

## Unità **C4**

# Lo standard SQL



**ESERCIZI COMMENTATI**

Segui la risoluzione passo passo



**LABORATORI "CASE STUDIES"**

Approfondisci con i casi pratici

### In questa unità imparerai...

- A comprendere lo standard SQL
- A conoscere i vari comandi del linguaggio SQL
- A formulare interrogazioni per estrarre dati da un database
- A costruire query complesse per estrarre dati specifici

# 1 Il linguaggio SQL

Il **linguaggio SQL** (*Structured Query Language*), di tipo non procedurale (o dichiarativo), è divenuto ormai da tempo lo standard per creare, manipolare e interrogare database relazionali (come, ad esempio, Microsoft Access, Oracle, Microsoft SQL Server, Informix, MySQL e così via).

È presente in diverse versioni o dialetti, che sono in genere aderenti agli standard internazionali ANSI (*American National Standards Organization*) e ISO (*International Standards Organization*); tutte comunque si rifanno alla versione dello standard adottata nel 1992, detta **SQL-2** o **SQL-92**.

Il linguaggio SQL assolve alle funzioni di:

- **DDL** (*Data Definition Language*), che prevede le istruzioni per definire la struttura delle relazioni della base di dati; serve quindi a creare tabelle, vincoli, viste e così via;
- **DML** (*Data Manipulation Language*), che prevede le istruzioni per manipolare i dati contenuti nelle diverse tabelle; in particolare permette di effettuare inserimenti, cancellazioni e modifiche delle righe delle tabelle, nonché di effettuare interrogazioni sulle basi di dati;
- **DCL** (*Data Control Language*), che prevede istruzioni per controllare il modo in cui le operazioni vengono eseguite; consente di gestire il controllo degli accessi per più utenti e i permessi per gli utenti autorizzati.

SQL può essere usato nelle seguenti modalità:

- *stand-alone* o a sé stante;
- *embedded* o a linguaggio ospite.

In particolare adotteremo le seguenti convenzioni:

- le parole chiave del linguaggio sono scritte in **grassetto**.
- le parole racchiuse tra parentesi angolari (< >) rappresentano **categorie sintattiche**, ossia elementi generali del linguaggio che, negli specifici programmi, saranno sostituiti con opportune occorrenze. Ad esempio, la scrittura:

```
WHERE <Attributo> <Operatore> <Valore>
```

significa che occorre mantenere la parola chiave **WHERE** e sostituire <Attributo> e <Valore> con opportune occorrenze. Ad esempio:

```
WHERE Cognome = "Rossi"  
WHERE Stipendio > 1500  
WHERE Data <> "12/05/2014"
```

- I blocchi racchiusi tra parentesi quadre [ ] indicano **opzionalità**, ossia il fatto che tali blocchi possono anche non essere presenti. Ad esempio:

```
SELECT [DISTINCT] <Attributo1>, <Attributo2>, ..., <AttributoN>
```

indica che la parola chiave **DISTINCT** può essere utilizzata o meno ma, se si desidera inserirla, va posta esattamente dopo **SELECT**.

- I blocchi racchiusi tra parentesi graffe { } indicano la possibilità di **ripetizione**. Ad esempio:

```
SELECT [DISTINCT] {, <Attributo>}
```

indica che è possibile inserire più attributi separati da una virgola. Sono pertanto valide le seguenti righe di codice:

```
SELECT Nome
```

```
SELECT Nome, Cognome
```

- Il simbolo | ha il significato di **OR**: all'interno di una lista di parole chiave separate da questo simbolo, occorre sceglierne **soltanto una**. Ad esempio, in base alla seguente sintassi:

```
ANY | ALL <Sottoquery>
```

Sottoquery dovrà essere preceduta dalla parola chiave **ANY** oppure da **ALL**.

## 2 Identificatori e tipi di dati

SQL non è un linguaggio case-sensitive, pertanto le istruzioni possono essere scritte utilizzando indifferentemente caratteri maiuscoli o minuscoli.

Le istruzioni vengono generalmente separate dal punto e virgola ";" ma questo non accade per tutte le versioni. Per garantire la migliore leggibilità, utilizzeremo caratteri maiuscoli per le parole chiave e per le tabelle.

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, comunque, può creare problemi quando si usano le espressioni regolari.

### Rifletti

Esistono versioni per le quali è previsto un controllo di tipo case-sensitive per i nomi delle tabelle e per quelli delle variabili, mentre per i comandi non si fa differenza.

Nella terminologia SQL:

- le relazioni sono chiamate **tabelle**;
- le tuple sono chiamate **righe** o **record**;
- gli attributi sono le **colonne** delle tabelle.

I **tipi di dato** (o **domini** della tabella) utilizzabili per gli attributi sono riepilogati nella tabella seguente. In alcune versioni di SQL tali tipi potrebbero essere differenti e si classificano in:

- Domini elementari (predefiniti):
  - Carattere: singoli caratteri o stringhe di lunghezza variabile;
  - Bit: flag booleani o stringhe;
  - Numerici: esatti e approssimati;
  - Data, ora e intervalli di tempo.
- Domini definiti dall'utente.

In questa trattazione ci occuperemo solo dei domini elementari.

## Rifletti

Nello standard ISO 9075 il tipo booleano non esiste, ma esiste il tipo **bit**, il cui valore 0 corrisponde a *false* e il cui valore 1 corrisponde a *true*. Inoltre è importante ricordare che il separatore dei decimali è il punto e non la virgola.

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

Sono pertanto valide entrambe le stringhe seguenti:

“Stringa valida”      ‘Stringa valida’

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, cominciando dal carattere più a sinistra (se si tratta di stringhe di più caratteri).

	<b>Dominio</b>	<b>Descrizione</b>
<b>DOMINI CARATTERE</b>	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.
<b>DOMINI NUMERICI: TIPI NUMERICI ESATTI</b>	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.

	<b>Dominio</b>	<b>Descrizione</b>
<b>DOMINI NUMERICI: TIPI NUMERICI APPROXIMATI</b>	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.
<b>DOMINI TEMPORALI</b>	timestamp	<b>Data e ora</b> Esempio: '25-may-2017 11:15:48'
	date	<b>Data</b> Esempio: (formato raccomandabile 'anno-mese-giorno') '2017-02-10'
	time	<b>Ora</b> Esempio: '2.25 am'
	interval	<b>Intervalli di tempo</b> Esempio: '1 day 12 hours 50 min 10 sec ago'
<b>DOMINI BOOLEANI</b>	boolean	<b>Tipo booleano</b> 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 una 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.

