Il linguaggio SQL

Matteo Gorgone
Università degli Studi di Messina

Il linguaggio SQL

Il **linguaggio SQL** (*Structured Query Language*), di tipo non procedurale (o dichiarativo), è divenuto ormai da tempo il linguaggio standard per creare, manipolare e interrogare database relazionali.

Il linguaggio SQL assolve alle funzioni di:

- DDL (*Data Definition Language*);
- DML (*Data Manipulation Language*);
- DCL (*Data Control Language*).

Il linguaggio SQL

Per descrivere la sintassi di un linguaggio di programmazione ci si serve generalmente di un formalismo molto usato: la **BNF** (*Backus-Naur Form*) o **Forma Normale di Backus**.

Convenzioni usate:

- Le parole chiave del linguaggio sono scritte in grassetto;
- Le parole racchiuse tra parentesi angolari (< >) rappresentano categorie sintattiche:

  WHERE <Attributo> <Operatore> <Valore>

  WHERE Stipendio > 2000;
- I blocchi racchiusi tra parentesi quadre [ ] indicano l'opzionalità;
- I blocchi racchiusi tra parentesi graffe { } indicano la possibilità di ripetizione;
- Il simbolo | ha il significato di OR.

Identificatori e tipi di dati

SQL non è un linguaggio case-sensitive.

Gli **identificatori** utilizzati per i nomi delle tabelle e degli attributi devono:

- avere una lunghezza massima di 18 caratteri;
- iniziare con una lettera;
- contenere come unico carattere speciale l'underscore "_", che in molte versioni può creare problemi quando si usano le espressioni regolari.

Terminologia:

- Le relazioni sono chiamate **tabelle**.
- Le n-uple sono chiamate **righe** o **record**.
- Gli attributi sono le **colonne** delle tabelle.

Identificatori e tipi di dati

	Dominio	Descrizione
DOMINI CARATTERE	char(n)	Stringhe di n caratteri (lunghezza fissa). A stringhe più corte sono aggiunti spazi in coda.
	char	Sinonimo di char(1).
	varchar(n)	Stringhe di al più n caratteri.
DOMINI NUMERICI: TIPI NUMERICI ESATTI	smallint	Intero. 2 byte, range $[-2^{15}; 2^{15}-1]$.
	integer	Intero. 4 byte, range $[-2^{31}; 2^{31}-1]$.
	numeric(prec,scala)	Per calcoli esatti fino a 1000 cifre significative. Scala è il numero di cifre dopo la virgola. Prec è il numero di cifre significative. Esempio: 22,4454 ha precisione 6 e scala 4. Gli interi hanno scala 0.
	numeric	Memorizza numeri decimali fino alla precisione massima consentita dall'implementazione, e assume scala 0.
	decimal(prec,scala)	Come numeric(prec,scala) ma assume che prec sia limite inferiore alla precisione.
DOMINI NUMERICI: TIPI NUMERICI APPROSSIMATI	Real	Numeri in virgola mobile. Tipicamente nel range $[10^{-37}; 10^{37}]$, con almeno 6 cifre corrette.
	double precision	Numeri in virgola mobile. Tipicamente nel range $[10^{-307}; 10^{307}]$, con almeno 15 cifre corrette.
	float(prec)	prec specifica la precisione minima accettabile come numero di cifre binarie.
DOMINI TEMPORALI	timestamp	**Data e ora** Esempio: **'25-may-2014 11:15:48'**
	date	**Data** Esempio: **(formato raccomandabile 'anno-mese-giorno') '2014-02-10'**
	time	**Ora** Esempio: **'2.25 am'**
	interval	**Intervalli di tempo** Esempio: **'1 day 12 hours 50 min 10 sec ago'**
DOMINI BOOLEANI	boolean	**Tipo booleano** Esempi di valori booleani: TRUE, FALSE / 't', 'f' / 'true', 'false', 'y', 'n' / 'yes', 'no', '1', '0'
	auto_increment	Campi di incremento automatico vengono utilizzati per i valori di generazione automatica di particolare colonna ogni volta che viene inserita nuova riga. Molto spesso la chiave primaria di una tabella deve essere creata automaticamente; definiamo quel campo come campo che si incrementa automaticamente.

Identificatori e tipi di dati

Le **costanti stringa** sono rappresentabili utilizzando indifferentemente gli apici (‘ ’) o i doppi apici (“ ”).

Nelle **espressioni** possono essere usati i seguenti **operatori**:

- **Aritmetici** (+, –, /, *);
- **Relazionali** [<, >, <= (minore o uguale), >= (maggiore o uguale), <> (diverso)];
- **Logici** (AND, OR, NOT).

I confronti tra dati numerici sono effettuati in base al loro valore algebrico.

I confronti tra dati alfanumerici sono effettuati in base al valore del codice ASCII dei caratteri che li compongono.

Istruzioni del DDL di SQL: un nuovo database

I comandi del **DDL** di SQL creano o modificano lo schema di una base di dati.

Per creare un nuovo database utilizzeremo il comando **CREATE DATABASE**, la cui sintassi è la seguente:

CREATE DATABASE <NomeDatabase> [AUTHORIZATION <Proprietario>];

Se non viene specificata la clausola **AUTHORIZATION** si suppone che il nome del proprietario sia quello dell'utente collegato in quel momento.

