gli impiegati sulla base del dipartimento e della mansione
- per ogni gruppo si vuole determinare 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 BY Nome_Dip, Mansione;
```


SQL - Raggruppamento

- 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

SQL - Raggruppamento

Clausola HAVING

- è possibile specificare condizioni di ricerca su gruppi di tuple, selezionando solo alcuni dei gruppi ottenuti dal partizionamento
- esempio: supponiamo di essere interessati solo ai gruppi che contengono almeno due impiegati

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


SQL - Raggruppamento

Clausola HAVING

- risultato

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

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

SQL – Raggruppamento e funzioni di gruppo

- Un modello di "esecuzione"
  1. si applica la condizione di ricerca specificata nella clausola WHERE a tutte le tuple della relazione oggetto della query
     la valutazione avviene tupla per tupla
  2. alle tuple ottenute al passo precedente, si applica il partizionamento specificato dalla clausola GROUP BY
  3. ad ogni gruppo di tuple ottenuto al passo precedente, si applica la condizione di ricerca specificata dalla clausola HAVING
  4. i gruppi ottenuti al passo precedente sono i gruppi di tuple che verificano la query
     per tali gruppi, vengono calcolate le funzioni di gruppo specificate nella clausola di proiezione della query
     i valori restituiti da tali funzioni costituiscono il risultato della query

SQL - Valori nulli

- SQL usa una logica a tre valori per valutare il valore di verità di una condizione di ricerca (clausola where)
  True (T), False (F), Unknown (?)
- un predicato semplice valutato su un attributo a valore nullo dà come risultato della valutazione ?
- il valore di verità di un predicato complesso viene calcolato in base alle seguenti tabelle di verità
- una tupla per cui il valore di verità è ? non viene restituita dalla query

SQL - Valori nulli

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
?	?

SQL - Valori nulli

- esempio: R

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

- SELECT * FROM R WHERE A=a OR B=b;
  il valore di verità della condizione per ogni tupla è il seguente:
  t1 T OR ? --> T
  t2 F OR T --> T
  t3 F OR ? --> ?
  le tuple che verificano la query sono t1 e t2

SQL - Valori nulli

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

SQL - Valori nulli

- il predicato IS NULL applicato ad un dato attributo di una tupla restituisce True se la tupla ha valore nullo per l'attributo
- esempio:
  - SELECT * FROM R WHERE B IS NULL;
    restituisce le tuple t1 e t3
  - SELECT * FROM R WHERE B IS NULL AND C IS NULL;
    restituisce la tupla t3
  - SELECT * FROM R WHERE B IS NULL OR C IS NULL;
    restituisce le tuple t1 e t3

SQL - Valori nulli

- 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
- si noti che quindi le interrogazioni
  SELECT * FROM R WHERE B = B;
  SELECT * FROM R WHERE B = 'b' OR B <> 'b';
  SELECT * FROM R WHERE B IS NOT NULL;
  sono tutte equivalenti

SQL - Valori nulli

- nelle espressioni (ad es. aritmetiche) se un argomento è NULL allora il valore dell'intera espressione è NULL
- esempio: Stipendio * 12, Stipendio + Premio_P
- nelle funzioni di gruppo i valori nulli sono invece ignorati
- esempio: SUM(Premio_P)
- questo ha come conseguenza che, ad esempio,
  SUM(Colonna1 + Colonna2)
  può dare risultato diverso da
  SUM(Colonna1) + SUM(Colonna2)
- una funzione di gruppo può comunque restituire NULL se applicata a un insieme di valori vuoto o contenente il solo valore NULL

SQL - Valori nulli

- se e1 ed e2 sono NULL, e1=e2 non è vero (è ?)
- quindi due espressioni con valore nullo non sono uguali ma non sono distinte, cioè vengono considerate duplicati
  - SELECT DISTINCT: si ha al più un NULL nel risultato
  - GROUP BY: si ha al più un gruppo per il valore NULL
- esiste un predicato IS DISTINCT FROM che coincide con <> tranne che per il valore nullo, cioè se e1 ed e2 sono NULL:
  - e1 <> e2 restituisce UNKNOWN, ma
  - e1 IS DISTINCT FROM e2 restituisce FALSE

SQL - Sottointerrogazioni

- una delle ragioni che rendono SQL un linguaggio potente è la possibilità di esprimere interrogazioni più complesse in termini di interrogazioni più semplici, tramite il meccanismo delle subqueries (sottointerrogazioni)
- la clausola WHERE di una query (detta query esterna) può infatti contenere un'altra query (detta subquery)
- la subquery viene usata per determinare uno o più valori da usare come valori di confronto in un predicato della query esterna

SQL - Sottointerrogazioni

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

