SQL - TIPI DI DATO

I tipi di dato in SQL:1999 si suddividono in

- tipi predefiniti
- tipi strutturati
- tipi user-defined

Ci concentreremo sui tipi predefiniti (i tipi strutturati e user-defined vengono considerati nelle caratteristiche object-relational di SQL: 1999)
I tipi di dato predefiniti sono suddivisi in 5 categorie: tipi numerici, tipi binari, tipi carattere, tipi temporali e tipi booleani.

Tipi numerici

Si dividono in:

- Tipi numerici esatti: rappresentano valori interi e valori decimali in virgola fissa (es. 75, 4.5, -6.2)
- Tipi numerici approssimati: rappresentano valori numerici approssimati in virgola mobile (es. 1256E-4).

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. I valori di questo tipo sono usati per eventuali ottimizzazioni poiché richiedono minore spazio di memorizzazione. L'unico requisito è 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 è che il primo deve essere implementato esattamente con la precisione richiesta, mentre il secondo può avere una precisione maggiore.

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. Il formato è FLOAT (p).

Tipi binari

`BIT`	Rappresenta stringhe di bit di lunghezza massima predefinita. La specifica di questo tipo di dato ha la forma BIT(n) dove n è la lunghezza massima delle stringhe (se non viene specificata alcuna

	lunghezza, il default è 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 è la lunghezza massima delle stringhe. La differenza con il tipo BIT è 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. 01111 o 0x44).
BINARY LARGE OBJECT (BLOB)	Questo tipo di dato permette di definire sequenze di bit di elevate dimensioni.

Tipi carattere

CHARACTER	Questo tipo di dato (spesso abbreviato in CHAR) permette di definire stringhe di caratteri di lunghezza massima predefinita. La specifica di questo tipo di dato ha la forma CHAR(n) dove n è la lunghezza massima delle stringhe (se non viene specificata alcuna lunghezza, il default è 1).
CHARACTER VARYING	Questo tipo di dato (spesso abbreviato in VARCHAR) permette di definire stringhe di caratteri di lunghezza massima predefinita. La specifica di questo tipo di dato ha la forma VARCHAR(n) dove n è la lunghezza massima delle stringhe. La differenza con il tipo CHAR è che per questo tipo si alloca per ogni stringa la lunghezza massima predefinita, mentre per VARCHAR si usano strategie diverse.
CHARACTER LARGE OBJECT (CLOB)	Questo tipo di dato permette di definire sequenze di caratteri di elevate dimensioni (testo). È possibile associare al tipo carattere il CHARACTER SET di riferimento e la relativa COLLATION (ordine dei caratteri nel set). Per ognuno dei tipi carattere esiste inoltre la variante NATIONAL.

Tipi temporali

DATE	Rappresenta le date espresse come anno (4 cifre), mese (2 cifre comprese tra 1 e 12), giorno (2 cifre comprese tra 1 e 31 ed ulteriori restrizioni a seconda del mese).
TIME	Rappresenta i tempi espressi come ora (2 cifre), minuti (2 cifre) e secondi (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.

Esempio (DB2)

Tipo di dato	Descrizione
SMALLINT	Intero a 16 bit

INTEGER	Intero a 32 bit
DECIMAL (P,S)	Numero decimale con precisione p e scala s (default p=5, s=0). NUMERIC (p,s) è sinonimo di DECIMAL (p,s)
DOUBLE	Numero in virgola mobile a 64 bit. FLOAT e DOUBLE PRECISION sono sinonimi di DOUBLE.
CHAR(N)	Stringa di lunghezza n, dove n <= 254 (default n=1)
VARCHAR(N)	Stringa di lunghezza variabile con lunghezza massima pari ad n, dove n <= 4000
DATE	Consiste di anno, mese e giorno
TIME	Consiste di ora, minuti e secondi
TIMESTAMP	Consiste di anno, mese, giorno, ora, minuti, secondi, microsecondi

SQL - DDL

Creazione di tabelle

La creazione avviene tramite il comando CREATE TABLE.

Sintassi

```
CREATE TABLE NOME-TABELLA(
   SPEC_COL1,
   …
   SPEC_COLN,
   VINCOLO_TUPLA1,
   …
   VINCOLO_ TUPLAN
);
```

dove:

- nome-tabella è il nome della relazione che viene creata;
- spec_col$_i$ (1=1…n) è una specifica di colonna il cui formato è il seguente:

```
NOMECOLONNAI DOMINIOI [DEFAULT VALORE_DEFAULT]
   [VINCOLO_ATTRIBUTO1 … VINCOLO_ATTRIBUTON]
```

- vincolo_tupla$_i$ è la specifica di un vincolo per le tuple della relazione. Ugualmente, vincolo_attributo$_i$ è la specifica di un vincolo relativo ai valori di un singolo attributo della tabella. I vincoli verranno discussi più avanti.

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)
);
```

Domini

È possibile definire domini ed utilizzarli nella definizione di tabelle.
Un dominio è un nuovo nome per un tipo di dato cui si possono associare altre informazioni (valori di default e/o vincoli sui valori).

Sintassi

```
CREATE DOMAIN NOME-DOMINIO AS DESCRIZIONE-TIPO
   [DEFAULT VALOREDEFAULT] [CHECK (VINCOLO_DOMINIO)];
```

Esempio

```
CREATE DOMAIN DOMINIOMANSIONE AS CHAR(10) DEFAULT 'IMPIEGATO';
```

```
CREATE TABLE IMPIEGATI (
   IMP   #DECIMAL(4),
   NOME  CHAR(20),
   MANSIONE    DOMINIOMANSIONE,
   DATA_A      DATE,
   STIPENDIO   DECIMAL(7,2),
   PREMIO_P    DECIMAL(7,2) DEFAULT 0,
   DIP#  DECIMAL(2));
```

Vincoli d'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 su attributo o su tupla (`CHECK`)
- Asserzioni (vincoli `CHECK` su più tuple o tabelle)
- Vincoli sui valori di un dominio (`CHECK`)

È possibile specificare se il controllo del vincolo debba essere compiuto 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 momento in cui sono esaminati (es. riferimenti al tempo di sistema).

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 (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`.
Se la chiave è formata da un solo attributo è sufficiente far seguire la specifica dell'attributo da `UNIQUE` o `PRIMARY KEY` (*vincolo su attributo*); alternativamente si può far seguire la definizione della tabella da `PRIMARY(LISTANOMICOLONNE) KEY` o `UNIQUE(LISTANOMICOLONNE)` (*vincolo su tupla*).

Esempi