Per impartire i comandi SQL per uno specifico database occorre prima selezionare quest'ultimo utilizzando il comando **USE**, la cui sintassi è:

USE <NomeDatabase>

Istruzioni del DDL di SQL: tabella e vincoli di integrità

Per creare una tabella si utilizza la seguente sintassi:

```
CREATE TABLE <NOMETABELLA>
   (<Attributo1> <Tipo1> [<VincoloAttributo1>],
   <Attributo2> <Tipo2> [<VincoloAttributo2>],
   ...,
   <AttributoN> <TipoN> [<VincoloAttributoN>],
   [<VincoloTabella>]);
```

Nella definizione di una tabella sono presenti **vincoli**:

- per **un singolo attributo**;
- per un gruppo di attributi, detti **vincoli di ennupla**;
- per l'**integrità referenziale**.

Vincoli per un singolo attributo

I **vincoli per un singolo attributo** (detti vincoli di dominio) impostano limitazioni da specificare sui valori di un singolo attributo.

Possono essere impostati attraverso le seguenti clausole:

- **NOT NULL**: richiede che l'attributo debba avere necessariamente un valore e non può rimanere non specificato.

Esempio:

Stipendio **INT(10) NOT NULL**

Stipendio **INT(10) NULL**

Vincoli per un singolo attributo

- **DEFAULT** <ValoreDiDefault>: assegna all'attributo il valore di default specificato in <ValoreDiDefault>.

Esempi:

Pensionato **BOOLEAN DEFAULT TRUE**

Stipendio **DECIMAL(5,2) DEFAULT 1500**

Vincoli per un singolo attributo

- **CHECK**(<Condizione>): serve per specificare un vincolo qualsiasi che riguarda il valore di un attributo.

Check(Stipendio>1000)

All'interno di CHECK è possibile usare, oltre agli operatori di confronto, anche gli operatori seguenti.

<Attributo> **IN** (<Valore1>, ..., <ValoreN>)	Richiede che il valore di <Attributo> sia tra quelli specificati da: <Valore1>, ..., <ValoreN>
<Attributo> **BETWEEN** <Min> **AND** <Max>	Richiede che il valore di <Attributo> sia compreso tra i valori <Min> e <Max>
<Attributo> **NOT BETWEEN** <Min> **AND** <Max>	Richiede che il valore di <Attributo> non sia compreso tra i valori <Min> e <Max>
<Attributo> **LIKE** <Espressione1>	Richiede che il valore di <Attributo> assuma il formato specificato da <Espressione1>
<Attributo> **NOT LIKE** <Espressione1>	Richiede che il valore di <Attributo> non assuma il formato specificato da <Espressione1>

Vincoli per un singolo attributo

Esempi con CHECK:

- **CHECK**(Stipendio **IN** (2000,3000,4000))
- **CHECK**(Stipendio **BETWEEN** 2000 **AND** 3000)
- **CHECK**(Stipendio **NOT BETWEEN** 4000 **AND** 5000)
- **CHECK**(CodArticolo **LIKE** "Cod%")
- **CHECK**(CodArticolo **NOT LIKE** "Art%")

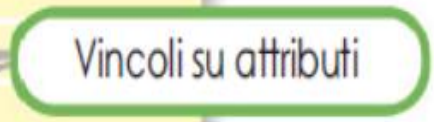
Vincoli per un singolo attributo

Esempio.

Consideriamo la relazione:

AZIENDA(CodAzienda, RagioneSociale, Fatturato, NumDipendenti):

```
CREATE TABLE AZIENDA
(
   CodAzienda          CHAR(5)      NOT NULL,
   RagioneSociale      CHAR(30)     NOT NULL,
   Fatturato           INT(9)       DEFAULT 1000000,
   NumDipendenti       INT(5),
   CHECK(NumeroDip BETWEEN 5 AND 200)
);
```



Vincoli di ennupla

I **vincoli di ennupla** impostano limitazioni da specificare sui valori di più attributi della stessa tabella.

Possono essere impostati con le seguenti clausole:

- **PRIMARY KEY**(<Attributo1>, ..., <AttributoN>): indica le colonne che fanno parte della chiave primaria;
- **UNIQUE**(<Attributo1>,...,<AttributoN>): indica che i valori degli attributi specificati in <Attributo1>,...,<AttributoN> (che non formano una chiave primaria) devono essere distinti all'interno della tabella;
- **CHECK**(<Condizione>): specifica un qualsiasi vincolo che riguarda il valore di più attributi della tabella.

Vincoli di integrità referenziale

I **vincoli di integrità referenziale** possono essere dichiarati con la seguente clausola:

```
FOREIGN KEY(<Attributo1>, ..., <AttributoN>)
REFERENCES <NOMETABELLA>(<Attr1>, ..., <AttrN>)
[[ON DELETE | ON UPDATE] RESTRICT | CASCADE | SET NULL | SET DEFAULT | NO ACTION]
```

dove <Attributo1>,…,<AttributoN> rappresentano la chiave esterna e corrispondono alle colonne <Attr1>,…,<AttrN> della tabella <NOMETABELLA>.

La parte successiva riguarda la politica da seguire in caso violazione del vincolo referenziale: cancellazione (**ON DELETE**) o modifica (**ON UPDATE**) di un attributo interessato da vincolo referenziale.