# 3 Istruzioni del DDL di SQL

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

## Creare e selezionare un nuovo database

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

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

Questo comando crea un nuovo database di nome **<NomeDatabase>** ed eventualmente specifica in **<Proprietario>** il nome dell'utente proprietario, cioè dell'utente che possiede i privilegi di accesso e che, come tale, è l'unico a poter svolgere determinate azioni sul database. Se non viene specificata la clausola **AUTHORIZATION** si suppone che il nome del proprietario sia quello dell'utente collegato in quel momento. Ad esempio:

```
CREATE DATABASE Negozio;
```

crea una nuovo database di nome Negozio.

```
CREATE DATABASE Negozio AUTHORIZATION Venditore;
```

crea un nuovo database di nome Negozio e assegna i privilegi di accesso all'utente Venditore.

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

```
USE <NomeDatabase>;
```

dove <NomeDatabase> è il nome del database creato con l'istruzione **CREATE DATABASE**.

```
USE Negozio;
```

## Creare una tabella e i 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.

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**: se presente, richiede che il corrispondente attributo debba necessariamente avere un valore e quindi non può rimanere “non specificato” (valore **NULL**). Ad esempio:

```
Stipendio DECIMAL(8,3) NOT NULL
```

- **DEFAULT <ValoreDiDefault>**: assegna all’attributo il valore di default (cioè predefinito) specificato in **<ValoreDiDefault>**. Ad esempio:

```
Pensionato BIT DEFAULT 0
```

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

```
CHECK(Stipendio > 1000)
```

- impedisce che il valore di **Stipendio** sia inferiore a 1000.

- All'interno di **CHECK** è possibile utilizzare, oltre agli operatori di confronto, anche i seguenti operatori:

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

Ad esempio, per richiedere che l'attributo Stipendio sia uguale a uno dei valori 1500, 2000, 2500 e 3000, scriveremo:

```
CHECK(Stipendio IN(1500, 2000, 2500, 3000))
```

Per specificare che l'attributo Stipendio sia compreso tra 1500 e 3000 scriveremo:

```
CHECK(Stipendio BETWEEN 1500 AND 3000)
```

Per specificare che l'attributo CodiceArticolo inizi con Cod scriveremo:

```
CHECK(CodiceArticolo LIKE "Cod%")
```

Il carattere % (**carattere jolly**) rappresenta una sequenza di zero o più caratteri.

Consideriamo la seguente relazione:

```
AZIENDA(CodAzienda, RagioneSociale, Fatturato, NumDipendenti)
```

Creiamo la relativa tabella imponendo i seguenti vincoli:

- i valori degli attributi CodAzienda e RagioneSociale devono essere sempre presenti;
- il valore di default di Fatturato è 1000000;
- il numero dei dipendenti deve essere compreso tra 5 e 200.

Scriveremo:

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

Vincoli su attributi

Se i valori degli operatori **IN**, **BETWEEN**, **NOT BETWEEN** sono di tipo stringa, occorrerà racchiuderli tra apici o virgolette. Nel seguito, li utilizzeremo entrambi.

# 4 Vincoli di ennupla e di integrità

Per comprendere questi vincoli ci serviremo della precedente relazione AZIENDA e della nuova relazione DIPENDENTE:

```
AZIENDA(CodAzienda, RagioneSociale, Fatturato, NumDipendenti)  
DIPENDENTE(CodDip, Cognome, Nome, DataNascita, DataAssunzione, Livello,  
              StipendioLordo, CodAzienda)
```

## 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. Ad esempio, per la tabella AZIENDA avremo:

```
PRIMARY KEY(CodAzienda)
```

e per la tabella DIPENDENTE:

```
PRIMARY KEY(CodDip)
```

- **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 (formano cioè una chiave candidata). Ad esempio, considerando gli attributi Cognome, Nome e DataAssunzione della tabella DIPENDENTE, possiamo scrivere:

```
UNIQUE(Cognome, Nome, DataAssunzione)
```

- **CHECK(<Condizione>)**: specifica un vincolo che riguarda il valore di più attributi della tabella. Ad esempio, considerando gli attributi DataNascita e DataAssunzione della tabella DIPENDENTE possiamo scrivere:

```
CHECK(DataNascita < DataAssunzione)
```

## Vincoli di integrità referenziale e politiche di violazione

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]
```

Le colonne *<Attributo1>, ..., <AttributoN>* rappresentano la chiave esterna e corrispondono alle colonne *<Attr1>, ... <AttrN>* che formano la chiave primaria della tabella *<NOMETABELLA>*.

Non è indispensabile citare le colonne *<Attr1>, ... <AttrN>* quando la chiave primaria della tabella *<NOMETABELLA>* è costituita da una sola colonna.

La parte successiva è relativa al tipo di politica da seguire in caso di violazione del vincolo referenziale. I tipi possibili sono: **CASCADE, SET NULL, SET DEFAULT, NO ACTION**.

**RESTRICT**

Al momento della **cancellazione** di un attributo interessato da un vincolo referenziale (cosa che si specifica attraverso la clausola **ON DELETE**) si ha il seguente comportamento, a seconda del tipo di politica specificato:

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

Al momento di una **modifica** di un attributo interessato da un vincolo referenziale (cosa che si specifica attraverso la clausola **ON UPDATE**) si ha il seguente comportamento, a seconda del tipo di politica specificato:

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

Esaminiamo ora le istruzioni SQL per la creazione delle due tabelle AZIENDA e DIPENDENTE, tenendo conto delle nuove clausole introdotte qui:

```
CREATE TABLE AZIENDA
(
    CodAzienda      CHAR(5)      NOT NULL,
    RagioneSociale  CHAR(30)     NOT NULL,
    Fatturato        NUMERIC(9,2) DEFAULT 1000000.00,
    NumDipendenti   INTEGER,
    PRIMARY KEY (CodAzienda),
    CHECK(NumeroDip BETWEEN 5 AND 200) ← Vincoli 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 NUMERIC(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

## ORA TOCCA A TE

---

1. Utilizzando il linguaggio SQL, crea le tabelle necessarie per implementare il seguente schema relazionale:

FILM(CodiceFilm, Titolo, Regista, Anno, CostoNoleggio)

ARTISTA(CodiceAttore, Cognome, Nome, Sesso, DataNascita, Nazionalità)

INTERPRETAZIONI(CodiceFilm, CodiceAttore, Personaggio)

## 5 Indici e modifica delle tabelle

È possibile legare agli attributi di una tabella alcune tabelle speciali dette **indici**. Tali indici sono file contenenti le chiavi delle tabelle alle quali sono associati. Gli indici vengono utilizzati automaticamente da SQL per accelerare i processi di ricerca dei dati (all'interno della tabella a cui sono associati), anche se rallentano le operazioni di modifica e richiedono una maggiore occupazione di memoria. Il comando per creare un indice per una tabella è:

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

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

```
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 NUMERIC(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

Le seguenti istruzioni creano due indici sulla tabella Dipendente:

- il primo indice viene creato sull'attributo chiave CodDip, per facilitare le operazioni di ricerca per chiave;
- il secondo indice viene creato su attributi non chiave (è utilizzato per la ricerca in ordine alfabetico).

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

## Modificare la struttura di una tabella

Una volta creata la struttura di una tabella, la si può successivamente modificare con il comando **ALTER TABLE**, la cui sintassi è:

```
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>**

oppure:

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

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

oppure:

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

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

Ad esempio, per aggiungere l'attributo **TitoloStudio** alla tabella **Dipendente** possiamo scrivere:

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

e per eliminarlo:

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

## Eliminare una tabella

Per cancellare completamente una tabella dalla base di dati si utilizza il comando **DROP**, la cui sintassi è:

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

La cancellazione di una tabella può provocare inconsistenze dovute al fatto che potrebbero esistere tabelle collegate tramite chiavi esterne o, più in generale, tramite vincoli di integrità. Per far fronte a tali situazioni si utilizzano le clausole:

- **RESTRICT**, che non permette la cancellazione se la tabella da cancellare è legata tramite vincoli ad altre tabelle;
- **CASCADE**, che dà luogo a una cancellazione in cascata di tutte le tabelle collegate;
- **SET NULL**, che pone a **NULL** tutti i valori delle chiavi interessate.

Esaminiamo ora le istruzioni SQL per la creazione delle due tabelle AZIENDA e DIPENDENTE, tenendo conto delle nuove clausole introdotte qui:

```
CREATE TABLE AZIENDA
(
    CodAzienda      CHAR(5)      NOT NULL,
    RagioneSociale  CHAR(30)     NOT NULL,
    Fatturato        NUMERIC(9,2) DEFAULT 1000000.00,
    NumDipendenti   INTEGER,
    PRIMARY KEY (CodAzienda),
    CHECK(NumeroDip BETWEEN 5 AND 200) ← Vincoli 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 NUMERIC(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

Per cancellare la tabella AZIENDA dell'esempio precedente scriveremo:

```
DROP TABLE AZIENDA;
```

Se, invece, scrivessimo:

```
DROP TABLE AZIENDA RESTRICT;
```

l'operazione non sarebbe consentita qualora esistessero valori chiave nell'attributo CodAzienda della tabella DIPENDENTE che si riferissero a tuple di AZIENDA.

Scrivendo invece:

```
DROP TABLE AZIENDA CASCADE;
```

verrebbe cancellata la tabella AZIENDA e, con essa, tutti i riferimenti alle chiavi di Dipendente presenti nell'attributo CodDip di AZIENDA.

## Eliminare un indice e un database

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

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

Ad esempio:

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

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

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

Ad esempio:

```
DROP DATABASE Negozio;
```

# 6 Istruzioni del DML di SQL

Occupiamoci ora della parte di SQL che assolve alle funzioni di DML (*Data Manipulation Language*) che consente di **inserire** dati nelle tabelle, di **modificarli** e soprattutto di **visualizzarli** attraverso opportune **interrogazioni**. Le corrispondenti istruzioni che assolvono a tale scopo sono **INSERT**, **UPDATE**, **DELETE** e **SELECT**.

## Inserire i valori in una tabella

I valori delle righe possono essere inseriti utilizzando il comando **INSERT INTO**, la cui sintassi è:

```
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 invece si specifica una lista di attributi, l'ordine e il tipo dei valori dovranno rispettare questa lista. Si assume che il valore per gli attributi omessi sia **NULL**.

Inizializziamo la tabella AZIENDA del nostro solito esempio inserendo la seguente riga:

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

Se, invece, utilizziamo l'istruzione:

```
INSERT INTO AZIENDA (CodAzienda)  
    VALUES ("C002");
```

inseriremo il codice “C002” come valore dell’attributo CodAzienda e **NULL** come valore degli altri attributi.

## Modificare i valori delle righe di una tabella

Per aggiornare una o più righe di una tabella utilizziamo il comando **UPDATE**, la cui sintassi è:

```
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>**.

Riferiamoci sempre al nostro esempio. Per cambiare la ragione sociale dell'azienda A001, scriveremo:

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

Per incrementare di 100 euro il valore dello stipendio di tutti i dipendenti, scriveremo:

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

Per incrementare del 10% lo stipendio lordo di quei dipendenti che percepiscono meno di 1000 euro, scriveremo:

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

## Cancellare righe di una tabella

Per cancellare una o più righe di una tabella utilizziamo il comando **DELETE**, la cui sintassi è:

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

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

Per cancellare i dipendenti assunti prima del 31 dicembre 1990, scriveremo:

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

Per cancellare i dipendenti dell'azienda caratterizzata dal codice A001 che hanno uno stipendio lordo maggiore di 6000 euro, scriveremo:

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

# 7 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. Il risultato è sempre una tabella. La sintassi del comando **SELECT** è molto complessa; vediamo una sua forma semplificata:

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

**SELECT** restituisce una tabella formata dagli attributi: <Attributo1>, <Attributo2>, ..., <AttributoN> del prodotto delle tabelle: <TABELLA1>, <TABELLA2>, ..., <TABELLAK> ristretto alle righe che soddisfano la <Condizione>; 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**. Vediamo alcuni semplici esempi. Per visualizzare il titolo di tutti i film scriveremo:

```
SELECT Titolo  
FROM FILM;
```

Supponendo che possano esistere film con lo stesso titolo, potremmo evitare di scrivere righe duplicate nel seguente modo:

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

Per visualizzare tutti gli attributi dei film scriveremo:

```
SELECT *  
FROM FILM;
```

## 8 Alias e calcoli

La tabella risultato di un'operazione **SELECT** ha (per default) 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**.

Per visualizzare il nome, la città e i posti presenti in un cinema, utilizzando l'intestazione “Numero posti” per l'ultima colonna, scriveremo:

```
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.

Se vogliamo detrarre una quota del 20% dall'incasso di tutte le proiezioni:

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

Spesso è utile utilizzare delle abbreviazioni per fare riferimento ai nomi delle tabelle. Ad esempio, l'interrogazione vista poco fa:

```
SELECT Titolo  
FROM FILM;
```

che consentiva di visualizzare il titolo di tutti i film, potrebbe essere riscritta nel seguente modo:

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

o meglio:

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

dove F è l'abbreviazione usata per la tabella FILM. Questo modo di procedere risulta particolarmente utile quando si ha a che fare con query che coinvolgono tabelle con campi dello stesso nome. Per questo motivo, all'interno di questa unità di apprendimento utilizzeremo entrambe le forme, privilegiando quella con le abbreviazioni quando analizzeremo database caratterizzati dalla presenza di attributi con lo stesso nome.