```
SELECT Nome 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 è

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

SQL - Sottointerrogazioni

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

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


SQL - Sottointerrogazioni

- risultato

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

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

SQL - Sottointerrogazioni

- negli esempi visti, le subqueries restituiscono un solo valore
- tali subqueries sono note come subqueries **scalari**
- se una subquery scalare restituisce più di una tupla si genera un errore a run-time
- una particolarità delle subqueries scalari è che se nessuna tupla verifica la subquery, viene restituito il valore NULL
- se si vuole invece utilizzare una subquery che restituisce più valori (**table** subquery) è necessario specificare come i valori restituiti devono essere usati nella clausola WHERE

SQL - Sottointerrogazioni

- a tale scopo vengono usati i *quantificatori* ANY ed ALL, che sono inseriti tra l'operatore di confronto e la subquery (esiste anche SOME, sinonimo di ANY)
- esempio: determinare lo stipendio, la mansione, il nome e il numero di dipartimento degli impiegati che guadagnano più di *almeno un* impiegato del dipartimento 30

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


SQL - Sottointerrogazioni

- la query è equivalente a

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

- risultato

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

SQL - Sottointerrogazioni

- se si usa il quantificatore ALL, la query restituisce gli impiegati il cui stipendio è 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 più di *tutti gli* impiegati del dipartimento 30

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


SQL - Sottointerrogazioni

- la query è equivalente a

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

- risultato

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

SQL - Sottointerrogazioni

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


SQL - Sottointerrogazioni

- esempio (2): 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
dirigente	Verdi
ingegnere	Dare
ingegnere	Milli

risultato (2)

Mansione	Nome
tecnico	Andrei
tecnico	Bianchi
tecnico	Turni
segretaria	Martini

SQL - Sottointerrogazioni

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

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


SQL - Sottointerrogazioni

- la clausola di WHERE di una query può contenere una qualsiasi combinazione di condizioni normali e condizioni con subqueries
- esempio: si vuole elencare gli impiegati con la stessa mansione di Gianni o con uno stipendio maggiore o uguale a quello di Fordi, ordinando 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;
```

SQL - Sottointerrogazioni

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

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


SQL - Sottointerrogazioni correlate

- negli esempi visti ogni subquery viene eseguita una volta per tutte ed il valore (o insieme di valori) è usato nella clausola WHERE della query esterna
- è 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 più dello stipendio medio del proprio dipartimento

SQL - Sottointerrogazioni correlate

- è 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);
```

- è 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 - Sottointerrogazioni correlate

- ogni volta che la query esterna considera una tupla candidata, deve invocare la subquery e "passare" il numero di dipartimento dell'impiegato in esame
- la subquery calcola quindi la media degli stipendi nel dipartimento che è stato passato e restituisce tale valore alla query esterna
- la query esterna può quindi confrontare lo stipendio dell'impiegato in esame con il valore restituito dalla subquery

SQL - Sottointerrogazioni correlate

- questo tipo di interrogazioni è chiamato correlato, perchè ogni esecuzione della subquery è correlata al valore di uno o più 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 è definito nella query esterna e riferito nella query interna

SQL - Sottointerrogazioni correlate

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

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

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

SQL - Sottointerrogazioni correlate

- due sono i concetti principali che sono alla base della correlazione:
  a) una subquery correlata fa riferimento ad un attributo selezionato dalla query esterna
  b) se una subquery seleziona tuple dalla stessa relazione riferita dalla query esterna, è necessario definire un alias per tale relazione della query esterna;

  la subquery deve usare l'alias per riferire i valori di attributo nelle tuple candidate nella query principale