Vincoli di integrità referenziale

Tipi possibili di politica in caso di violazione del vincolo referenziale:

- **RESTRICT:** l'azione si limita alla riga corrispondente;
- **CASCADE:** l'azione viene applicata a tutte le righe corrispondenti;
- **SET NULL:** le righe corrispondenti vengono impostate a **NULL;**
- **SET DEFAULT:** le righe corrispondenti vengono impostate al valore di default;
- **NO ACTION:** non viene eseguita alcuna azione. È questa l'impostazione di default se non viene specificata la clausola **ON DELETE** o la clausola **ON UPDATE**.

Esempio: creazione di tabelle

```
CREATE TABLE AZIENDA
(
    CodAzienda          CHAR(5)             NOT NULL,
    RagioneSociale      CHAR(30)            NOT NULL,
    Fatturato           DECIMAL(9,2)        DEFAULT 1000000.00,
    NumDipendenti       INT(5),
    PRIMARY KEY(CodAzienda),
    CHECK(NumeroDip BETWEEN 5 AND 200)
);
```

Vincolo di ennupla

```
CREATE TABLE DIPENDENTE
(
    CodDip              CHAR(6)             NOT NULL,
    Cognome             CHAR(30)            NOT NULL,
    Nome                CHAR(20)            NOT NULL,
    DataNascita         DATE,
    DataAssunzione      DATE,
    Livello             CHAR(1)             DEFAULT '6',
    StipendioLordo      DECIMAL(8,3)        NOT NULL,
    CodAzienda          CHAR(5)             NOT NULL,
    PRIMARY KEY(CodDip),
    UNIQUE(Cognome, Nome, DataAssunzione),
    CHECK(DataAssunzione > DataNascita),
    FOREIGN KEY(CodAzienda)             REFERENCES AZIENDA(CodAzienda)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);
```

Vincolo di integrità referenziale

Indici delle tabelle

È possibile legare agli attributi di una tabella alcune tabelle speciali dette **indici**. Gli indici sono file contenenti le chiavi delle tabelle alle quali sono associati, e vengono utilizzati per accelerare i processi di ricerca dei dati, anche se rallentano le operazioni di modifica e richiedono una maggiore occupazione di memoria.

```
CREATE [UNIQUE] INDEX <NomeIndice>
   ON <NOMETABELLA> (<Attributo1>, <Attributo2>, ... , <AttributoN>);
```

La clausola **UNIQUE** crea un indice su attributi chiave; se essa non è presente, viene creato un indice su attributi non chiave.

```
CREATE UNIQUE INDEX IndCodDip
   ON DIPENDENTE (CodDip);
CREATE INDEX IndAlfabetico
   ON DIPENDENTE (Cognome, Nome);
```

Modificare le tabelle

Una volta creata la struttura di una tabella, la si può successivamente modificare con il comando **ALTER TABLE**.

```
ALTER TABLE <NOMETABELLA>
   ADD <NomeColonna1> <NomeTipo>
   [BEFORE <NomeColonna2>];
```

L'istruzione **ADD** è usata per aggiungere la nuova colonna <NomeColonna1>, del tipo specificato da <NomeTipo>, alla tabella <NOMETABELLA >; eventualmente si specifica la posizione in cui inserirla, ad esempio prima (**BEFORE**) della colonna <NomeColonna2>.

```
ALTER TABLE <NOMETABELLA>
   DROP COLUMN <NomeColonna>;
```

L'istruzione **DROP COLUMN** è usata per eliminare la colonna <NomeColonna> dalla tabella <NOMETABELLA>.

Modificare le tabelle

```
ALTER TABLE <NOMETABELLA>
   MODIFY (<NomeColonna>, <NuovoTipoColonna>);
```

L'istruzione **MODIFY** è usata per modificare solo il tipo di una colonna e non il suo nome.

Esempi:

```
ALTER TABLE DIPENDENTE
   ADD TitoloStudio  CHAR(30);
```

```
ALTER TABLE DIPENDENTE
   DROP COLUMN TitoloStudio;
```

Eliminare una tabella

Per cancellare una tabella dalla base di dati si utilizza il comando **DROP TABLE**.

```
DROP TABLE <NOMETABELLA> [RESTRICT | CASCADE | SET NULL];
```

- **RESTRICT:** non permette la cancellazione se la tabella da cancellare è legata tramite vincoli ad altre tabelle;
- **CASCADE:** permette la cancellazione della tabella e di tutti i riferimenti delle chiavi ad essa collegate;
- **SET NULL:** pone a NULL tutti i valori delle chiavi interessate.

```
DROP TABLE AZIENDA;

DROP TABLE AZIENDA RESTRICT;

DROP TABLE AZIENDA CASCADE;
```

Eliminare un indice e un database

Per cancellare un indice si utilizza il comando **DROP** con la seguente sintassi**:**

```
DROP INDEX <NomeIndice> ON <NOMETABELLA>;
```

Esempio:

```
DROP INDEX IndAlfabetico ON DIPENDENTE;
```

Per cancellare un database si utilizza il comando **DROP** con la seguente sintassi**:**

```
DROP DATABASE <NomeDatabase>;
```

Esempio:

```
DROP DATABASE Negozio;
```

Istruzioni del DML di SQL

Una parte di SQL assolve alle funzioni di DML e consente di:

- inserire i valori nelle tabelle (**INSERT INTO**):

```
INSERT INTO <NOMETABELLA> [(<Attributo1>, <Attributo2>, ..., <AttributoN>)]
        VALUES (<Valore1>, <Valore2>, ..., <ValoreN>);
```

Se non è presente la lista degli attributi, si intende che i valori specificati devono corrispondere in ordine, tipo e numero a quelli specificati nella dichiarazione della tabella <NOMETABELLA>. Se si specifica la lista degli attributi, l'ordine e il tipo dei valori devono rispettare questa lista.