# 9 Il valore **NULL**

SQL prevede il valore **NULL**, che viene utilizzato per indicare diverse situazioni. Un esempio è quando il valore esiste ma è sconosciuto, oppure quando il valore non esiste (pensate all'importo dello stipendio di un disoccupato). Il valore **NULL** assume un ruolo significativo nel risultato di un'espressione logica o aritmetica. Ci sono alcune regole fondamentali da ricordare:

1. il valore **NULL** è diverso da zero (per i dati numerici) e dalla stringa vuota (" ") per i dati alfanumerici;
2. il risultato di un'espressione aritmetica è sconosciuto (**UNKNOWN**) se un operando ha valore **NULL**;
3. il confronto tra **NULL** e un qualsiasi altro valore, compreso **NULL**, produce sempre **UNKNOWN**;
4. il valore **NULL**, che può essere un valore di un attributo, non è una costante, quindi non può apparire in un'espressione;
5. se il risultato del predicato della clausola **WHERE** è il valore **UNKNOWN**, la tupla non viene considerata;
6. nelle funzioni di aggregazione, in generale, le righe con valore **NULL** dell'attributo considerato sono ignorate;
7. il valore **UNKNOWN** è un valore di verità come **TRUE** e **FALSE**.

Nelle interrogazioni è molto 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**. Per elencare tutti i clienti che non hanno numero di telefono scriveremo:

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

Quindi:

```
WHERE Telefono = NULL
```

è errata nella sintassi, mentre:

```
WHERE Telefono IS NULL
```

è corretta. Questo perché **NULL** è considerato un particolare valore che indica “valore mancante”.

# 10 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 consentite dalla sintassi.

## 11 L'operazione di proiezione

L'operazione di **proiezione**, che permette di ottenere una relazione contenente solo alcuni attributi della relazione di partenza, si realizza indicando accanto alla parola **SELECT** l'elenco degli attributi richiesti. Volendo ottenere l'elenco degli attori e visualizzarne soltanto il cognome e il nome, si deve effettuare una proiezione sulla tabella ATTORE estraendo soltanto le colonne corrispondenti.

Algebra relazionale	Interrogazione SQL
$\pi_{\text{Cognome, Nome}}(\text{ATTORE})$	<b>SELECT Cognome, Nome FROM ATTORE;</b>

# 12 L'operazione di selezione

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

Per ottenere l'elenco di tutti gli attori maschi si opera una selezione sulla tabella ATTORE.

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

# 13 L'operazione di giunzione (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).

Analizziamo le differenze tra questi tipi di join utilizzando l'esempio degli studenti e delle classi. Consideriamo il seguente schema relazionale:

STUDENTE (Matricola, Cognome, Nome, NomeClasse)

CLASSE (NomeClasse, Piano)

Supponiamo che il contenuto delle tabelle sia quello mostrato nella seguente figura.

**STUDENTE**

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

**CLASSE**

NomeClasse	Piano
5CA	2
4CA	2
3CA	1

Si evince che Luigi Neri, pur essendo uno studente della scuola, non è ancora stata assegnato a una classe, e la classe 3CA non ha ancora avuto studenti assegnati.

# 14 Join o cross join

Esaminiamo l'istruzione **JOIN** o la sua equivalente **CROSS JOIN**. La sintassi è:

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

Questa operazione corrisponde all'operazione relazionale di **prodotto**, ovvero la tabella risultato contiene tutte le combinazioni possibili tra i valori delle tuple della **<TABELLA1>** e quelli delle tuple della **<TABELLA2>**.

Nell'esempio degli studenti e delle classi il risultato è mostrato dopo il codice.

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

## Rifletti

<TABELLA1>  
e <TABELLA2>  
potrebbero essere  
generate da ulteriori  
query.

## (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

Per individuare le specifiche colonne di cui vogliamo la visualizzazione, poiché possono comparire con lo stesso nome in tabelle diverse, abbiamo usato la dot notation:

```
<NOMETABELLA>. <NomeColonna>
```

# 15 Inner join

La sintassi dell'operazione di **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 tuple della prima tabella che trovano una corrispondenza (per gli attributi comuni specificati nella condizione) nell'altra tabella.

La seguente interrogazione visualizza gli studenti che sono stati iscritti alla scuola e che hanno avuto una classe assegnata.

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

## INNER (o NATURAL) JOIN

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

Lo studente *Luigi Neri* non è visualizzato perché non ha ancora classi assegnate.

La classe 3CA non è visualizzata perché non ha alcuno studente assegnato.

# 16 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>**. Avremmo potuto ottenere lo stesso risultato mostrato in precedenza scrivendo:

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

# 17 Left join

La sintassi dell'operazione di **LEFT JOIN** o join sinistro è:

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

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

La seguente interrogazione visualizza tutti gli studenti che sono iscritti alla scuola anche se non è stata ancora assegnata loro una classe:

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

## LEFT JOIN

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

Gli studenti compaiono tutti, indipendentemente dal fatto che siano o meno stati assegnati alle classi. Lo studente *Luigi Neri*, infatti, non è stato ancora assegnato a una classe, ma compare lo stesso.

La classe *3CA* non compare perché non è collegata a nessuna tupla a sinistra, cioè a nessuno studente.

Il risultato è strettamente legato alla parzialità della diretta; in sostanza elimina le tuple di destra che non hanno una controparte a sinistra.

# 18 Right join

La sintassi dell'operazione di **RIGHT JOIN** o join destro è:

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

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

La seguente interrogazione visualizza tutte le classi presenti nella scuola e solo gli studenti che sono stati assegnati a tali classi:

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

#### RIGHT JOIN

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

Le classi sono presenti tutte, indipendentemente dal fatto che abbiano o meno studenti. La classe 3CA, infatti, non ha studenti assegnati ma compare lo stesso.

Lo studente *Luigi Neri* non compare poiché non è collegato a nessuna tupla a destra, cioè a nessuna classe.

Questa operazione è strettamente legata alla parzialità dell'inversa. In sostanza elimina le tuple di sinistra che non hanno una controparte a destra.

# OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

1. Visualizziamo il nome di tutti cinema presenti nella città di Lecce.

```
SELECT C.Nome  
FROM CINEMA C  
WHERE C.Citta = "Lecce";
```

2. Visualizziamo il nome di tutti gli attori il cui cognome inizia con la lettera C.

```
SELECT A.Cognome, A.Nome  
FROM ATTORE A  
WHERE A.Cognome LIKE "C%";
```

3. Visualizziamo il titolo dei film diretti da Dario Argento o da Ferzan Ozpetek.

```
SELECT F.Titolo  
FROM FILM F  
WHERE F.CognomeRegista = "Argento" OR F.CognomeRegista = "Ozpetek";
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 4.** Visualizziamo il titolo dei film di Dario Argento prodotti dopo il 1985.

```
SELECT F.Titolo  
FROM FILM F  
WHERE F.CognomeRegista = "Argento" AND F.AnnoProduzione > 1985;
```

- 5.** Visualizziamo i cinema presenti nella città di Bari con più di 500 posti in sala.

```
SELECT C.Nome  
FROM CINEMA C  
WHERE C.Citta = "Bari" AND C.Posti > 500;
```

# OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

## 6. Visualizziamo le attrici italiane nate dopo il 1970.