SQL - Sottointerrogazioni correlate

- non è obbligatorio, anche se è preferibile per chiarezza, utilizzare un alias se la relazione della query esterna e quella della query interna sono diverse
- esempio:determinare nome e stipendio degli impiegati che sono dirigenti del loro dipartimento

```
SELECT Nome, Stipendio FROM Impiegati
WHERE Imp# = (SELECT Dirigente FROM Dipartimenti
              WHERE Impiegati.Dip# = Dip#)
```

è equivalente a

```
SELECT Nome, Stipendio FROM Impiegati X
WHERE Imp# = (SELECT Dirigente FROM Dipartimenti
              WHERE X.Dip# = Dip#)
```


SQL - Sottointerrogazioni correlate

- gli alias possono essere utilizzati anche in interrogazioni senza sottointerrogazioni
- in particolare sono utili quando si vuole fare riferimento a due diverse tuple della stessa relazione
- esempio:determinare nome e stipendio degli impiegati che guadagnano più del dirigente del loro dipartimento

```
SELECT X.Nome, X.Stipendio
FROM Impiegati X, Impiegati Y, Dipartimenti
WHERE X.Dip# = Dipartimenti.Dip# AND
      Dipartimenti.Dirigente = Y.Imp# AND
      X.Stipendio > Y.Stipendio
```


SQL - Sottointerrogazioni correlate

ALL ed ANY

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

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

nota: si può semplicemente esprimere con un join

SQL - Sottointerrogazioni correlate

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

SQL - Sottointerrogazioni correlate

EXISTS e NOT EXISTS

- 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);
```

SQL - Sottointerrogazioni correlate

EXISTS e NOT EXISTS

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

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


SQL - Sottointerrogazioni correlate

Divisione

- le subqueries correlate e l'operatore di NOT EXISTS permettono di esprimere l'operazione di divisione
- 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

SQL - Sottointerrogazioni correlate

Divisione

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

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


SQL - Sottointerrogazioni correlate

Divisione

- si noti che un modo alternativo di esprimere la divisione e' mediante l'uso di funzioni di gruppo: un dipartimento partecipa a tutti i progetti con budget maggiore di 50,000 se il numero di progetti distinti con budget maggiore di 50,000 cui partecipa coincide con il numero totale di progetti con budget maggiore di 50,000

```
SELECT Nome_Dip
FROM Dipartimenti NATURAL JOIN Partecipanti NATURAL JOIN
   Progetti
WHERE Budget > 50,000
GROUP BY Dip#
HAVING COUNT(DISTINCT Prog#) = (SELECT COUNT(*)
                                FROM Progetti
                                WHERE Budget > 50,000);
```


SQL - Sottointerrogazioni correlate

UNIQUE

- il predicato UNIQUE(sq) restituisce il valore Booleano True se la subquery restituisce tutte tuple distinte (in particolare se non restituisce alcuna tupla o una tupla sola)
- restituisce il valore Booleano False se nel risultato della subquery ci sono dei duplicati
- nota: la valutazione di predicati con questo operatore non restituisce mai il valore Unknown

SQL - Sottointerrogazioni correlate

UNIQUE

- esempio: si vogliono determinare i dipartimenti in cui non esistono due impiegati con la stessa mansione