Esempio:

```
INSERT INTO AZIENDA
        VALUES ("C001", "A&B Tessile", 1500000.00, 80);
```

Istruzioni del DML di SQL

- modificare una o più righe di una tabella (**UPDATE**):

```
UPDATE <NOMETABELLA>
SET   <Attributo1> = <Espressione1>,
          <Attributo2> = <Espressione2>,
          ...,
          <AttributoN> = <EspressioneN>
[WHERE <Condizione>];
```

dove gli attributi di <NOMETABELLA> (specificati nella clausola **SET**) vengono aggiornati con i valori delle corrispondenti espressioni in tutte le righe che soddisfano la <Condizione>.

Istruzioni del DML di SQL

Esempi con il comando **UPDATE**:

```
UPDATE AZIENDA
SET RagioneSociale = "NuovaElettronica 3000"
WHERE CodAzienda = "A001";
```

```
UPDATE DIPENDENTE
SET StipendioLordo = StipendioLordo + 100.0;
```

```
UPDATE DIPENDENTE
SET StipendioLordo = StipendioLordo + (StipendioLordo * 0,1)
WHERE StipendioLordo < 1000;
```

Istruzioni del DML di SQL

- cancellare una o più righe di una tabella (**DELETE**):

```
DELETE FROM <NomeTabella>
WHERE <Condizione>;
```

In questo modo si eliminano tutte le righe della tabella <NomeTabella> che soddisfano la <Condizione>.

Esempi:

```
DELETE FROM DIPENDENTE
WHERE DataAssunzione <= '1990/12/31';
```

```
DELETE FROM DIPENDENTE
WHERE (CodAzienda = "A001") AND (StipendioLordo > 6000);
```

Istruzioni del DML di SQL

Schema logico relazionale per successivi esempi:

ATTORE(CodiceAttore, Cognome, Nome, Sesso, AnnoNascita, Nazionalita)

FILM(CodiceFilm, Titolo, AnnoProduzione, LuogoProduzione, CognomeRegista, Genere)

CINEMA(CodiceCinema, Nome, Posti, Citta)

INTERPRETA(CodiceAttore, CodiceFilm, Personaggio)

PROGRAMMATO(CodiceFilm, CodiceCinema, Incasso, DataProiezione)

Reperimento dei dati: SELECT

Per estrarre dei dati dal database, il linguaggio SQL prevede il comando **SELECT**. Estrarre dati è sinonimo di effettuare una **query** o **interrogazione** sulla base di dati.

```
SELECT [DISTINCT] <Attributo1>, <Attributo2>,  ..., <AttributoN>
FROM <TABELLA1>, <TABELLA2>, ..., <TABELLAK>
[WHERE <Condizione>];
```

SELECT restituisce una tabella formata dagli attributi: <Attributo1>, ..., <AttributoN> del prodotto delle tabelle: <TABELLA1>, ..., <TABELLAK>, ristretto alle righe che soddisfano la <Condizione>.

Precisiamo che:

- se è assente la clausola **WHERE**, la condizione si assume sempre vera;
- se è presente l'opzione **DISTINCT**, il risultato è fornito privo di righe duplicate.

Se si vogliono visualizzare tutti gli attributi presenti nel prodotto delle tabelle, è possibile utilizzare il simbolo "*", il cui significato sarà: "tutti gli attributi del prodotto delle tabelle". La <Condizione>, inoltre, può essere composta da più condizioni semplici combinate con gli operatori logici AND, NOT, OR.

Reperimento dei dati: SELECT

Esempi:

```
SELECT Titolo
FROM FILM;
```

```
SELECT DISTINCT Titolo
FROM FILM;
```

```
SELECT *
FROM FILM;
```

Alias e calcoli

La tabella risultato di un'operazione **SELECT** ha come intestazione delle colonne il nome degli attributi della tabella interrogata. Se si vuole assegnare un diverso nome a ogni colonna del risultato, cioè se si vuole assegnare un **alias**, si deve utilizzare la clausola **AS**.

```
SELECT Nome, Citta, Posti AS "Numero posti"
FROM CINEMA;
```

È possibile eseguire con il comando **SELECT** il calcolo di un'espressione sugli attributi. Il risultato viene visualizzato a video in una nuova colonna intestata con la clausola **AS**. Il calcolo viene eseguito esternamente alla tabella, quindi senza modificare i dati in essa contenuti.

```
SELECT Incasso * 0,8 AS "Incasso netto"
FROM PROGRAMMATO;
```

Abbreviazioni

Quando si ha a che fare con query che coinvolgono tabelle con campi dello stesso nome è utile utilizzare delle abbreviazioni per fare riferimento ai nomi delle tabelle.

Ad esempio,

```
SELECT FILM.Titolo
FROM FILM;
```

```
SELECT F.Titolo
FROM FILM F;
```

sono forme equivalenti a

```
SELECT Titolo
FROM FILM;
```

“F” è l’abbreviazione della tabella FILM.