```
SELECT A.Cognome, A.Nome  
FROM ATTORE A  
WHERE A.Sesso = 'F' AND A.Nazionalita = "Italiana" AND A.AnnoNascita > 1970;
```

## 7. Visualizziamo il titolo e il regista dei film gialli prodotti in Italia o in America.

```
SELECT F.Titolo, F.CognomeRegista  
FROM FILM F  
WHERE F.Genere = "Giallo" AND (F.LuogoProduzione = "Italia" OR  
F.LuogoProduzione = "America");
```

# OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 8.** Visualizziamo il titolo e il regista dei film gialli prodotti in Italia o in America dopo il 2000.

```
SELECT F.Titolo, F.CognomeRegista  
FROM FILM F  
WHERE F.Genere = "Giallo" AND (F.LuogoProduzione = "Italia" OR  
F.LuogoProduzione = "America") AND F.AnnoProduzione > 2000;
```

- 9.** Visualizziamo il titolo dei film gialli italiani prodotti dopo il 2000 oppure tedeschi.

```
SELECT F.Titolo  
FROM FILM F  
WHERE F.Genere = "Giallo" AND ((F.LuogoProduzione = "Italia" AND  
F.AnnoProduzione > 2000) OR F.LuogoProduzione = "Germania");
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

### 10. Visualizziamo il titolo e il genere dei film proiettati il giorno di Capodanno 2008.

```
SELECT F.Titolo, F.Genere  
FROM FILM F, PROGRAMMATO P  
WHERE P.DataProiezione = "01/01/08" AND F.CodiceFilm = P.CodiceFilm;
```

oppure:

```
SELECT F.Titolo, F.Genere  
FROM FILM F INNER JOIN PROGRAMMATO P ON F.CodiceFilm = P.CodiceFilm  
WHERE P.DataProiezione = "01/01/08";
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

### 11. Visualizziamo i film che hanno incassato più di 10.000 Euro.

```
SELECT F.Titolo, P.Incasso  
FROM FILM F, PROGRAMMATO P  
WHERE P.Incasso > 10000 AND F.CodiceFilm = P.CodiceFilm;
```

oppure:

```
SELECT F.Titolo, P.Incasso  
FROM FILM F INNER JOIN PROGRAMMATO P ON F.CodiceFilm = P.CodiceFilm  
WHERE P.Incasso > 10000;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 12.** Visualizziamo, per ogni film in cui recita Madonna, il titolo del film e il personaggio interpretato.

```
SELECT F.Titolo, I.Personaggio,  
FROM ATTORE A, INTERPRETA I, FILM F  
WHERE A.CodiceAttore = I.CodiceAttore AND F.CodiceFilm = I.CodiceFilm  
      AND A.Nome = "Madonna";
```

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

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)

## 19 Join tra più di due tabelle

È possibile utilizzare l'istruzione **SELECT** per effettuare l'operazione di join tra più di due tabelle contemporaneamente. Vediamo alcuni 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;
```

Il join tra più tabelle può anche essere ottenuto applicando l'istruzione di inner join in sequenza a gruppi di due tabelle alla volta.

L'interrogazione precedente può essere riscritta nel seguente modo equivalente:

```
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");
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

*Visualizzare i nomi dei cinema di Roma in cui il giorno di Capodanno 2008 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;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

*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;
```

## 20 Self join

Utilizzando gli alias è possibile effettuare operazioni di **self-join**. Consideriamo il seguente esempio.

**PERSONA**(CodPersona, Cognome, Nome)  
**GENITOREDI**(CodPers1, CodPers2)

Se in SQL 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.CodPers AND G.CodPers2 = TAB2.CodPers;
```

Abbiamo dato i ruoli di genitore e figlio rispettivamente alle chiavi **CodPers1** e **CodPers2**.

# 21 Le operazioni di unione, intersezione e differenza

Per tradurre le operazioni dell'algebra relazionale di unione, intersezione e differenza simmetrica, SQL utilizza rispettivamente gli operatori **UNION**, **INTERSECT**, **MINUS** che si applicano ai risultati delle interrogazioni.

Consideriamo le seguenti relazioni:

**REGISTA**(CodRegista, Cognome, Nome)

**ATTORE**(CodAttore, Cognome, Nome)

Per ottenere i registi che sono stati anche attori scriveremo:

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

Le parentesi sono obbligatorie.

Per ottenere i registi che non sono mai stati attori scriveremo:

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

## Rifletti

Nei tre casi precedenti le due relazioni risultato parziale della SELECT devono essere compatibili per poter essere utilizzate come operandi degli operatori di intersezione, unione e differenza simmetrica.

Per ottenere tutti i registi e tutti gli attori scriveremo:

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

## 22 Le query parametriche

Pur non supportate dallo standard ANSI, e quindi assenti nella maggior parte delle versioni di SQL, le interrogazioni parametriche sono molto utili quando si utilizza questo linguaggio in modalità stand alone.

Riferiamoci sempre al nostro database sulle proiezioni cinematografiche e consideriamo la seguente query:

```
SELECT *
FROM FILM
WHERE CognomeRegista = "Ozpetek" AND AnnoProduzione = "2008";
```

Per visualizzare i dati relativi al regista “Verdone” dovremmo riscrivere la query. Alcuni DBMS, come ad esempio Microsoft Access, permettono di parametrizzare una query in modo da scriverla una sola volta e sfruttarla per diversi valori del parametro. Per fare questo, è necessario racchiudere il parametro tra parentesi quadre. Prima di eseguire l’interrogazione, verrà chiesto di inserire il valore per i parametri specificati. L’esempio precedente può allora essere riscritto nel seguente modo:

```
SELECT *
FROM FILM
WHERE CognomeRegista = [Inserire il cognome] AND AnnoProduzione =
      [Inserire l'anno];
```

## 23 Le funzioni di aggregazione

SQL possiede alcune funzioni predefinite, utilissime in molte circostanze in cui occorre effettuare conteggi, somme, calcoli di medie o altro ancora. Tali funzioni si applicano a una colonna di una tabella. La loro sintassi è:

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

dove **<FunzioneDiAggregazione>** può essere:

- **COUNT**, che conta il numero di elementi della colonna **<Attributo>**. Il parametro **<Attributo>** può anche essere sostituito con il carattere \* (asterisco). In questo caso la funzione **COUNT(\*)** calcola il numero delle righe della tabella incluse quelle con campi contenenti valore Null. In altri termini, utilizzando l'asterisco, otteniamo come risultato il numero di tuple che **SELECT** produrrebbe. È importante ricordare che la clausola **DISTINCT** non può essere usata con la funzione **COUNT(\*)**;
- **MIN**, che restituisce il valore minimo della colonna **<Attributo>**;
- **MAX**, che restituisce il valore massimo della colonna **<Attributo>**;
- **SUM**, che restituisce la somma degli elementi della colonna **<Attributo>**;
- **AVG**, che restituisce la media aritmetica degli elementi della colonna **<Attributo>**.