```
SELECT Dip# FROM Impiegati X
WHERE UNIQUE (SELECT Mansione
              FROM Impiegati
              WHERE Dip# = X.Dip#);
```

SQL - Operazioni insiemistiche

Union

- una interrogazione o sottointerrogazione può essere costituita da una o più interrogazioni connesse dall'operatore UNION
- l'operatore UNION restituisce tutte le tuple distinte restituite da almeno una delle sottointerrogazioni a cui è applicata
- esempio: si considerino le seguenti relazioni con lo stesso schema della relazione Impiegati
  - Impiegati_Tempo_Parziale
  - Impiegati_In_Congedo

SQL - Operazioni insiemistiche

Union

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

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


SQL - Operazioni insiemistiche

Union

- l'operatore UNION impone alcune restrizioni sulle interrogazioni si cui opera
- le interrogazioni devono restituire lo stesso numero di colonne, e le colonne corrispondenti devono avere lo stesso dominio (non è richiesto che abbiano la stessa lunghezza) o domini compatibili
- se non viene specificato niente la corrispondenza si basa sulla posizione delle colonne, indipendemente dal loro nome

SQL - Operazioni insiemistiche

Union

- se si usa invece la keyword CORRESPONDING dopo UNION allora il match tra colonne non avviene per posizione ma per nome delle colonne
- è anche possibile specificare esplicitamente le colonne da mettere in corrispondenza per nome con CORRESPONDING BY(ListaNomiColonne)
- 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

SQL - Operazioni insiemistiche

Union

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

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


SQL - Operazioni insiemistiche

Union

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

SQL - Operazioni insiemistiche

Intersect ed Except

- gli operatori INTERSECT ed EXCEPT (o MINUS) eseguono l'intersezione e la differenza
- per questi operatori valgono le stesse condizioni di applicabilità 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;
```


SQL - Operazioni insiemistiche

Intersect ed Except

- 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;
```

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

SQL - Operazioni insiemistiche

Intersect ed Except

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

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

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

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


SQL - DML Inserzione

- il comando di inserzione ha il seguente formato

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

  dove:
  - R è il nome della relazione su cui si esegue l'inserzione
  - C_1,C_2,....,C_n è 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

SQL - DML Inserzione

- e_1,e_2,....,e_n è 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) è assegnato alla colonna C_i
- sq è 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 più 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 espressione) i-esima contenuta nella clausola di proiezione di sq

SQL - DML Inserzione

- esempio di inserzione con valori espliciti
  - si vuole inserire un nuovo dipartimento; il numero del dipartimento è 40, il nome è Edilizia Industriale; la divisione è D2; il dirigente è Blacchi (Imp# 7698); l'ufficio è il numero 6100

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

SQL - DML Inserzione

- esempio di inserzione con tuple i cui valori sono ottenuti tramite una subquery
  - si vuole creare una relazione Promozioni, che 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 è superiore al 25% del loro stipendio; le informazioni sugli ingegneri devono essere estratte dalla relazione Impiegati

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

  - In questo caso gli ingegneri inseriti sono Rossi ed Adami

SQL - DML Cancellazione

- il comando di cancellazione ha il seguente formato

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

dove:

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

SQL - DML Modifica

- il comando di UPDATE ha il seguente formato

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

dove:

- R è il nome della relazione su cui si esegue la modifica
- il nome della relazione può avere associato un alias se è necessario riferire a tuple di tale relazione in una qualche sottointerrogazione presente in F

SQL - DML Modifica

- $C_i = \{e_i \mid NULL\}$ (i=1,...,n) è un'espressione di assegnamento che specifica che alla colonna C_i deve essere assegnato il valore dell'espressione e_i
- tale espressione può essere una costante, oppure un'espressione aritmetica o di stringa, spesso funzione dei valori correnti delle tuple da modificare, oppure una subquery
- alternativamente si può specificare che alla colonna sia assegnato il valore nullo
- F è la clausola di qualificazione che specifica le tuple da modificare
- se non è specificata alcuna clausola di qualificazione, vengono modificate tutte le tuple

SQL - DML Modifica

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

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

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

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


SQL - DML Modifica

- nel comando di Update le subqueries possono essere usate per
  - (a) determinare le tuple da modificare
  - (b) determinare i nuovi valori da assegnare alle tuple
- esempio (a): si consideri la relazione Bonus e si supponga di voler aumentare del 5% lo stipendio di tutti gli impiegati presenti in tale relazione

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

SQL - DML Modifica

- 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';
```

- le modifiche in SQL sono eseguite in modo set-oriented: la clausola WHERE e il valore dell'espressione SET vengono valutati un'unica volta, poi gli aggiornamenti vengono effettuati su tutte le tuple "contemporaneamente"