Il valore NULL

Il valore **NULL** viene usato in diverse situazioni e assume un ruolo significativo nel risultato di un'espressione logica o aritmetica.

Regole fondamentali:

- **NULL** è diverso da zero e dalla stringa vuota (" ");
- Il risultato di un'espressione aritmetica è sconosciuto (**UNKNOWN**) se un operando ha valore **NULL**;
- Il confronto tra **NULL** e un qualsiasi altro operatore (compreso **NULL**) produce sempre **UNKNOWN**;
- **NULL** non è una costante e non può apparire in un'espressione;
- Se il risultato del predicato della clausola **WHERE** è il valore **UNKNOWN**, la n-upla non viene considerata;
- Nelle funzioni di aggregazione, in generale, le righe con valore **NULL** vengono ignorate;
- Il valore **UNKNOWN** è un valore di verità come **TRUE** o **FALSE**.

Il valore NULL

Nelle interrogazioni è utile controllare se il valore di un attributo è presente oppure è uguale a **NULL**. Possiamo effettuare questa verifica ricorrendo ai predicati **IS NULL** e **IS NOT NULL** nelle condizioni della clausola WHERE.

Esempio:

```
SELECT CodCli, Cognome, Nome
FROM CLIENTI
WHERE Telefono IS NULL;
```

OSS:

È errato scrivere “**WHERE** Telefono = **NULL**”. Infatti, **NULL** è considerato un particolare valore che indica “valore mancante”.

Le operazioni relazionali in SQL

Le operazioni di **proiezione**, **selezione** e **giunzione** su una base dati relazionale vengono realizzate attraverso il comando **SELECT**, secondo le diverse forme previste dalla sintassi.

Proiezione

Proiezione: permette di ottenere una relazione contenente solo alcuni attributi della relazione di partenza e si realizza indicando accanto alla parola **SELECT** l'elenco degli attributi richiesti.

Algebra relazionale	Interrogazione SQL
$\pi_{\text{Cognome, Nome}}$ (ATTORE)	**SELECT** Cognome, Nome **FROM** ATTORE;

Selezione

Selezione: consente di ricavare da una relazione un'altra relazione contenente solo le righe che soddisfano una certa condizione e viene realizzata utilizzando la clausola **WHERE** nel comando **SELECT**.

Algebra relazionale	Interrogazione SQL
$\delta_{Sesso = "M"}$ (ATTORE)	**SELECT** * **FROM** ATTORE; **WHERE** Sesso = 'M';

Join

In SQL è possibile utilizzare i vari tipi di join:

- **CROSS JOIN** (prodotto di relazioni)
- **INNER JOIN** (join interno – equi join)
- **OUTER JOIN** (join esterno):
  - **LEFT JOIN** (outer left join – join esterno sinistro);
  - **RIGHT JOIN** (outer right join – join esterno destro);
- **SELF JOIN** (join su un'unica tabella).

Per gli esempi:

STUDENTE(Matricola, Cognome, Nome, NomeClasse)
CLASSE(NomeClasse, Piano)

Matricola	Nome	Cognome	NomeClasse
1111	Paolo	Rossi	5CA
2222	Gino	Verdi	4CA
3333	Luigi	Neri	NULL

NomeClasse	Piano
5CA	2
4CA	2
3CA	1

Cross join

L’operazione di **CROSS JOIN** corrisponde all’operazione relazionale di prodotto, ovvero la tabella risultato contiene tutte le combinazioni possibili tra i valori delle n-uple della <TABELLA1> e quelli della <TABELLA2>.

```
<TABELLA1> [CROSS] JOIN <TABELLA2>
```

<TABELLA1> e <TABELLA2> potrebbero essere generate da ulteriori query, ad esempio:

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
JOIN CLASSE C;
```

Cross join

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	5CA
Luigi	Neri	NULL	5CA
Paolo	Rossi	5CA	4CA
Gino	Verdi	4CA	4CA
Luigi	Neri	NULL	4CA
Paolo	Rossi	5CA	3CA
Gino	Verdi	4CA	3CA
Luigi	Neri	NULL	3CA

Inner join

La sintassi dell'**INNER JOIN** (equivalente al **NATURAL JOIN**) è:

```
<TABELLA1> INNER JOIN <TABELLA2> ON <Condizione>
```

Viene restituita una tabella, in cui sono presenti solo le combinazioni delle n-uple della prima tabella che trovano una corrispondenza (per gli attributi comuni specificati nella condizione) nell'altra.

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
INNER JOIN CLASSE C
ON S.NomeClasse = C.NomeClasse;
```

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA

Inner join tra tabelle utilizzando SELECT

L'operazione di **INNER JOIN** può essere realizzata anche utilizzando il solo comando **SELECT**. La sintassi è:

```
SELECT <ListaAttributi>
FROM <TABELLA1>, <TABELLA2>
WHERE <TABELLA1>.<AttributoX> = <TABELLA2>.<AttributoX>;
```

Nella clausola **FROM** si specificano i nomi delle due tabelle, <TABELLA1>, <TABELLA2>.
Nella clausola **WHERE** si specifica la condizione sull'attributo comune: <AttributoX>.