Consideriamo la seguente relazione:

```
DIPENDENTE(CodDip, Cognome, Nome, Livello, DataStipendio, Stipendio)
```

Per conoscere il numero di dipendenti con stipendio >2000 euro, scriveremo:

```
SELECT COUNT(Stipendio)
FROM DIPENDENTE
WHERE Stipendio > 2000;
```

Per conoscere l'esborso totale per gli stipendi dei dipendenti scriveremo;

```
SELECT SUM(Stipendio)
FROM DIPENDENTE;
```

Per conoscere il valore massimo degli stipendi dei dipendenti scriveremo:

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

Per conoscere il valore medio degli stipendi dei dipendenti scriveremo:

```
SELECT AVG(Stipendio) AS StipMedio  
FROM DIPENDENTE;
```

Le funzioni **MIN** e **MAX** operano anche sugli attributi di tipo carattere e di tipo **DATE**.  
Per conoscere l'ultimo cognome dei dipendenti scriveremo:

```
SELECT MAX(Cognome)  
FROM DIPENDENTE;
```

## Ordinamento

In SQL è possibile ordinare le righe di una tabella ottenuta dall'esecuzione di una query utilizzando la clausola **ORDER BY** del comando **SELECT** con la seguente sintassi:

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

dove **ASC** e **DESC** stanno rispettivamente per ordine crescente e decrescente. Per default l'ordinamento è crescente.

L'ordinamento viene eseguito prima sull'**Attributo1**, poi, a parità di ordinamento sull'**Attributo1**, si ordina sulla base dell'**Attributo2** e così via.

Per ottenere una tabella con i dipendenti in ordine alfabetico scriveremo:

```
SELECT *
FROM DIPENDENTE
ORDER BY Cognome, Nome;
```

Per ordinare i dipendenti con stipendi maggiori di 3000 euro in ordine decrescente, scriveremo:

```
SELECT Cognome, Nome, Stipendio
FROM DIPENDENTE
WHERE Stipendio > 3000
ORDER BY Stipendio DESC;
```

## Raggruppamento

Le funzioni di aggregazione sono generalmente abbinate alla clausola di raggruppamento, la cui sintassi è:

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

Con questa clausola il risultato del comando **SELECT** è il seguente:

- viene eseguito il prodotto delle tabelle presenti nella clausola **FROM**;
- su tale prodotto si fa una selezione in base alla clausola **WHERE** (se presente);
- la tabella risultante viene logicamente partizionata in gruppi di righe. Due righe appartengono allo stesso gruppo se hanno gli stessi valori per gli attributi elencati nella clausola **GROUP BY**;
- tutti i gruppi che non soddisfano la clausola **HAVING** vengono eliminati.

Per raggruppare i dipendenti in base al loro livello e conoscere lo stipendio medio per livello, possiamo scrivere:

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

Supponendo che la tabella DIPENDENTE sia popolata con numerose tuple, un risultato dell'interrogazione potrebbe essere quello visibile qui:

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

Per raggrupparli per livello e in più conoscere lo stipendio medio e il numero di dipendenti in quel livello, scriveremo:

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

Il risultato potrà essere:

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

Per raggrupparli per livelli, solo però per quelli maggiori del sesto, scriveremo:

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

Il risultato potrà essere:

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

# OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

## 1. Visualizziamo il numero totale di posti nei cinema di Lecce.

```
SELECT SUM(C.Posti)
FROM CINEMA C
WHERE C.Citta = "Lecce";
```

## 2. Visualizziamo il numero totale di posti nei cinema di una città data in input.

```
SELECT SUM(C.Posti)
FROM CINEMA C
WHERE C.Citta = [Inserire il nome della città];
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

### 3. Visualizziamo il numero totale di cinema presenti nelle varie città.

```
SELECT COUNT(*) AS "Totale cinema"  
FROM CINEMA C;
```

### 4. Visualizziamo il numero di cinema presenti nella città di Firenze che hanno meno di 100 posti.

```
SELECT COUNT(*)  
FROM CINEMA C  
WHERE C.Citta = "Firenze" AND C.Posti < 100;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

### 5. Visualizziamo il numero totale di cinema presenti in ogni città.

```
SELECT C.Citta, COUNT(*)  
FROM CINEMA C  
GROUP BY C.Citta;
```

### 6. Visualizziamo il numero di cinema che hanno meno di 200 posti presenti in ogni città.

```
SELECT C.Citta, COUNT(*)  
FROM CINEMA C  
WHERE C.Posti < 200  
GROUP BY C.Citta;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 7.** Visualizziamo il numero di film prodotti dopo il 2000 in ogni luogo di produzione.

```
SELECT F.LuogoProduzione, COUNT(*)  
FROM FILM F  
WHERE F.AnnoProduzione > 2000  
GROUP BY F.LuogoProduzione;
```

# OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

## 8. Visualizziamo l'incasso totale delle proiezioni dei film di ogni regista.

```
SELECT F.CognomeRegista, SUM(P.Incasso) AS "Totale incassato"  
FROM FILM F INNER JOIN PROGRAMMATO P ON F.CodiceFilm = P.CodiceFilm  
GROUP BY F.CognomeRegista;
```

## 9. Visualizziamo il titolo e l'incasso totale di tutti i film prodotti nel 2015.

```
SELECT F.Titolo, SUM(P.Incasso) AS "Totale incassato"  
FROM FILM F INNER JOIN PROGRAMMATO P ON F.CodiceFilm = P.CodiceFilm  
WHERE F.AnnoProduzione = 2015  
GROUP BY F.Titolo;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

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)

**10.** Visualizziamo per ogni film di animazione e di avventura il titolo e l'incasso totale relativo alle proiezioni effettuate nel primo trimestre del 2015.

```
SELECT F.Titolo, SUM(P.Incasso) AS "Totale incassato"  
FROM FILM F INNER JOIN PROGRAMMATO P ON F.CodiceFilm = P.CodiceFilm  
WHERE F.Genere="Animazione" OR F.Genere = "Avventura"  
      AND P.DataProiezione BETWEEN "2015-01-01" AND "2015-03-31"  
GROUP BY F.Titolo;
```

# OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 11.** Visualizziamo, per ogni film di F. Ozpetek, il titolo del film, il numero totale di proiezioni effettuate in una città fornita in input e l'incasso totale.

```
SELECT F.Titolo, COUNT(*) AS "Totale proiezioni", SUM(P.Incasso) AS "Totale incassato"  
FROM FILM F, PROGRAMMATO P, CINEMA C  
WHERE F.CodiceFilm = P.CodiceFilm  
      AND P.CodiceCinema = C.CodiceCinema  
      AND F.CognomeRegista = "Ozpetek" AND C.Citta = [Inserire nome città]  
GROUP BY F.Titolo;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 12.** Per ogni regista e per ogni attore, visualizziamo il numero di film del regista in cui è presente l'attore. Visualizziamo anche il personaggio interpretato dall'attore.