SQL - DML Valori nulli

- assegnamento di valori nulli nel comando di Update
- si usa la parola chiave NULL (da non confondere con la stringa 'NULL')

```
UPDATE R
SET B=NULL WHERE C=C2;
```

- risultato

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

SQL - Vincoli CHECK su attributi

- Alla specifica della colonna viene affiancata la parola chiave CHECK seguita da una condizione, cioè un predicato o una combinazione booleana di predicati (componente WHERE di una query SQL)
- esempio

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

- se la valutazione del predicato di un constraint è ? Il constraint non è violato
- tale condizione può contenere sottointerrogazioni che fanno riferimento ad altre tabelle, ma il vincolo viene controllato solo quando viene modificato il valore dell'attributo a cui è associato

SQL - Vincoli CHECK su attributi

- esempio

```
Stipendio Decimal (7,2) CHECK (Stipendio <=
          (SELECT MAX(Stipendio)
          FROM Impiegato
          WHERE Mansione = 'dirigente'))
```

è un vincolo CHECK corretto,ma viene controllato solo sulla tupla che sto aggiornando ⇒ se diminuisco lo stipendio di un dirigente posso trovarmi in uno stato in cui un impiegato guadagna più di ogni dirigente

- i vincoli sui domini sono del tutto analoghi

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


SQL - Vincoli CHECK su tuple

- Alla definizione di una tabella viene aggiunta la parola chiave CHECK seguita da una condizione, cioè un predicato o una combinazione booleana di predicati (componente WHERE di una query SQL)
- esempio

```
CHECK (Stipendio > PremioP)
```

- tale condizione può contenere sottointerrogazioni che fanno riferimento ad altre tabelle, ma il vincolo viene controllato solo quando viene modificato il valore di una tupla della tabella a cui è associato

SQL - Asserzioni

- Sono elementi dello schema, servono per scrivere vincoli che coinvolgono più tuple o più tabelle
- sintassi:

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

- la condizione è un predicato o una combinazione booleana di predicati (componente WHERE di una query SQL)

SQL - Asserzioni

- esempio

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

- non tutti i DBMS prevedono tutti i tipi di vincoli, in particolare le asserzioni non sono generalmente supportate
- quasi tutti i DBMS si limitano a gestire i vincoli che possono essere verificati esaminando una sola tupla
- motivazione: efficienza della valutazione

SQL - Constraint

- È possibile assegnare un nome ai vincoli, facendo seguire la specifica del vincolo dalla parola chiave CONSTRAINT e dal nome
- esempi:
  - Imp# Char(6) CONSTRAINT ChiaveImp PRIMARY KEY
  - CONSTRAINT StipOk CHECK (Stipendio > PremioP)
- specificare un nome per i vincoli è utile per potervisi poi riferire (ad esempio per modificarli)

SQL - Constraint

- ogni vincolo ha associato un descrittore costituito da
  - nome (se non specificato è 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 {ListaNomiConstraint|ALL}
    {DEFERRED|IMMEDIATE}
```


SQL - Constraint

- esempio