```
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));
```

Valori NULL

Per indicare che un colonna non può assumere valore nullo (che è il default per tutti gli attributi di una tupla, se non specificato diversamente tramite la parola chiave `DEFAULT`) è sufficiente includere il vincolo `NOT NULL` nella specifica della colonna. Le colonne dichiarate `PRIMARY KEY` non possono assumere valori nulli.

Esempio

```
CREATE TABLE IMPIEGATI  (
    IMP#  DECIMAL(4) PRIMARY KEY,
    CODICE_FISCALE    CHAR(16) UNIQUE,
    NOME  CHAR(20) NOT NULL,
    MANSIONE    DOMINIOMANSIONE,
    DATA_A      DATE,
    STIPENDIO   DECIMAL(7,2) NOT NULL,
    PREMIO_P    DECIMAL(7,2),
    DIP#  DECIMAL(2) NOT NULL);
```

Chiavi esterne

Una chiave esterna specifica che i valori di un particolare insieme di attributi (chiave esterna) di una tupla devono necessariamente corrispondere a quelli presenti in un corrispondente insieme di attributi che sono una chiave per le tuple di un'altra relazione. Una relazione può avere zero o più chiavi esterne.
La specifica di chiavi esterne avviene mediante la clausola `FOREIGN KEY` del comando `CREATE TABLE`.

Sintassi

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

dove listaNomiColonne è un sottoinsieme delle colonne della tabella che corrisponde ad una chiave (UNIQUE o PRIMARY) della tabella riferita. Non è 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 può far semplicemente seguire la specifica dell'attributo da REFERENCES NomeTabellaRiferita (*vincolo su attributo*).

Tipo di Match

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

il default è MATCH SIMPLE.

Azioni

Le clausole opzionali ON DELETE e ON UPDATE indicano al DBMS cosa fare nel caso una tupla della tabella riferita cui fa riferimento una tupla della tabella referente venga cancellata o modificata.

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

- 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. In altre parole, l'operazione di cancellazione viene respinta se la tupla da cancellare è in relazione con qualche tupla della tabella referente.
- 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'integrità referenziale.
- 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.

Le opzioni riguardanti le azioni da eseguire nel caso di modifica della chiave della tupla riferita tramite chiave esterna hanno lo stesso significato delle opzioni viste per la cancellazione, con l'eccezione di `CASCADE`, che ha l'effetto di assegnare alla chiave esterna il nuovo valore della chiave della tupla riferita.

Il default è `NO ACTION` sia per la cancellazione sia per la modifica.
L'ordine in cui vengono considerate le varie opzioni (nel caso di più riferimenti) è `RESTRICT`, `CASCADE`, `SET NULL`, `SET DEFAULT`, `NO ACTION`.

Nel caso di inserimento o modifica nella tabella referente non è possibile specificare alcuna azione.

Esempi

```
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) REFERENCES DOCENTE
         ON DELETE RESTRICT
         ON UPDATE CASCADE,
   SNO   CHAR(6),
   PRIMARY KEY (SNO),
   FOREIGN KEY (SNO) REFERENCES STUDENTE
         ON DELETE CASCADE
         ON UPDATE CASCADE
);
```

Vincoli CHECK

Quando si definisce una tabella è possibile aggiungere alla specifica delle colonne, delle chiavi e delle chiavi esterne la parola chiave `CHECK` seguita da una condizione, cioè un

predicato o una combinazione booleana di predicati, comprendenti anche sottointerrogazioini che fanno riferimento ad altre tabelle (componente `WHERE` di una query SQL).
Il vincolo viene controllato solo quando viene inserita o modificata una tupla nella.

Esempio

```
CREATE TABLE IMPIEGATO (
…
STIPENDIO DECIMAL(7 ,2),
CHECK (STIPENDIO > PREMIOP)
);
```

```
CREATE TABLE IMPIEGATO (
…
STIPENDIO DECIMAL(7 ,2),
CHECK (STIPENDIO < (SELECT MAX(STIPENDIO)
   FROM IMPIEGATO WHERE MANSIONE = 'DIRIGENTE'))
);
```

```
CREATE TABLE IMPIEGATO (
…
MANSIONE CHAR(10) CHECK (MANSIONE IN
   ('DIRIGENTE', 'INGEGNERE', 'TECNICO', 'SEGRETARIA'))
);
```

I vincoli `CHECK` possono essere anche applicati nella definizione di un dominio (la parola chiave `VALUE` indica i valori possibili nel dominio):

```
CREATE DOMAIN DOMINIOMANSIONE AS CHAR(10)
   CHECK (VALUE IN ('DIRIGENTE', 'INGEGNERE', 'TECNICO',
'SEGRETARIA'))
```

Attribuire un nome ai vincoli

Ogni vincolo ha associato un descrittore costituito da

- nome (se non specificato assegnato dal sistema)
- differibile o meno
- checking time iniziale

È possibile assegnare un nome ai vincoli facendo precedere la specifica del vincolo dalla parola chiave `CONSTRAINT` e dal nome. Specificare un nome per i vincoli è utile per potervisi poi riferire (ad esempio per modificarli).

Esempi

```
Imp# Char(6) CONSTRAINT Chiave PRIMARY KEY
```

```
CONSTRAINT StipOK CHECK(Stipendio > PremioP)
```

```
CREATE TABLE Esame (
   Sno  Char(6) NOT NULL,
   Cno  Char(6) NOT NULL,
   Cnome Varchar (20) NOT NULL,
```

```
   Voto Integer NOT NULL,
   PRIMARY KEY (Sno,Cno),
   CONSTRAINT Rel_Stud FOREIGN KEY (Sno) REFERENCES Studente
        ON DELETE CASCADE,
   CONSTRAINT Cname-constr
        CHECK Cname IN
        ('Documentazione Automatica','SEI I', 'SEI II'),
   CONSTRAINT Voto-constr
        CHECK Voto > 18 AND Voto < 30);
```

Valutazione dei vincoli

- Un vincolo su una tupla è violato se la condizione valutata sulla tupla restituisce valore FALSE.
- Un vincolo la cui valutazione su una data tupla restituisce valore TRUE o Unknown (a causa di valori nulli) non è violato.
- Durante l'inserimento o la modifica di un insieme di tuple, la violazione di un vincolo per una delle tuple causa la non esecuzione dell'intero insieme.

Asserzioni

E' possibile specificare vincoli di integrità che non sono in relazione con una particolare tabella, ma coinvolgono più in generale lo stato del database. Il sistema segnalerà qualsiasi operazione che renda falsa un'asserzione, e tale operazione non verrà accettata.