```
SELECT F.CognomeRegista, A.Cognome, A.Nome, I.Personaggio, COUNT(*) AS  
"Totale film"  
FROM ATTORE A, INTEPRETA I, FILM F  
WHERE A.CodiceAttore = I.CodiceAttore AND I.CodiceFilm = F.CodiceFilm  
GROUP BY F.CognomeRegista, A.Cognome, A.Nome;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

**13.** Per i film in programmazione, visualizziamo il numero di attori nati tra il 1970 e il 1980.

```
SELECT F.Titolo, COUNT(*) AS "Attori presenti nel film"  
FROM ATTORE A, INTERPRETA I, FILM F  
WHERE A.CodiceAttore = I.CodiceAttore AND I.CodiceFilm = F.CodiceFilm  
GROUP BY F.Titolo  
HAVING A.AnnoNascita BETWEEN 1970 AND 1980;
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni esaminate sinora effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 14.** Per ogni film di avventura proiettato dopo il Natale del 2014, visualizziamo il titolo e il totale incassato in tutte le proiezioni.

```
SELECT F.Titolo, SUM(P.Incasso) AS "Totale incassato"  
FROM FILM F, PROGRAMMATO P  
WHERE F.Genere = "Avventura" AND F.CodiceFilm = P.CodiceFilm  
GROUP BY F.Titolo  
HAVING (P.DataProiezione) >= "2014-12-25";
```

# 24 Query e subquery annidate

Per eseguire query complesse è possibile strutturare opportunamente più comandi select. Ciò consente di costruire un'interrogazione al cui interno sono presenti altre interrogazioni, dette **sottointerrogazioni** o **subquery**. In una interrogazione che ne richiama al suo interno un'altra possiamo 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 detta **tabella derivata** che può essere composta da:

- un solo valore (tabella formata di sola riga e una sola colonna): si parla di **tabella scalare**;
- una sola riga (ma più colonne);
- più righe e più colonne.

Una tabella scalare può essere utilizzata tutte le volte in cui è richiesto l'utilizzo di un singolo valore in una query. Facciamo un esempio. Prendiamo in esame la seguente relazione:

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

Vogliamo conoscere il nome dei dipendenti che hanno uno stipendio maggiore della media. Potremmo pensare di realizzare un'interrogazione del seguente tipo:

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

ma il risultato sarebbe un messaggio di errore; infatti non è possibile far seguire la clausola **WHERE** da funzioni di aggregazione. Dobbiamo necessariamente usare una subquery dopo la clausola **WHERE**:

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

In una subquery è spesso conveniente utilizzare la clausola **AS** (già utilizzata per rinominare le colonne). Tale clausola è utile per assegnare un nome alla tabella derivata secondo la sintassi:

```
SELECT <ListaAttributi> FROM <ListaTabelle> AS <NomeTabellaDerivata>
```

La clausola **AS** non crea una nuova tabella, ma consente di fare riferimento con il nuovo nome alla tabella derivata solo ed esclusivamente durante quella query. Se, infatti, elenchiamo le tabelle contenute nel database, non troveremo le tabelle rinominate con questa clausola.

Vediamo nel seguito un esempio di subquery che produce una tabella derivata e con la clausola **AS**.

Consideriamo le seguenti relazioni sull'uso di laboratori da parte di una classe di studenti:

**LABORATORIO**(CodLab, NumPosti, NomeLab)

**CLASSE**(CodClasse, NumPosti)

**UTILIZZA**(CodLab, CodClasse)

e consideriamo la seguente interrogazione: “visualizzare il nome dei laboratori utilizzati dalla classe “A45”. Per rispondere scriveremo:

```
SELECT NomeLab  
      FROM LABORATORIO, (SELECT CodLab  
                          FROM UTILIZZA  
                         WHERE CodClasse = "A45") AS LAB  
      WHERE LAB.CodLab = LABORATORIO.CodLab;
```

Come possiamo vedere, abbiamo una sottointerrogazione all'interno della query principale.

Sottolineiamo l'importanza della clausola **AS**. Nella condizione **WHERE**, per riferirci alla tabella derivata prodotta dalla sottoquery, si è fatto esplicito riferimento al nome della tabella derivata LAB invece di ripetere l'intera sottointerrogazione.

Una subquery può essere composta a sua volta da altre query. Si viene così a creare una struttura di **query annidate**. Naturalmente, è possibile annidare più di una interrogazione, arrivando a una struttura con diversi livelli. Per eseguire sottointerrogazioni annidate si deve eseguire prima l'interrogazione più annidata, poi a seguire quella più esterna fino ad arrivare alla query principale.

# 25 Tipi di subquery: predici ANY e ALL

## Subquery che producono un singolo valore

Finora abbiamo sempre supposto che il risultato di un'interrogazione producesse una tabella. Spesso accade che, in conseguenza dell'impostazione della query, il risultato dell'interrogazione restituisca una tabella costituita dal valore di un solo attributo. Quando si è sicuri che una sottointerrogazione produca un singolo valore come risultato, è possibile utilizzare tale sottointerrogazione nelle espressioni delle query.

Prendiamo in esame il nostro database sulle proiezioni cinematografiche e visualizziamo il titolo dei film prodotti dal regista di "Via col vento":

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

Produce un solo  
valore come risultato

## Subquery che producono valori appartenenti a un insieme: predicati ANY e ALL

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**.

**ANY** e **ALL** vengono utilizzati secondo la sintassi:

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

Il significato è il seguente:

- **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>.

Ad esempio, consideriamo le seguenti relazioni che rappresentano i dipendenti di un'azienda e gli stipendi medi europei di aziende dello stesso settore:

**DIPENDENTE**(CodStipendio, Cognome, Nome, StipendioNetto)

**STIPENDIO**(CodNazione, NomeNazione, Continente, StipendioMedio)

Per rispondere alla domanda: “Quali sono i dipendenti che hanno lo stipendio superiore *ad almeno uno* degli stipendi medi delle nazioni europee?” scrivremo:

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

che restituisce un insieme di valori relativi agli stipendi medi europei. La query principale seleziona il nome e il cognome dei dipendenti che hanno lo stipendio maggiore di almeno uno di quelli delle nazioni europee selezionati dalla sottoquery. Da notare che la clausola **DISTINCT** nella subquery evita che lo stesso valore relativo a nazioni europee con lo stesso stipendio medio compaia più volte.

Per rispondere, invece, alla domanda: "Quali sono i dipendenti che hanno lo stipendio superiore a *tutti* gli stipendi medi delle nazioni europee?" scriveremo:

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

La query principale seleziona il nome e il cognome dei dipendenti che hanno lo stipendio maggiore di tutti quelli delle nazioni europee selezionati dalla subquery.

Ricapitolando:

```
SELECT <ListaAttributi>  
FROM <TABELLE>  
WHERE <Attributo> Δ ALL (SELECT ...)
```

genera una tabella non vuota quando <Attributo> soddisfa la relazione  $\Delta$  ( $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ,  $=$ ,  $\neq$ ) con tutte le tuple restituite da (SELECT ...).