```
CREATE TABLE Esame
(Sno      Char(6) REFERENCES Studente
                  ON DELETE CASCADE,
 Cno      Char(6) REFERENCES Corso,
 Voto     Integer NOT NULL,
 PRIMARY KEY (Sno,Cno),
 CONSTRAINT Voto-constr CHECK Voto ≥ 18 AND Voto ≤ 30);
```


SQL - Constraint

- valutazione constraints:
  - (a) un constraint è violato da una tupla se la condizione valutata sulla tupla dà valore False
  - (b) un constraint la cui valutazione su una tupla dà valore True o Unknown (a causa di valori nulli) non è 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 di aggiornamenti

  il programma tuttavia continua la sua esecuzione

SQL - Vincoli di integrità

- È inoltre possibile specificare se il controllo del vincolo debba essere effettuato non appena si esegue un'operazione che ne può causare la violazione (NON DEFERRABLE) o se possa essere rimandato alla fine della transazione (DEFERRABLE)
- per i vincoli differibili si può specificare un check time iniziale INITIALLY DEFERRED (default) o INITIALLY IMMEDIATE
- i vincoli non possono contenere condizioni la cui valutazione può dare risultati differenti a seconda di quando è valutata (es. riferimenti al tempo di sistema)

SQL - Viste

- in SQL è possibile definire viste alternative degli stessi dati
- una vista (**view**) è una relazione virtuale (simile ad una window) attraverso cui è possibile vedere i dati memorizzati nelle relazioni reali (dette **di base**)
- una vista non contiene tuple, ma può essere usata quasi a tutti gli effetti come una relazione di base
- una vista è definita da una interrogazione su una o più relazioni di base o altre viste
- una vista è materializzata eseguendo l'interrogazione che la definisce

SQL - Viste

- il meccanismo delle viste è utile per
  - semplificare l'accesso ai dati
  - fornire indipendenza logica
  - garantire la privatezza dei dati
- il comando di creazione di viste ha il seguente formato

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

dove:

SQL - Viste

- V è il nome della vista che viene creata; tale nome deve essere unico rispetto a tutti i nomi di relazioni e di viste definite dallo stesso utente che definisce V
- Q è l'*interrogazione di definizione* della vista
  una vista ha lo stesso numero di colonne pari alle colonne (di base o virtuali) specificate nella clausola di proiezione di Q
- ListaNomiColonne è una lista di nomi da assegnare alle colonne della vista; tale specifica non è obbligatoria, tranne nel caso in cui l'interrogazione contenga nella clausola di proiezione funzioni di gruppo e/o espressioni

SQL - Viste

- esempio: si vuole creare una vista costituita da un sottoinsieme delle tuple della relazione Impiegati; più 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

SQL - Viste

- su una vista si possono eseguire (con alcune importanti restrizioni) sia interrogazioni che modifiche
- esempio: selezionare le tuple della vista Imp10

```
SELECT * FROM Imp10;
```

- risultato:

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

SQL - Viste

Uso di join

- può essere facile per alcuni utenti lavorare con una sola relazione piuttosto che eseguire join tra relazioni diverse
- esempio: si vuole creare una vista Personale che contiene le colonne Nome e Mansione della relazione Impiegati e il nome del progetto a cui ogni impiegato lavora

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

SQL - Viste

Uso di espressioni e funzioni

- è possibile definire viste tramite interrogazioni che contengono espressioni o funzioni
- queste espressioni appaiono del tutto simili alle altre colonne della vista, ma il loro valore è calcolato dalle relazioni di base ogni volta che la vista è materializzata
- tali colonne sono spesso chiamate colonne virtuali; è obbligatorio specificare un nome nella vista per tali colonne
- esempio: si vuole definire una vista che contenga lo stipendio annuale degli impiegati

SQL - Viste

Uso di espressioni e funzioni

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

- se si esegue l'interrogazione

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

si ottiene il seguente risultato

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

SQL - Viste

- è possibile definire viste tramite interrogazioni che contengono funzioni di gruppo e clausole di GROUP BY
- esempio: si vuole definire una vista che calcoli per ogni dipartimento alcune statistiche riguardanti lo stipendio degli impiegati

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


SQL - Viste

- se si esegue l'interrogazione

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

si ottiene il seguente risultato

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

- una volta creata, una vista può essere considerata (quasi) una relazione di base
- in realtà si hanno alcune restrizioni sia sulle interrogazioni che sulle modifiche che si possono eseguire sulle viste

SQL - Viste - interrogazioni

- importante restrizione:
  - non è possibile l'uso di funzioni di gruppo su colonne di viste definite tramite funzioni di gruppo

  (alcuni DBMS in realtà lo permettono)
- quando si specifica una interrogazione su una vista, il sistema sostituisce nell'interrogazione la vista con la sua definizione
- esempio: si consideri la vista Personale e la query

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


SQL - Viste - interrogazioni

- la query che si ottiene dopo la composizione è la seguente

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

- nel caso di funzioni di gruppo applicate a colonne di viste definite tramite funzioni di gruppo questa composizione non sarebbe possibile (non si possono applicare funzioni di gruppo a funzioni di gruppo)

SQL - Viste - modifiche

Problema 1: inserimento

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

- lo schema di V3 è
  (Imp#, Nome, Mansione, Data_A, Dip#)
- supponiamo di inserire nella view la seguente tupla
  (8001, 'Smith', 'Tecnico', '13-Dic-91', 20)
- questa insert viene trasformata in una operazione di insert nella relazione Impiegati

SQL - Viste - modifiche

Problema 1: inserimento

- il problema è quale valore assegnare agli attributi Stipendio e Premio_P
- se per gli attributi è stato specificato un valore di default può essere assegnato tale valore
- altrimenti una soluzione è assegnare il valore nullo
- questa soluzione crea problemi se il valore di Stipendio e Premio_P deve essere diverso dal valore nullo

SQL - Viste - modifiche

Problema 2: cancellazione

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

- lo schema di V4 è (Nome, Ufficio)
- supponiamo di eseguire la seguente cancellazione