```
CREATE ASSERTION NOME_ASSERZIONE CHECK (CLAUSOLA_WHERE)
```

Dove CLAUSOLA_WHERE può essere specificata esattamente come nel caso dei vincoli CHECK su tuple visti in precedenza.

Esempio

```
CREATE ASSERTION STIOPK CHECK (
   NOT EXISTS (
        SELECT *
        FROM IMPIEGATI I, IMPIEGATI J
        WHERE I.LIVELLO > J.LIVELLO AND I.STIPENDIO < J.STIPENDIO
   )
);
```

Cancellazione e modifiche su tabelle

Cancellare una tabella

```
DROP TABLE R
```

dove R è il nome della relazione da cancellare.
Ad esempio: DROP TABLE IMPIEGATI;

Rinominare una tabella

```
RENAME RV TO RN
```

dove Rv e Rn sono, rispettivamente, il vecchio e nuovo nome della relazione.
Ad esempio: RENAME IMPIEGATI TO IMP;

Modificare le colonne di una tabella

```
ALTER TABLE R ADD COLUMN spec_col
```

aggiunge una nuova colonna ad una relazione.
Ad esempio: `ALTER TABLE IMPIEGATI ADD COLUMN PROG# DECIMAL(3);`

```
ALTER TABLE R DROP COLUMN nome_col
```

rimuove una colonna da una relazione.
Ad esempio:`ALTER TABLE IMPIEGATI DROP COLUMN STIPENDIO`

```
ALTER TABLE R ALTER COLUMN NOME_COL SET DEFAULT VALOREDEFAULT
```

modifica il valore di default di un attributo.
Ad esempio: `ALTER TABLE IMPIEGATI ALTER COLUMN PROG# SET DEFAULT 0;`

```
ALTER TABLE R ALTER COLUMN NOME_COL DROP DEFAULT
```

elimina il valore di default di un attributo.
Ad esempio: `ALTER TABLE IMPIEGATI ALTER COLUMN PROG# DROP DEFAULT;`

Modificare i vincoli di una tabella

```
ALTER TABLE R DROP CONSTRAINT C {RESTRICT | CASCADE}
```

cancella il vincolo chiamato C dalla tabella R

```
ALTER TABLE R ADD CONSTRAINT C VINCOLO_TUPLA
ALTER TABLE R ADD VINCOLO_TUPLA
```

crea un vincolo chiamato C o anonimo nella tabella R. VINCOLO_tupla può essere uno qualsiasi dei vincoli visti in precedenza.
Ad esempio: `ALTER TABLE IMPIEGATI ADD UNIQUE(NOME)`
oppure: `ALTER TABLE IMPIEGATI ADD COSTRAINT STIPOK CHECK(STIPENDIO < PREMIO_P)`

```
SET CONSTRAINTS (LISTANOMICONSTRAINTS|ALL} {IMMEDIATE|DEFERRED }
```

modifica il checking time (solo per i vincoli `DEFERRABLE`).

Cancellazione e modifiche su domini e asserzioni

Cancellare un dominio

```
DROP DOMAIN D {RESTRICT | CASCADE}
```

rimuove un dominio

- Se viene specificato RESTRICT: il dominio viene rimosso solo se nessuna tabella lo utilizza
- Se viene specificato CASCADE: in ogni tabella che lo utilizza il nome del dominio viene sostituito dalla sua definizione (la modifica non influenza i dati presenti nella tabella)

Modificare un dominio

```
ALTER DOMAIN D SET DEFAULT VALORE_DEFAULT
```

modifica il valore di default di un dominio.

```
ALTER DOMAIN D DROP DEFAULT
```

elimina il valore di default di un dominio.

```
ALTER DOMAIN D DROP CONSTRAINT C
```
rimuove il vincolo C dal dominio D.

```
ALTER DOMAIN D ADD CONSTRAINT C
```
aggiunge il vincolo C un dominio D.

Cancellare un'asserzione

```
DROP ASSERTION A
```
rimuove l'asserzione A.

SQL - QL

Interrogazioni di base

```
SELECT LISTA_ATTRIBUTI
FROM LISTA_TABELLE
[WHERE CONDIZIONE]
```

più dettagliatamente:

```
SELECT ATTRIBUTO [[AS] [ALIAS]] {, ATTRIBUTO [[AS] [ALIAS] ] }
FROM TABELLA [[AS] [ALIAS]] {, TABELLA [[AS] [ALIAS] ] }
[WHERE CONDIZIONE]
```

Significato

La forma più comune di interrogazione in SQL ha la seguente struttura:

```
SELECT R_I1.C_1, R_I2.C_2…, R_IN.C_N
FROM R_1, R_2…, R_K
WHERE F
```

dove

- R_1, R_2…,R_K è una lista di nomi di relazioni;
- $R_{I1}.C_1$, $R_{I2}.C_2$…,$R_{IN}.C_N$ è una lista di colonne: la notazione R.C indica la colonna di nome C nella relazione R. Se una sola relazione nella lista di relazioni della clausola FROM ha una colonna di nome C, si può usare C invece di R.C;
- F è un predicato sulle tabelle R_1, …,R_K.

Il significato di una query SQL può essere espresso per mezzo dell'espressione algebrica $\pi(R_{I1}.C_1, R_{I2}.C_2\ldots,R_{IN}.C_N)(\sigma F (R_1 \times R_2\ldots \times R_K))$

Esempio

Siano date le seguenti definizioni di relazioni:

```
IMPIEGATO (NOME, COGNOME,UFFICIO,STIPENDIO,REPARTO)
REPARTO (NOME,INDIRIZZO,CITTÀ)
```

I dati inseriti nelle relazioni sono i seguenti:

Nome	Cognome	Reparto	Ufficio	Stipendio
Mario	Rossi	Amm.ne	10	45
Carlo	Bianchi	Prod.ne	20	36
Giuseppe	Verdi	Amm.ne.	20	40
Franco	Neri	Distr.ne	16	45
Carlo	Rossi	Direz.ne	14	80
Lorenzo	Lanzi	Direz.ne	7	73
Paola	Borroni	Amm.ne	75	40

Marco	Franco	Prod.ne	20	46

Nome	Indirizzo	Città
Amm.ne	via Tito Livio, 27	Milano
Prod.ne	p.le Lavater, 3	Torino
Distr.ne	via Segre 9	Roma
Direz.ne	via tito Livio, 27	Milano
Ricerca	via Morone, 6	Milano

Reperire i cognomi degli impiegati del reparto "Produzione"
*Nota: il simbolo * nella clausola di proiezione indica che tutte le colonne della relazione devono essere restituite.*

```
SELECT COGNOME
FROM IMPIEGATO
WHERE REPARTO = 'PROD.NE'
```

Cognome
Bianchi
Franco

Reperire i dati degli impiegati del reparto Amministrazione

```
SELECT *
FROM IMPIEGATO
WHERE REPARTO = 'AMM.NE'
```

Nome	Cognome	Reparto	Ufficio	Stipendio
Mario	Rossi	Amm.ne	10	45
Giuseppe	Verdi	Amm.ne	20	40
Paola	Borroni	Amm.ne	75	40

Reperire lo stipendio mensile degli impiegati di cognome Bianchi

```
SELECT STIPENDIO /12 AS STIPNDIOMENSILE
FROM IMPIEGATO
WHERE COGNOME = 'BIANCHI'
```

StipendioMensile
3,00

Reperire i cognomi degli impiegati del reparto Amministrazione o Produzione.