Esempio precedente:

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S, CLASSE C
WHERE S.NomeClasse = C.NomeClasse;
```

Left join

La sintassi del **LEFT JOIN** (join sinistro) è:

```
<TABELLA1> LEFT JOIN <TABELLA2> ON <Condizione>
```

Il risultato è dato dalle n-uple della <TABELLA1> (quella che compare a sinistra nella sintassi) e da quelle della <TABELLA2> che hanno un valore corrispondente per l'attributo comune.

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
LEFT JOIN CLASSE C
ON S.NomeClasse = C.NomeClasse;
```

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA
Luigi	Neri	NULL	NULL

Vengono eliminate le n-uple della tabella di destra (<TABELLA2>) che non hanno corrispondenza nella tabella di sinistra (<TABELLA1>).

Right join

La sintassi del **RIGHT JOIN** (join destro) è:

```
<TABELLA1> RIGHT JOIN <TABELLA2> ON <Condizione>
```

Il risultato è dato dalle n-uple della <TABELLA2> (quella che compare a destra nella sintassi) e da quelle della <TABELLA1> che hanno un valore corrispondente per l'attributo comune.

```
SELECT S.Nome, S.Cognome, S.NomeClasse, C.NomeClasse
FROM STUDENTE S
RIGHT JOIN CLASSE C
ON S.NomeClasse = C.NomeClasse;
```

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA
NULL	NULL	NULL	3CA

Vengono eliminate le n-uple della tabella di sinistra (<TABELLA1>) che non hanno corrispondenza nella tabella di destra (<TABELLA2>).

Join tra più di due tabelle

Si può utilizzare **SELECT** per effettuare il **JOIN** tra più di due tabelle.

Esempi:

Visualizzare il titolo dei film in cui recita Tom Cruise oppure Brad Pitt:

```
SELECT DISTINCT F.Titolo
FROM FILM F, INTERPRETA I, ATTORE A
WHERE (A.Cognome = "Cruise" OR A.Cognome = "Pitt")
    AND F.CodiceFilm = I.CodiceFilm AND I.CodiceAttore = A.CodiceAttore;
```

Oppure, utilizzando l'**INNER JOIN**, la stessa query si può scrivere:

```
SELECT DISTINCT F.Titolo
FROM ((FILM F INNER JOIN INTERPRETA I ON F.CodiceFilm = I.CodiceFilm)
    INNER JOIN ATTORE A ON I.CodiceAttore = A.CodiceAttore)
WHERE (A.Cognome = "Cruise" OR A.Cognome = "Pitt");
```

Join tra più di due tabelle

Visualizzare i nomi dei cinema di Roma in cui il giorno di Capodanno 2018 era in programmazione un film con Tom Cruise:

```
SELECT DISTINCT C.Nome
FROM ATTORE A, INTERPRETA I, FILM F, PROGRAMMATO P, CINEMA C
WHERE P.DataProiezione = "01/01/2008" AND C.Citta = "Roma"
    AND A.Nome = "Tom"  AND A.Cognome = "Cruise"
    AND A.CodiceAttore = I.CodiceAttore AND I.CodiceFilm = F.CodiceFilm
    AND F.CodiceFilm=P.CodiceFilm AND P.CodiceCinema=C.CodiceCinema;
```

Visualizzare il titolo del film e il nome dei cinema della città di Milano in cui sono stati proiettati film tra il 15 Marzo 2015 e il 10 Aprile 2015:

```
SELECT DISTINCT F.Titolo, C.Nome
FROM FILM F, PROGRAMMATO P, CINEMA C
WHERE F.CodiceFilm = P.CodiceFilm
    AND P.CodCinema = C.Codcinema AND C.Citta = "Milano"
    AND P.DataProiezione BETWEEN 2015-03-15 AND 2015-04-10;
```

Self-join

Utilizzando gli alias è possibile utilizzare operazioni di **self-join**.

Esempio:

PERSONA(CodPersona, Nome, Cognome)

GENITOREDI(CodPers1, CodPers2)

Se vogliamo ottenere il cognome e il nome delle persone accanto al cognome e nome dei genitori, dobbiamo scrivere:

SELECT Tab1.Cognome, Tab1.Nome, Tab2.Cognome, Tab2.Nome
FROM PERSONA Tab1, PERSONA Tab2, GENITOREDI G
WHERE G.CodPers1=Tab1.CodPersona **AND**
G.CodPers2=Tab2CodPersona

Abbiamo dato il ruolo di genitori e figli rispettivamente a CodPers1 e CodPers2.

Self-join

Sia data la relazione:

GENITORI(Genitore, Figlio)

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

Per trovare la relazione Nonni-Genitori-Nipoti occorre scrivere:

```
SELECT Tab1.Genitore AS 'Nonno', Tab2.Genitore AS 'Genitore',
                    Tab2.Figlio AS 'Nipote'
FROM GENITORI Tab1, GENITORI Tab2
WHERE Tab1.Figlio=Tab2.Genitore;
```

Database: RISTORANTEMARCONI

VINO(IdVino, NomeVino, TipoVino, Annata, Cantina)

CANTINE(IdCantina, Cantina, SiglaProvincia)

PROVINCE(IdProvincia, NomeProvincia, SiglaProvincia, Regione)

Scrivere le query che soddisfano le seguenti richieste:

1. Vini rossi del 2007 e rispettive cantine;
2. Vini bianchi prodotti in provincia di Brescia e rispettive cantine;
3. Vini prodotti in Piemonte e rispettive cantine;
4. Vini rossi del 2000 prodotti in Veneto e rispettive cantine;
5. Vini bianchi del 2012 della cantina ‘Piccariello’;
6. Vini delle regioni che finiscono con la lettera ‘a’ di annata dopo il 2006;

Database: RISTORANTEMARCONI

VINO(IdVino, NomeVino, TipoVino, Annata, Cantina)

CANTINE(IdCantina, Cantina, SiglaProvincia)

PROVINCE(IdProvincia, NomeProvincia, SiglaProvincia, Regione)

7. Vini delle regioni che finiscono con la lettera ‘a’ di annata compresa tra 2000 e il 2010;
8. Tutte le informazioni dei vini rossi delle regioni che finiscono con la lettera ‘a’ di annata compresa tra 2001 e 2009;
9. Nome della provincia in cui vengono prodotti i vini della cantina ‘Radoer’;
10. Regione, nome e annata dei vini bianchi prodotti in provincia di Bolzano.

Unione, intersezione e differenza

Le istruzioni SQL corrispondenti alle operazioni di unione, intersezione e differenza vengono rappresentate, rispettivamente, dagli operatori **UNION**, **INTERSECT** e **MINUS**.

Esempio:

Si considerino le relazioni:

REGISTA(CodRegista, Cognome, Nome)

ATTORE(CodAttore, Cognome, Nome)

- Visualizzare tutti i registi e tutti gli attori.

```
(SELECT Cognome, Nome FROM REGISTA)
UNION
(SELECT Cognome, Nome FROM ATTORE);
```

Unione, intersezione e differenza

Si considerino le relazioni:

REGISTA(CodRegista, Cognome, Nome)

ATTORE(CodAttore, Cognome, Nome)

- Visualizzare i registi che sono stati anche attori.

```
(SELECT Cognome, Nome FROM REGISTA)
INTERSECT
(SELECT Cognome, Nome FROM ATTORE);
```

- Visualizzare i registi che non sono mai stati attori.

```
(SELECT Cognome, Nome FROM REGISTA)
MINUS
(SELECT Cognome, Nome FROM ATTORE);
```

Le funzioni di aggregazione

<FunzioneDiAggregazione>([**DISTINCT**] <Attributo>)

Name	Description
AVG()	Return the average value of the argument
BIT_AND()	Return bitwise AND
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
GROUP_CONCAT()	Return a concatenated string
JSON_ARRAYAGG()	Return result set as a single JSON array
JSON_OBJECTAGG()	Return result set as a single JSON object
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

Le funzioni di aggregazione

- **Ordinamento**:

```
ORDER BY <Attributo1> [ASC | DESC], ... , <AttributoN> [ASC | DESC]
```

- **Raggruppamento**:

```
GROUP BY <Attributo1>, ... , <AttributoN> [HAVING <CondizioneGruppo>]
```

Utilizzando la clausola di raggruppamento, la tabella risultante viene partizionata in gruppi di righe.

Due righe appartengono allo stesso gruppo se hanno gli stessi valori per gli attributi elencati nella clausola **GROUP BY** e tutti i gruppi che non soddisfano la clausola **HAVING** vengono eliminati.

Le funzioni di aggregazione: esempi

- Raggruppare i dipendenti in base al loro livello, conoscere il loro stipendio medio per livello e il loro numero per livello:

```
SELECT Livello, AVG(Stipendio), COUNT(Livello)
FROM DIPENDENTE
GROUP BY Livello;
```

Livello	AVG(Stipendio)	COUNT(Livello)
5	1600,00	10
6	1700,00	8
7	2000,00	5

- Se, in aggiunta alla query precedente, si volessero raggruppare i dipendenti per livelli ma solo per quelli maggiori del sesto:

```
SELECT Livello, AVG(Stipendio), COUNT(Livello)
FROM DIPENDENTE
GROUP BY Livello
HAVING Livello > 6;
```

Livello	AVG(Stipendio)	COUNT(Livello)
7	2000,00	5

Query e subquery annidate

È possibile costruire tramite il comando **SELECT** un'interrogazione al cui interno sono presenti altre interrogazioni, dette **sottointerrogazioni** o **subquery**.

Si può distinguere:

- la **query principale** o **query esterna**: quella individuata dalla prima parola chiave **SELECT**;
- la **query secondaria** o **subquery** o **query interna**: quella individuata dalla seconda parola chiave **SELECT** (delimitata all'interno di parentesi tonde). La query interna passa i risultati alla query esterna che li verifica nella condizione che segue la clausola **WHERE**.

La subquery genera una **tabella derivata** composta da:

- Un solo valore (**tabella scalare** formata da una riga e una colonna);
- Una sola riga ma più colonne;
- Più righe e più colonne.

Query e subquery annidate

Una tabella scalare si può utilizzare ogni volta che è richiesto un singolo valore in una query.

Esempio:

DIPENDENTE(CodDipendente, Cognome, Nome, DataStipendio, Stipendio)

Visualizzare il nome dei dipendenti che hanno uno stipendio maggiore della media.

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

← **NO!**

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

← **SI!**

Tipi di subquery

Subquery che producono un singolo valore: è possibile utilizzare tali sottointerrogazioni nelle espressioni delle query.