```
DELETE FROM V4 WHERE Nome = 'Rossi' AND Ufficio = 2100;
```

- questa cancellazione potrebbe essere tradotta come segue

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


SQL - Viste - modifiche

Problema 2: cancellazione

- stato della vista prima della cancellazione

```
SELECT * FROM V4;
```

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

SQL - Viste - modifiche

Problema 2: cancellazione

- stato della vista dopo la cancellazione

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

- la modifica precedente ha side-effect sulla vista
- se l'utente esegue una select sulla vista dopo la modifica tutte le tuple dello stesso ufficio di Rossi sono sparite dalla vista

SQL - Viste - modifiche

Problema 2: cancellazione

- il problema è che il mapping dell'operazione in termini di operazioni sulle relazioni di base può 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'ambiguità del mapping, in molti DBMS non vengono ammesse modifiche su viste definite tramite join

SQL - Viste - modifiche

Problema 3: modifica

```
CREATE VIEW V5 (Imp#, StipendioTot) AS
SELECT Imp#, Stipendio+Premio_P
FROM Impiegati;
```

- supponiamo di eseguire la seguente modifica

```
UPDATE V5 SET StipendioTot = StipendioTot * 1.2
WHERE Imp# =7782;
```

- questa modifica può essere ottenuta in diversi modi: aumentando Stipendio, aumentando Premio_P, aumentando entrambi
- in generale bisognerebbe conoscere la funzione inversa di quella che definisce StipendioTot

SQL - Viste - modifiche

- l'esecuzione di una operazione di modifica su una vista è propagata alla relazione su cui la vista è definita
- valgono le seguenti restrizioni:
  1) è possibile eseguire l'operazione di DELETE se l'interrogazione di definizione della vista soddisfa le seguenti condizioni:
     - è su una sola relazione
     - non contiene la clausola GROUP BY, la clausola DISTINCT, o una funzione di gruppo

SQL - Viste - modifiche

2) è possibile eseguire l'operazione di UPDATE se l'interrogazione di definizione della vista soddisfa le due condizioni precedenti ed inoltre:
   - la colonna modificata non è definita da un'espressione

3) è possibile eseguire l'operazione di INSERT se l'interrogazione di definizione della vista soddisfa le tre condizioni precedenti ed inoltre:
   - qualsiasi colonna per cui non sia specificato un valore di default e valga il vincolo NOT NULL sia presente nella vista

SQL - Viste - check option

- una vista essendo definita da una interrogazione contiene condizioni sul contenuto delle tuple
- solo le tuple che verificano una certa condizione sono restituite dalla vista
- un problema è cosa succede se in una vista in cui si possono eseguire inserimenti, si inserisce una tupla che non verifica la condizione specificata dalla query nella vista
- esempio: si consideri la seguente vista

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


SQL - Viste - check option

- supponiamo di inserire nella vista la seguente tupla

  (200, Haas, 2000)

  la condizione specificata nell'interrogazione non è verificata dalla nuova tupla; pertanto la tupla viene inserita ma non è ritrovata da un'interrogazione sulla vista
- per assicurare che le tuple inserite tramite una vista (o modificate tramite una vista) siano accettate solo se verificano la condizione nell'interrogazione della vista, si usa la CHECK OPTION

SQL - Viste - check option

- il formato della definizione di una vista è pertanto il seguente:

```
CREATE VIEW NomeVista [(ListaNomiColonne)]
AS Query
WITH [LOCAL|CASCADED] CHECK OPTION]
```

- la differenza tra LOCAL e CASCADED è rilevante nei casi in cui una vista è definita in termini di un'altra vista

SQL - Viste - check option

- esempio: se la vista precedente è definita come segue

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

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

SQL - Viste - check option

- sia V1 una vista definita in termini di un'altra vista V2
  - se V1 è definita WITH LOCAL CHECK OPTION, le inserzioni eseguite su V1 devono verificare
    (1) la definizione di V1
    (2) la definizione di V2 solo se V2 è a sua volta definita con check option
  - se V1 è definita WITH CASCADED CHECK OPTION, le inserzioni eseguite su V1 devono verificare
    (1) la definizione di V1
    (2) la definizione di V2 (indipendentemente dal fatto che V2 sia definita con check option o no)

SQL - Viste - modifiche di schema

- formato del comando di cancellazione
  DROP VIEW V;
  dove V è il nome della vista da cancellare
- formato del comando di renaming
  RENAME V_v to V_n
- non è possibile, invece, modificare la definizione di una vista; l'unico modo è ridefinire la vista

SQL - Viste - modifiche di schema

- cosa succede ad una vista V quando una tabella (o vista) usata nella query di definizione della vista viene cancellata?
- nei comandi di DROP TABLE e DROP VIEW si può specificare l'opzione RESTRICT o CASCADE
  - se si specifica RESTRICT il comando ha effetto solo se la tabella/vista non è utilizzata nella definizione di altre viste (o altri elmenti dello schema, es. asserzioni)
  - se si specifica CASCADE il comando ha l'effetto di cancellare anche le viste (e altri elementi dello schema) la cui definizione si basa sulla tabella/vista da cancellare

SQL - Cataloghi

- il catalogo è un database di sistema che contiene informazioni (descrittori) riguardanti i vari oggetti che sono di interesse al sistema: tabelle, viste, constraint, indici, diritti di accesso
- i cataloghi stessi sono organizzati in relazioni
- esempi:
  - una tupla per ogni relazione o vista, con attributi:
    - nome
    - nome dell'utente creatore
    - tipo (tabella/vista)
    - numero di colonne

SQL - Cataloghi

  - una tupla per ogni colonna di ogni relazione o vista, con attributi:
    - nome della colonna
    - nome della tabella o vista a cui la colonna appartiene
    - posizione ordinale della colonna tra le colonne della stessa tabella o view
    - tipo della colonna
- nota: i cataloghi possono avere nomi diversi nei diversi DBMS

SQL - Cataloghi

- è possibile interrogare i cataloghi come se fossero delle relazioni ordinarie
- esempi:
  - trovare tutte le tabelle che hanno una colonna che comincia con S
  - trovare tutte le colonne della relazione Impiegati
  - determinare quante relazioni sono state create da Rossi
- in genere non è possibile modificare i cataloghi
- i cataloghi sono modificati automaticamente dal sistema a seguito dei comandi di DDL

SQL - Cataloghi

- alcune eccezioni sono rappresentate dalla possibilità per gli amministratori della base di dati di poter modificare direttamente informazioni sulle statistiche usate per l'ottimizzazione
- i cataloghi includono anche informazioni sulle relazioni stesse che li memorizzano
- 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)

SQL - Caratteristiche non viste

- Alcuni aspetti del modello dei dati di SQL:1999 che non sono stati trattati:
  - caratteristiche object-relational:
    - tipi collezione, tipi riga, tipi riferimento, tipi user-defined
    - ereditarietà
  - trigger
  - ricorsione in interrogazioni e viste
  - costrutti di tipo OLAP (analisi dei dati) nelle interrogazioni con funzioni di gruppo
  - stored procedures