```
SELECT COGNOME
FROM IMPIEGATO
WHERE REPARTO='AMM.NE' OR REPARTO='PROD.NE'
```

Cognome
Rossi
Bianchi
Verdi
Borroni
Franco

Siano date le seguenti definizioni di relazioni:

```
IMPIEGATO (#IMP,NOME,MANSIONE,DATA_A,STIPENDIO,PREMIO_P,#DIP)
REPARTO (#DIP,NOMINATIVO,LOCALITÀ)
```

Lista di tutti i manager

```
SELECT *
FROM IMPIEGATI
WHERE MANSIONE = 'MANAGER'
```

Lista gli id di tutti gli impiegati

```
SELECT IMP#
FROM IMPIEGATI
```

Lista di tutti i reparti con numero di codice maggiore o uguale a 30

```
SELECT *
FROM REPARTI
WHERE #DIP >=30
```

Listare gli impiegati che hanno premio di produzione maggiore dello stipendio

```
SELECT *
FROM IMPIEGATI
WHERE PREMIO_P > STIPENDIO
```

Selezionare gli impiegati che hanno uno stipendio maggiore di 2000

```
SELECT * FROM IMPIEGATI
WHERE STIPENDIO>2000;
```

Imp#	Nome	Mansione	Data_A	Stipendio	Premio_P	Dip
7566	Rosi	dirigente	02-Apr-81	2975,00	?	20
7698	Biacchi	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

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

```
SELECT NOME, DIP#
FROM IMPIEGATI
WHERE STIPENDIO>2000 AND MANSIONE = 'INGEGNERE';
```

Nome	Dip#
Neri	10
Dare	10

Selezionare il numero di matricola degli impiegati che lavorano nel dipartimento 30 e sono ingegneri o tecnici

```
SELECT IMP#
FROM IMPIEGATI
WHERE DIP#=30 AND (MANSIONE = 'INGEGNERE' OR MANSIONE = 'TECNICO');
```

Imp#
7499
7521

7844
7900

Lista degli impiegati dei reparto 10 che guadagnano più di 2000

```
SELECT NOME, STIPENDIO, #DIP
FROM IMPIEGATI
WHERE STIPENDIO > 2000 AND #DIP=10
```

Operatori speciali per clausole WHERE

Condizioni su intervalli di valori

L'operatore `BETWEEN` permette di determinare le tuple che contengono in un particolare attributo valori in un intervallo dato.

Sintassi

```
C BETWEEN v1 AND v2 / C NOT BETWEEN v1 AND v2
```

Esempio

```
SELECT NOME,STIPENDIO
FROM IMPIEGATI
WHERE STIPENDIO BETWEEN 1100 AND 1400
```

Nome	Stipendio
Adami	1100,00
Muli	1300,00

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.

Sintassi

- (con lista valori) `C IN (v1, v2  ,vN) / C NOT IN (v1, v2  ,vN)`
- (con subquery) `C IN SQ / C NOT IN SQ`

Esempio

Tutti i dati dei dipartimenti 10 e 30

```
SELECT *
FROM DIPARTIMENTI
WHERE DIP# IN (10,30);
```

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

Lista degli impiegati che non sono né analisti né programmatori

```
SELECT NOME, STIPENDIO, #DIP, QUALIFICA
FROM IMPIEGATI
WHERE QUALIFICA NOT IN ('ANALISTA', 'PROGRAMMATORE')
```

Tutti i dati dei dipartimenti per cui lavorano degli analisti

```
SELECT *
FROM DIPARTIMENTI
WHERE DIP# IN (
   SELECT DIP#
   FROM IMPIEGATI
   WHERE MANSIONE = 'ANALISTA'
);
```

Confronto tra stringhe di caratteri

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

Sintassi

`C 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.

Esempio

Determinare tutti gli impiegati che hanno 'R' come terza lettera dei cognome

```
SELECT NOME
FROM IMPIEGATI
WHERE NOME LIKE '__R%'
```

Nome
Martini
Neri
Dare
Turni
Fordi
Verdi

Nota: I predicati formati con gli operatori BETWEEN, IN e LIKE possono essere connessi con AND ed OR nella specifica di condizioni complesse.

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).
E' possibile specificare un ordinamento diverso aggiungendo alla fine dell'interrogazione la clausola `ORDER BY`.

Sintassi

`ORDER BY COLONNA [ASC | DESC] {, COLONNA [ASC | DESC]}`

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;
```

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	Biacchi

Si vogliono elencare mansione, nome e stipendio di tutti gli impiegati ordinando le tuple in base alla mansione in ordine crescente, ed in base allo stipendio in ordine decrescente

```
SELECT MANSIONE, STIPENDIO, NOME
FROM IMPIEGATI
ORDER BY MANSIONE, STIPENDIO DESC;
```

Mansione	Stipendio	Nome
dirigente	3000,00	Verdi
dirigente	2975,00	Rosi
dirigente	2850,00	Biacchi
ingegnere	2450,00	Neri
ingegnere	2000,00	Dare
ingegnere	1950,00	Gianni
ingegnere	1600,00	Rossi
ingegnere	1300,00	Muli
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

Eliminazione dei duplicati

Supponiamo di voler ritrovare la lista di tutte le mansioni presenti nella relazione Impiegati

```
SELECT MANSIONE
FROM IMPIEGATI;
```

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;
```

Mansione
ingegnere
tecnico
dirigente
segretaria

Esempio

```
SELECT INDIRIZZO
FROM REPARTO
WHERE CITTÀ = 'MILANO'
```

Indirizzo
via Tito Livio, 27
via Tito Livio, 27
via Morone, 6

```
SELECT DISTINCT INDIRIZZO
FROM REPARTO
WHERE CITTÀ = 'MILANO'
```

Indirizzo
via Tito Livio, 27
via Morone, 6

Espressioni e funzioni aritmetiche

I predicati usati nelle interrogazioni possono coinvolgere, oltre a nomi di colonna, anche espressioni aritmetiche. Tali espressioni sono formulate applicando operatori aritmetici o funzioni ai valori delle colonne delle tuple.

Le espressioni aritmetiche possono anche comparire nella clausola di proiezione (`SELECT`) e nelle espressioni di assegnamento del comando `UPDATE`. Un'espressione aritmetica 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, sono solo *materializzate* come risultato delle interrogazioni.

Funzioni aritmetiche di base

Si tratta delle usuali funzioni aritmetiche: +, -, *, /, …

Esempio

Trovare il nome, lo stipendio e il premio di produzione di tutti gli ingegneri per cui la somma dello stipendio e dei premio di produzione è maggiore di 2000

```
SELECT NOME, STIPENDIO, PREMIO_P
FROM IMPIEGATI
WHERE MANSIONE = 'INGEGNERE' AND STIPENDIO+PREMIO_P> 2000;
```

Nome	Stipendio	Premio_P
Rossi	1600,00	500,00
Neri	2450,00	200,00
Dare	2000,00	300,00

Trovare il nome, lo stipendio, il premio di produzione, e la somma dello stipendio e dei premio di produzione di tutti gli ingegneri.