Esempio: visualizzare il titolo dei film prodotti dal regista di ‘Via col vento’.

```
SELECT F.Titolo
FROM FILM F
WHERE F.Regista = (SELECT F.Regista
                   FROM FILM F
                   WHERE F.Titolo = "Via col vento");
```

Produce un solo valore come risultato

Subquery che producono valori appartenenti a un insieme.

Nelle clausole **WHERE** delle sottointerrogazioni è possibile utilizzare alcuni predicati per effettuare ricerche sui valori di attributi che soddisfano proprietà di appartenenza a insiemi di valori.
I predicati utilizzabili sono:

ANY e **ALL**; **IN** e **NOT IN**; **EXISTS** e **NOT EXISTS**.

Tipi di subquery: ANY e ALL

La sintassi è:

```
SELECT <ListaAttributi>
FROM <ListaTabelle>
WHERE <Attributo> <OperatoreRelazionale> ANY | ALL (<Sottoquery>)
```

- **ANY**: la condizione della clausola **WHERE** è vera se il valore dell'attributo <Attributo> compare in almeno uno dei valori forniti dalla subquery <Sottoquery>;
- **ALL**: la condizione della clausola **WHERE** è vera se il valore dell'attributo <Attributo> compare in tutti quelli restituiti dalla subquery <Sottoquery>.

Tipi di subquery: ANY e ALL

Si consideri lo schema relazionale:

DIPENDENTE(CodStipendio, Nome, Cognome, StipendioNetto)
STIPENDIO(CodNazione,Nazione,Continente,StipendioMedio)

- Visualizzare i dipendenti che hanno lo stipendio superiore *ad almeno uno* degli stipendi medi delle nazioni europee.

```
SELECT Nome, Cognome
FROM DIPENDENTE
WHERE StipendioNetto > ANY (SELECT DISTINCT StipendioMedio
                            FROM STIPENDIO
                            WHERE Continente = "Europa");
```

- Visualizzare i dipendenti che hanno lo stipendio superiore a *tutti* gli stipendi medi delle nazioni europee.

```
SELECT Nome, Cognome
FROM DIPENDENTE
WHERE StipendioNetto > ALL (SELECT DISTINCT StipendioMedio
                            FROM STIPENDIO
                            WHERE Continente = "Europa");
```

Tipi di subquery: IN e NOT IN

La sintassi è:

```
SELECT <ListaAttributi>
FROM <ListaTabelle>
WHERE <Attributo> <OperatoreRelazionale> IN | NOT IN (<Sottoquery>);
```

- **IN**: la condizione della clausola **WHERE** è vera se il valore dell'attributo <Attributo> appartiene all'insieme dei valori fornito dalla subquery <Sottoquery>;
- **NOT IN**: la condizione della clausola **WHERE** è vera se il valore dell'attributo <Attributo> non appartiene all'insieme dei valori fornito dalla subquery <Sottoquery>.

Tipi di subquery: IN e NOT IN

Si consideri lo schema relazionale:

ZOO(CodAnimale, Nome, DataArrivo, CodRazza)
RAZZA(CodRazza,Descrizione, Nazione, Continente,SpecieProtetta)

- Quanti sono gli animali dello zoo originari dell'Africa?

```
SELECT COUNT(CodAnimale)
FROM ZOO
WHERE CodRazza IN (SELECT CodRazza
                   FROM RAZZA
                   WHERE Continente = "Africa");
```

- Quanti sono gli animali dello zoo che non sono specie protette?

```
SELECT COUNT(CodAnimale)
FROM ZOO
WHERE CodRazza NOT IN (SELECT CodRazza FROM RAZZA WHERE SpecieProtetta = 1);
```

Tipi di subquery: EXISTS e NOT EXISTS

La sintassi è:

```
SELECT <ListaAttributi>
FROM <ListaTabelle>
WHERE EXISTS | NOT EXISTS (<Sottoquery>)
```

- **EXISTS**: la condizione della clausola **WHERE** è vera se la subquery <Sottoquery> produce una tabella non vuota;
- **NOT EXISTS**: la condizione della clausola **WHERE** è vera se il risultato della subquery <Sottoquery> è una tabella vuota, senza alcuna riga.

Le viste

In SQL è possibile definire un'altra classe di tabelle, chiamate **viste**, che non sono fisicamente memorizzate nella base di dati (sono infatti costruite nella memoria RAM), ma che possono essere definite solo logicamente.

Le viste vengono solitamente utilizzate per:

- proteggere i dati;
- convertire le unità di misura;
- semplificare la costruzione di query complesse.

Per creare una vista si utilizza la seguente sintassi:

```
CREATE VIEW <NOMEVISTA> AS <Query>;
```

- <NOMEVISTA> è il nome assegnato alla tabella fittizia della vista;
- <Query> è una normale query formulata con il comando **SELECT**.

Le viste

Consideriamo la seguente relazione:

ACCESSORIO(CodiceAccess, Descrizione, PrezzoAcquisto,
PrezzoVendita, Quantità)

Vista che permette di visualizzare gli accessori che devono essere ordinati:

```
CREATE VIEW DAORDINARE AS
SELECT *
FROM ACCESSORIO
WHERE Quantita = 0;
```

Per eliminare una vista:

DROP VIEW <NOMEVISTA>

Per concedere ad un utente i diritti di accesso ad una vita:

GRANT SELECT ON <NOMEVISTA> **TO** <NomeUtente>