```
SELECT <ListaAttributi>  
FROM <TABELLE>  
WHERE <Attributo> Δ ANY (SELECT ...)
```

genera una tabella non vuota quando <Attributo> soddisfa la relazione  $\Delta$  con almeno una delle tuple.

# 26 Tipi di subquery: predicati IN ed EXISTS

I predicati **IN** e **NOT IN** vengono utilizzati con la sintassi:

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

Il significato è il seguente:

- **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>**.

Ad esempio, consideriamo le seguenti relazioni che rappresentano gli animali presenti in uno zoo e le razze animali presenti nel mondo:

**ZOO**(CodAnimale, Nome, CodRazza, DataArrivo)

**RAZZA**(CodRazza, Descrizione, Nazione, Continente, SpecieProtetta)

Per rispondere alla domanda: "Quanti sono gli animali dello zoo originari dell'Africa?" scrivereemo:

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

La subquery:

```
(SELECT CodRazza  
FROM RAZZA  
WHERE Continente = "Africa");
```

restituisce un insieme di valori relativi alle razze di animali presenti in Africa. La query principale conta il numero degli animali dello zoo tra quelli originari dell'Africa e precedentemente selezionati dalla subquery.

Per rispondere, invece, alla domanda: “Quanti sono gli animali dello zoo che non sono specie protette?” scriveremo:

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

La subquery:

```
(SELECT CodRazza  
FROM RAZZA  
WHERE Specieprotetta = 1);
```

restituisce un insieme di valori relativi alle razze di animali considerati specie protetta (supponiamo che il valore 1 rappresenti una specie protetta). La query principale seleziona il nome, la razza e la data di arrivo degli animali dello zoo la cui razza non è tra quelle considerate specie protette.

I predicati **EXISTS** e **NOT EXISTS** vengono utilizzati con la sintassi:

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

Il significato è il seguente:

- **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.

Ad esempio, consideriamo le relazioni che rappresentano i clienti di un'agenzia viaggi e le richieste di viaggi.

**CLIENTE**(CodCli, Nome, Cognome, Tel)

**RICHIEDE**(CodCli, CodViaggio)

**VIAGGIO**(CodViaggio, Destinazione, Prezzo, NumPersone)

Per rispondere alla domanda: “Quali sono i clienti che hanno richiesto di viaggiare?” scriviamo:

```
SELECT *
FROM CLIENTE C
WHERE EXISTS (SELECT * FROM RICHIEDE H WHERE C.CodCli = H.CodCli);
```

Da notare il differente significato dell’impiego dell’asterisco “\*”: nella query principale serve per visualizzare tutti gli attributi di CLIENTE, mentre nella subquery non ha importanza quali siano gli attributi contenuti nella subquery, ciò che conta è la cardinalità della tabella derivata, non il suo specifico contenuto.

# OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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)

**1.** Visualizziamo il titolo dei film in cui recitano George Clooney e Brad Pitt.

# OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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)

**2.** Visualizziamo i titoli dei film che non sono mai stati proiettati a Bologna.

```
SELECT F.Titolo  
FROM FILM F  
WHERE "Bologna" NOT IN (SELECT C.Citta  
                           FROM PROGRAMMATO P, CINEMA C  
                           WHERE F.CodiceFilm = P.CodiceFilm  
                           AND P.CodiceCinema = C.CodiceCinema);
```

# OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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)

### **3. Visualizziamo i titoli dei film proiettati solo a Bologna.**

```
SELECT F.Titolo  
FROM FILM F  
WHERE NOT EXISTS (SELECT *  
    FROM PROGRAMMATO P, CINEMA C  
    WHERE Citta <> "Bologna"  
        AND F.CodiceFilm = P.CodiceFilm  
        AND P.CodiceCinema = C.CodiceCinema);
```

oppure:

```
SELECT F.Titolo  
FROM FILM F  
WHERE "Bologna" = ALL (SELECT C.Citta  
                          FROM PROGRAMMATO P, CINEMA C  
                          WHERE F.CodiceFilm = P.CodiceFilm  
                          AND P.CodiceCinema = C.CodiceCinema)
```

# OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 4.** Visualizziamo i titoli dei film che non hanno mai avuto una proiezione con incasso inferiore a 1000 Euro.

```
SELECT F.Titolo  
FROM FILM F  
WHERE NOT EXISTS (SELECT *  
                   FROM PROGRAMMATO P  
                   WHERE P.Incasso < 1000 AND F.CodiceFilm = P.CodiceFilm);
```

oppure:

```
SELECT F.Titolo  
FROM FILM F  
WHERE 500 >= ALL (SELECT P.Incasso  
                   FROM PROGRAMMATO P  
                   WHERE F.CodiceFilm = P.CodiceFilm);
```

# OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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*)

**5.** Visualizziamo i titoli dei film che hanno ottenuto un incasso compreso tra 500 e 1000 euro.

## OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 6.** Visualizziamo il nome degli attori inglesi che non hanno mai interpretato ruoli nei film di Ozpetek.

```
SELECT A.Nome
FROM ATTORE A
WHERE A.Nazionalita = "Inglese"
AND NOT EXISTS (SELECT *
                 FROM FILM F, INTERPRETA I
                 WHERE F.CodiceFilm = I.CodiceFilm
                   AND I.CodiceAttore = A.CodiceAttore
                   AND F.CognomeRegista = "Ozpetek");
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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)

**7.** Visualizziamo il titolo dei film di Ozpetek in cui non recitano attori cinesi.

```
SELECT F.Titolo  
FROM FILM F  
WHERE F.CognomeRegista = "Ozpetek"  
AND NOT EXISTS (SELECT *  
                  FROM ATTORE A, INTERPRETA I  
                  WHERE F.CodiceFilm = I.CodiceFilm  
                        AND I.CodiceAttore = A.CodiceAttore  
                        AND A.Nazionalita = "Cinese");
```

## OSSERVA COME SI FA

Riassumiamo le istruzioni relative alle subquery e ai predicati effettuando alcune interrogazioni sulla nostra base di dati:

**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)

- 8.** Visualizziamo gli attori che hanno recitato solo nei film di Ozpetek prodotti prima del 2000.

```
SELECT A.Nome  
FROM ATTORE A  
WHERE NOT EXISTS (SELECT *  
                   FROM FILM F, INTERPRETA I  
                   WHERE F.CodiceFilm = I.CodiceFilm  
                     AND I.CodiceAttore = A.CodiceAttore  
                     AND F.AnnoProduzione < 2000  
                     AND F.CognomeRegista <> "Ozpetek");
```