```
SELECT NOME, STIPENDIO, PREMIO_P, STIPENDIO+PREMIO_P
FROM IMPIEGATI
WHERE MANSIONE = 'INGEGNERE';
```

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

Nota: nel calcolo delle espressioni aritmetiche il valore nullo viene considerato uguale al valore zero.

Funzioni scalari

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

Funzioni

- `ABS(N)` calcola il valore assoluto del valore numerico n.
- `MOD(N,B)` calcola il resto intero della divisione di n per b.
- `SQRT(N)` calcola la radice quadrata.

Alcuni DBMS supportano anche funzioni trigonometriche e funzioni per il calcolo della parte intera superiore ed inferiore.

Esempio

```
SELECT ABS(B)
FROM R
WHERE MOD(A,5) > 3
```

Espressioni e Funzioni per Stringhe

Si applicano ai tipi carattere.

Funzioni

- `LENGTH(STR)`: calcola la lunghezza di un stringa.
- `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 è specificato) oppure fino all'ultimo carattere.
- `STR1 || STR2`: restituisce la concatenazione delle due stringhe str1 e str2.

Esempi

Restituisce un'unica stringa che contiene cognome, nome ed indirizzo separati da uno spazio bianco

```
SELECT COGNOME ||'' || NOME || '' || INDIRIZZO
FROM PERSONE
```

Restituisce i cognomi più lunghi di tre caratteri

```
SELECT DISTINCT COGNOME
FROM PERSONE
WHERE LENGTH(COGNOME) > 3
```

Funzioni per date e tempi

Funzioni di conversione

- `DATE(VAL), TIME(VAL), TIMESTAMP(VAL)`: convertono rispettivamente un valore scalare in una data, un tempo, un timestamp.
- `CHAR(DATA, [FORMATO])`: converte un valore di data/tempo in una stringa di caratteri; può inoltre ricevere una specifica di formato.
- `DAYS(VAL)`: converte una data in un intero che rappresenta il numero di giorni a partire dall'anno zero.

Esempi

```
DATE('6/20/1997')
CHAR(DATA_ASSUNZIONE, EUR)
DAYS('1/8/2001')
```

Funzioni per la determinazione dei valore di registri speciali

- `CURRENT_DATE`: rappresenta la data corrente
- `CURRENT_TIME`: rappresenta l'ora corrente
- `CURRENT_TIMESTAMP`: rappresenta il timestamp corrente

Espressioni aritmetiche con valori data/ora

Date e tempi possono essere usati in espressioni aritmetiche; in tali espressioni è possibile usare diverse *unità temporali* quali `YEAR[S], MONTH[S], DAY[S], HOUR[S], MINUTE[S], SECOND[S],` da posporre ai numeri.

Esempio

Si vuole avere un colloquio con tutti i nuovi impiegati dopo 90 giorni dalla loro assunzione. In particolare, per tutti gli impiegati dei dipartimento 10 per cui si deve ancora effettuare il

colloquio, si vogliono 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;
```

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

Operatori Aggregati

In algebra relazionale, le condizioni sono predicati valutati su singole tuple indipendentemente dalle altre. Spesso è invece necessario valutare proprietà che dipendono da insiemi di tuple. Ad esempio, il numero di impiegati del reparto Produzione non è una proprietà di una singola tupla.

Sintassi

- `COUNT (*|[DISTINCT|ALL] LISTAATTRIBUTI)`: conteggio attributi tuple.
- `SUM ([DISTINCT|ALL] EXPR)`: somma i valori di Expr nelle tuple selezionate.
- `MAX ([DISTINCT|ALL] EXPR)`: calcola il massimo tra i valori di Expr nelle tuple selezionate.
- `MIN ([DISTINCT|ALL] EXPR)`: calcola il minimo tra i valori di Expr nelle tuple selezionate.
- `AVG ([DISTINCT|ALL] EXPR)`: calcola la media tra i valori di Expr nelle tuple selezionate.

I modificatori `ALL` e `DISTINCT` possono essere specificati per richiedere che l'operatore aggregato sia applicato rispettivamente su tutti i valori o su tutti i valori diversi dell'espressione richiesta.
Per la funzione `COUNT`, l'argomento speciale * restituisce il numero di tuple selezionate. In SQL2, se `COUNT` ha come argomento il nome di una singola colonna, il modificatore `DISTINCT` è obbligatorio.

Le colonne della relazione-risultato ottenute dall'applicazione delle funzioni di gruppo sono colonne virtuali. Le funzioni di gruppo possono apparire in espressioni aritmetiche.

Esempi

Restituisce il numero di diversi valori dell'attributo Stipendio fra tutte le righe di Impiegato.

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

Restituisce il numero di righe che possiedono un valore non nullo per entrambi gli attributi Nome e Cognome.

```
SELECT COUNT (ALL NOME, COGNOME)
FROM IMPIEGATO
```

Determinare il numero di impiegati del reparto Produzione

```
SELECT COUNT(*)
FROM IMPIEGATO
WHERE REPARTO='PROD.NE'
```

Determinare la somma degli stipendi del reparto Amministrazione

```
SELECT SUM(STIPENDIO)
FROM IMPIEGATO
WHERE REPARTO='AMM.NE'
```

Determinare la somma degli stipendi e dei premi produzione del reparto Amministrazione

```
SELECT SUM(STIPENDIO) + SUM(PREMIO_P)
FROM IMPIEGATO
WHERE REPARTO='AMM.NE'
```

Determinare il massimo stipendio tra quelli degli Impiegati che lavorano in un reparto di Milano

```
SELECT MAX(STIPENDIO)
FROM IMPIEGATO, REPARTO
WHERE LMPIEGATO.REPARTO=REPARTO.NOME AND CITTÀ='MILANO'
```

Determinare lo stipendio massimo, minimo e medio tra gli impiegati

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

Determinare gli impiegati con il massimo stipendio

```
SELECT NOME
FROM IMPIEGATO
WHERE STIPENDIO = (
  SELECT MAX(STIPENDIO)
  FROM IMPIEGATO
)
```

Raggruppamento di tuple

Selezionare la somma degli stipendi degli impiegati in ciascun dipartimento

```
SELECT #DIP, SUM(STIPENDIO)
FROM IMPIEGATO;
```

E' una interrogazione ERRATA! Se è presente una funzione di gruppo nella clausola SELECT, SQL non accetta che vengano specificate altre colonne che non siano risultati di funzioni di gruppo. Per ottenere il risultato desiderato, dobbiamo prima usare un raggruppamento.

Sintassi

- `GROUP BY COLONNA {, COLONNA}`

Una funzione di gruppo permette di estrarre informazioni da gruppi di tuple di una relazione.

- Si PARTIZIONANO le tuple di una relazione in base al valore di una o più colonne della relazione.
- Si CALCOLANO funzioni di gruppo per ogni gruppo ottenuto dai partizionamento.

Se è presente un raggruppamento, una funzione di gruppo che ha come argomento una colonna si applica all'insieme di valori di questa colonna, estratti dalle tuple che appartengono **allo stesso gruppo**. I valori delle espressioni usate per il raggruppamento possono apparire anche nell'istruzione `SELECT`.

Esempi

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#
```

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 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.

Determinare la somma degli stipendi dei vari reparti

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

Dip#	SUM(Stipendio)
Amm.ne	125
Prod.ne	82
Distr.ne	45
Direz.ne	153

Determinare la somma degli stipendi dei vari reparti (di cui vogliamo conoscere il nome)

```
SELECT REPARTO.NOMINATIVO, SUM(IMPIEGATO.STIPENDIO)
FROM IMPIEGATO, REPARTO
WHERE REPARTO.DIP# = IMPIEGATO.DIP#
GROUP BY REPARTO.DIP#
```

E' una interrogazione ERRATA! Tutte le colonne non risultanti di una operazione di aggregazione presenti nella clausola `SELECT` devono essere anche presenti in quella `GROUP BY`!

```
SELECT REPARTO.NOMINATIVO, SUM(IMPIEGATO.STIPENDIO)
FROM IMPIEGATO, REPARTO
WHERE REPARTO.DIP# = IMPIEGATO.DIP#
GROUP BY REPARTO.NOMINATIVO
```

```
SELECT REPARTO.DIP#, REPARTO.NOMINATIVO, SUM(IMPIEGATO.STIPENDIO)
FROM IMPIEGATO, REPARTO
WHERE REPARTO.DIP# = IMPIEGATO.DIP#
GROUP BY REPARTO.DIP#, REPARTO.NOMINATIVO
```

Supponiamo di voler raggruppare gli impiegati sulla base del dipartimento e della mansione; per ogni gruppo si vogliono determinare il nome del dipartimento, la somma degli stipendi, quanti impiegati appartengono al gruppo e la media degli stipendi

```
SELECT REPARTO.NOMINATIVO, MANSIONE, SUM(STIPENDIO), COUNT(*),
AVG(STIPENDIO)
FROM IMPIEGATO, REPARTO
WHERE REPARTO.DIP#= IMPIEGATO.DIP#
GROUP BY REPARTO.NOMINATIVO, MANSIONE;
```

Reparto.Nominativo	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

La Clausola HAVING

E' possibile specificare condizioni di ricerca su gruppi di tuple usando la clausola `HAVING` dopo la clausola `GROUP BY`.

Sintassi

- `HAVING ESPRESSIONE`

dove l'espressione è specificata come nella clausola `WHERE`, ma può fare riferimento solo a funzioni di gruppo.

Esempio

supponiamo di voler raggruppare gli impiegati sulla base del dipartimento e della mansione; per ogni gruppo si vogliono determinare il nome del dipartimento, la somma degli stipendi, quanti impiegati appartengono al gruppo e la media degli stipendi. Inoltre, siamo interessati solo ai gruppi che contengono almeno due impiegati

```
SELECT REPARTO.NOMINATIVO, MANSIONE, SUM(STIPENDIO), COUNT(*),
AVG(STIPENDIO)
FROM IMPIEGATO, REPARTO
WHERE REPARTO.DIP#= IMPIEGATO.DIP#
GROUP BY REPARTO.NOMINATIVO, MANSIONE;
HAVING COUNT(*) >= 2;
```

Reparto.Nominativo	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

Determinare i reparti che spendono più di 100 in stipendi

```
SELECT REPARTO, SUM(STIPENDIO) AS SOMMASTIPENDI
FROM IMPIEGATO
GROUP BY REPARTO
```

```
HAVING SUM(STIPENDIO)> 100
```

Reparto	SommaStipendi
Amm.ne	125
Direz.ne	153

Determinare i reparti per cui la media degli stipendi degli impiegati dell'ufficio 20 è superiore a 25

```
SELECT REPARTO
FROM IMPIEGATO
WHERE UFFICIO=20
GROUP BY REPARTO
HAVING AVG(STIPENDIO) > 25
```

Un "modello di esecuzione" per le interrogazioni con raggruppamento

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. Per i gruppi ottenuti al passo precedente vengono calcolate le funzioni di gruppo specificate nella clausola di proiezione `SELECT` della query.

Trattamento dei valori NULL

Il valore null rappresenta "assenza di informazione". Si usa per dare "valore" ad attributi non applicabili o dal valore non noto. Su una tabella vuota tutte le funzioni di gruppo hanno valore `NULL` tranne `COUNT`, che vale zero.

Come trattare i valori nulli nelle query?
`WHERE STIPENDIO > 40` è verificato da tutte le righe con valore di stipendio superiore a 40.
`WHERE STIPENDIO <= 40` è verificato da tutte le righe con valore di stipendio inferiore o uguale a 40.
Le righe con valore nullo non verificano né un predicato né il suo complemento!

NULL nelle clausole WHERE

In SQL-89, la presenza di `NULL` rende un predicato FALSE. Nelle versioni successive di SQL, si fa uso uso del predicato `IS [NOT] NULL` nella clausola WHERE. Questo predicato effettua un test sulla presenza [assenza] di valori nulli.

Per trattare i valori NULL, nel caso l'utente non specifichi esplicitamente un predicato `IS [NOT] NULL` , 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 logico '?'. Il valore di verità di un predicato complesso viene calcolato in base alle seguenti tabelle di verità:

	AND		
	T	**F**	**?**
T	T	F	?
F	F	F	F
?	?	F	?

	OR		
	T	**F**	**?**
T	T	T	T
F	T	F	?
?	T	?	?

NOT
F
T
?

Una tupla per cui il valore di verità è '?' non viene restituita dalla query.

NULL nella clausole GROUP BY

Le tuple con valori NULL in un attributo di raggruppamento vengono poste nello stesso gruppo.

NULL come argomento delle funzioni di gruppo

Le funzioni di gruppo scartano (non usano nei calcoli) i valori NULL. Unica eccezione è costituita dalla funzione Count(*).

Esempio

Data la seguente relazione R:

A	B	C
a	?	c1
a1	b	c2
a2	?	?

```
SELECT *
FROM R
WHERE A=A OR B=B;
```

A	B	C
a	?	c1
a1	b	c2

```
SELECT *
FROM R
WHERE A=A AND B=B;
```

nessuna tupla verifica la query

```
SELECT *
FROM R
WHERE NOT C=c1;
```

A	B	C
a1	b	c2

```
SELECT *
FROM R
WHERE B IS NULL;
```

A	B	C
a	?	c1

a2	?	?

```
SELECT *
FROM R
WHERE B IS NULL AND C IS NULL;
```

A	B	C
a2	?	?

```
SELECT *
FROM R
WHERE B IS NULL OR C IS NULL;
```

A	B	C
a	?	c1
a2	?	?

```
SELECT *
FROM R
WHERE B IS NOT NULL;
```

A	B	C
a1	b	c2

Operazione di JOIN

L'operazione di join è molto importante in quanto permette di correlare dati rappresentati da relazioni diverse. Il join può essere 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.

Esempio

Determinare il nome dei dipartimento in cui lavora l'impiegato Rossi

```
SELECT REPARTO.NOMINATIVO
FROM IMPIEGATO, REPARTO
WHERE NOME = 'ROSSI' AND IMPIEGATO.DIP# = REPARTO.DIP#;
```

il predicato di join è `IMPIEGATO.DIP# = REPARTO.DIP#`

E' possibile eseguire il join tra una tupla di una relazione e più tuple di un'altra relazione.

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.

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

```
SELECT REPARTO.NOMINATIVO, NOME, MANSIONE, STIPENDIO
FROM IMPIEGATO, REPARTO
WHERE IMPIEGATO.DIP# = REPARTO.DIP#
ORDER BY REPARTO.NOMINATIVO, STIPENDIO DESC;
```

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	Biacchi	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

Join di Impiegato e Reparto

```
SELECT #IMP, NOME, STIPENDIO, QUALIFICA, IMPIEGATO.#DIP, NOMINATIVO,
LOCALITÀ
FROM IMPIEGATO, REPARTO
WHERE IMPIEGATO.#DIP = REPARTO.#DIP
```

```
SELECT #IMP, NOME, STIPENDIO, QUALIFICA, I.#DIP, NOMINATIVO, LOCALITÀ
FROM IMPIEGATO I, REPARTO R
WHERE I.#DIP = R.#DIP
```

Join di Impiegato e Reparto per gli impiegati Rossi

```
SELECT #IMP, NOME, STIPENDIO, QUALIFICA, I.#DIP, NOMINATIVO, LOCALITÀ
FROM IMPIEGATO I, REPARTO R
WHERE I.#DIP = R.#DIP AND NOME='ROSSÌ
```

Lista di tutti gli impiegati che guadagnano più di Rossi (theta-join)

```
SELECT ROSSI.NOME, ROSSI.STIPENDIO, ROSSI.QUALIFICA, ALTRI.NOME,
ALTRI.STIPENDIO, ALTRI.QUALIFICA
FROM IMPIEGATO ROSSI, IMPIEGATO ALTRI
WHERE ALTRI.STIPENDIO> ROSSI.STIPENDIO AND ROSSI.NOME='ROSSI'
```

Lista degli impiegati con i rispettivi capi (self-join)

```
SELECT SUBORDINATO.NOME, CAPO.NOME
FROM IMPIEGATO SUBORDINATO, IMPIEGATO CAPO
WHERE SUBORDINATO.CAPO=CAPO.#IMP
```

Join interni (inner join)

Abbiamo visto come è possibile esprimere join mediante *predicati di join* nella clausola `WHERE`. In realtà SQL:1999 prevede diversi tipi di *operatori di join.* Questi operatori producono relazioni e quindi possono essere usati nella clausola `FROM`.

La forma di operatore join più semplice, che corrisponde al prodotto Cartesiano, è il `CROSS JOIN`:

```
IMPEGATI CROSS JOIN DIPARTIMENTI
```

Il theta-join si ottiene come operatore `JOIN ON`:

```
IMPIEGATI JOIN DIPARTIMENTI ON IMPIEGATI.NOME > DIPARTIMENTI.NOME
```

Il join naturale corrisponde all'operatore `NATURAL JOIN`

```
DIPARTIMENTI NATURAL JOIN IMPIEGATI
```

Sintassi alternativa: `JOIN USING`(`LISTANOMICOLONNE`)

```
DIPARTIMENTI JOIN IMPIEGATI USING (DIP#)
```

Esempi

Inner (theta) join con i predicati di join

```
SELECT IMPEGATO.NOME, IMPEGATO.COGNOME, REPARTO.CITTÀ
FROM IMPIEGATO, REPARTO
WHERE IMPIEGATO.REPARTO=REPARTO.NOME
```

Inner (theta) join con gli operatori di join

```
SELECT IMPEGATO.NOME, IMPEGATO.COGNOME, REPARTO.CITTÀ
FROM IMPIEGATO INNER JOIN REPARTO ON IMPIEGATO.REPARTO=REPARTO.NOME
```

Join esterni (outer join)

Nel normale join tra due relazioni R e S non si ha traccia delle tuple di R che non corrispondono ad alcuna tupla di S. Questo non sempre è 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`.
Esistono diverse varianti dell'outer join:

- Left join: fornisce come risultato il join interno esteso con le righe della relazione che compare a sinistra nel join per le quali non esiste una corrispondente riga nella tabella di destra;
- Right join: fornisce come risultato il join interno esteso con le righe della relazione che compare a destra nel join per le quali non esiste una corrispondente riga nella tabella di sinistra;
- Full join: fornisce come risultato il join interno esteso con le righe escluse di entrambe le relazioni.

Nello standard SQL2 sono presenti gli operatori

- `RIGHT [OUTER] JOIN`: le tuple della relazione di destra che non partecipano al join vengono completate e inserite nel risultato
- `LEFT [OUTER] JOIN`: le tuple della relazione di sinistra che non partecipano al join vengono completate e inserite nel risultato
- `FULL [OUTER] JOIN`: le tuple di entrambe le relazioni che non partecipano ai join vengono completate e inserite nel risultato

La variante OUTER può essere utilizzata sia per join naturale che per theta-join.

Esempi

Siano date le seguenti relazioni:
GUIDATORE

Nome	Cognome	Npatente
Mario	Rossi	VR 7777
Carlo	Bianchi	PZ 8888
Marco	Neri	AP9999

AUTOMOBILE

Targa	Marca	NPatente
AB 574 WW	Fiat	VR 7777
AA642FF	Fiat	VR 7777
BJ 741 XX	Lancia	PZ 8888
BB 421 JJ	Fiat	Ml 4444

```
SELECT *
FROM GUIDATORE FULL JOIN AUTOMOBILE ON (GUIDATORE.NPATENTE =
AUTOMOBILE.NPATENTE)
```

Nome	Cognome	Npatente	Targa	Marca
Mario	Rossi	VR 7777	AB 574 WW	Fiat
Mario	Rossi	VR 7777	AA642FF	Fiat
Carlo	Bianchi	PZ 8888	BJ 741 XX	Lancia
Marco	Neri	AP 9999	Null	Null
null	null	MI 4444	BB 421 JJ	Fiat

Subqueries

Una delle ragioni che rendono SQL un linguaggio potente, è la possibilità di esprimere queries complesse in termini di queries 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 utilizzare come valori di confronto in un predicato della query esterna. E' possibile per una subquery avere al suo interno un'altra subquery, predicati di join, e tutti i predicati visti in precedenza.

Le subqueries possono essere usate anche all'interno dei comandi di manipolazione dei dati (`INSERT`, `DELETE`, `UPDATE`)

Esempio

Si vogliono 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.

Trovare i reparti con gli stipendi più elevati

```
SELECT #DIP
FROM IMPIEGATO
WHERE STIPENDIO = (
  SELECT MAX(STIPENDIO)
  FROM IMPIEGATO
);
```

(meglio con un operatore aggregato; più leggibile e in alcuni sistemi più efficiente: vedi dopo)

```
SELECT #DIP
FROM IMPIEGATO
WHERE STIPENDIO >= ALL (
  SELECT STIPENDIO
  FROM IMPIEGATO
);
```

Si vogliono 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
);
```

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

Negli esempi visti finora, le subqueries restituiscono un solo valore. Se la subquery restituisce più valori è 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 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
);
```

Stipendio	Mansione	Nome	Dip#
3000,00	dirigente	Verdi	10
2975,00	dirigente	Rosi	20
2850,00	dirigente	Biacchi	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	Adarni	10

Se si usa il quantificatore ALL, la query restituisce gli impiegati il cui stipendio è maggiore di *tutti* i valori restituiti dalla subquery.

Determinare lo stipendio, la mansione, il nome e il numero di dipartimento degli impiegati che guadagnano più di *tutti* gli impiegati dei dipartimento 30

```
SELECT STIPENDIO, MANSIONE, NOME, DIP#
FROM IMPIEGATI WHERE
STIPENDIO > ALL (
  SELECT STIPENDIO
  FROM IMPIEGATI
  WHERE DIP#=30
  );
```

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

E' inoltre possibile selezionare più di una colonna usando una subbuery: in tal caso è necessario porre tra parentesi la lista delle colonne a sinistra dell'operatore di confronto.

Si vogliono 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 = 'MARTINÌ
);
```

Altri tipi di query

Le query viste finora sono query di selezione. In SQL esistono altri tipi di query che permettono di modificare i dati nelle relazioni. I tipi più importanti sono:

- Query di aggiornamento
- Query di inserimento
- Query di cancellazione

Query di aggiornamento

Le query di aggiornamento permettono di variare il contenuto di tuple già presenti in una particolare relazione del database. Le tuple da modificare vengono selezionate tramite una clausola `WHERE`, e a tutte le tuple selezionate vengono applicati degli aggiornamenti che possono essere costanti o far riferimento agli attributi della stessa tupla.

Sintassi

```
UPDATE NomeTabella
SET Attributo1 = Espressione1,…,AttributoN = EspressioneN
WHERE Condizione
```

dove

- Attributo_I è il nome di un attributo delle tuple della tabella specificata
- Espressione_I è un'espressione costante, o che fa riferimento agli altri attributi della tabella specificata. Si possono utilizzare tutti gli operatori (aritmetici, stringa,…) messi a disposizione dal DBMS (ed utilizzabili, ad esempio, anche nella clausola `SELECT` per creare colonne virtuali).
- `Condizione` è una condizione `WHERE` standard.

Esempio

```
UPDATE Impiegati
SET Stipendio = Stipendio + Stipendio*0.1, Premio_P = 1000
WHERE Mansione = 'Amm.ne'
```

Query di inserimento

Le query di inserimento permettono di inserire nuove tuple in una relazione. L'origine delle tuple può essere una serie di valori costanti forniti dall'utente, o un insieme di tuple restituito da un'altra query.
E' possibile specificare quali attributi verranno inseriti nella tupla e in che ordine. Se non vengono specificati tutti gli attributi, gli altri avranno il loro valore di default (sempre che ciò non violi qualche vincolo).

Sintassi-1

```
INSERT INTO NomeTabella[(Nome_Colonna1,…,Nome_ColonnaN)]
VALUES (Valore_Colonna1,…,Valore_ColonnaN)
```

Viene creata una sola tupla con i valori di ciascuna colonna Nome_Colonna_I impostati al corrispondente Valore_Colonna_I (i cui tipi devono ovviamente essere compatibili).

Sintassi-2

```
INSERT INTO NomeTabella[(Nome_Colonna1,…,Nome_ColonnaN)]
SELECT Attributo1,…,AttributoN
FROM …
WHERE …
```

In questo caso vengono create nuove tuple nella tabella `NomeTabella` per ognuna delle tuple selezionate dalla query. Il valore di ciascun Attributo_I deve essere compatibile col tipo di Nome_Colonna_I.

Esempi

```
INSERT INTO IMPIEGATI(IMP#,NOME,MANSIONE)
VALUES (234,'ROSSI','INGEGNERE')
```

```
INSERT INTO MANSIONI
SELECT DISTINCT MANSIONE
FROM IMPIEGATI
```

Query di cancellazione

Con una query di cancellazione si possono cancellare da una tabella le tuple che rispondono a un certo criterio (espresso con una clausola `WHERE`).

Sintassi

```
DELETE FROM NOMETABELLA
WHERE CONDIZIONE
```

Esempi

```
DELETE FROM IMPIEGATI
WHERE DATA_A >  DATE('1/1/2001')
```

```
DELETE FROM IMPIEGATI
```

Elimina tutte le tuple dalla tabella!