4

Il linguaggio SQL

ESERCIZI COMMENTATI
Segui la risoluzione passo passo

LABORATORI CASE STUDIES
Approfondisci con i casi pratici

PREREQUISITI

- Conoscere le caratteristiche del modello relazionale.
- Conoscere la programmazione procedurale.
- Conoscere gli aspetti base dell'analisi di un problema.

PER COMINCIARE

1. **Una relazione è:**
   - A una tabella suddivisa in più campi alfanumerici
   - B una struttura dati in grado di contenere valori numerici
   - C una tabella contenente nelle colonne gli attributi e nelle righe i valori
   - D una variabile strutturata che può contenere numeri o stringhe

2. **Quale campo della tabella Studente (Matricola, Cognome, Nome, Età) può essere chiave primaria?**
   - A Matricola
   - B Nome
   - C Cognome
   - D Età

3. **Spiega come si realizzano le associazioni tra dati nel modello relazionale.**

4. **Definisci le relazioni che compongono il database Scuola con i dati degli studenti e degli insegnanti.**

CONOSCENZE

- Conoscere le caratteristiche del linguaggio SQL e come utilizzarlo.
- Conoscere le principali istruzioni di DDL e di DML.
- Conoscere le principali istruzioni per la gestione delle viste e per la sicurezza dei dati.

ABILITÀ

- Saper utilizzare il linguaggio SQL.
- Saper definire lo schema.
- Saper costruire le query.
- Saper effettuare operazioni complesse.
- Saper garantire la sicurezza dei dati.

COMPETENZE

- Scegliere dispositivi e strumenti in base alle loro caratteristiche funzionali.
- Gestire progetti secondo le procedure e gli standard previsti dai sistemi aziendali di gestione della qualità e della sicurezza.

1. DEFINIRE LO SCHEMA

SQL (**Structured Query Language**) è un linguaggio che consente di inserire, ricercare, aggiornare, cancellare i dati memorizzati in una base di dati di tipo relazionale.

Come linguaggio per le basi di dati relazionali, SQL fa parte dei **linguaggi non procedurali**. Per "non procedurale" si intende che gli utenti devono soltanto specificare quali dati desiderano e non come individuarli. Java e C sono, invece, esempi di linguaggi procedurali.
SQL può essere utilizzato sia in modo interattivo (quando l'utente digita il comando, quest'ultimo viene immediatamente eseguito) sia in modo *embedded*, cioè all'interno di programmi scritti con linguaggi tradizionali (per esempio C#). I risultati delle istruzioni in questo caso non sono immediatamente visibili all'utente, ma vengono elaborati dal programma ospite.
SQL nasce come prototipo a metà degli anni Settanta del Novecento (SEQUEL), ma solo negli anni Ottanta si afferma come linguaggio per il software dei database relazionali (DB2 e SQL/DS). Nel 1986 l'ANSI (*American National Standards Institute*), cioè l'associazione americana che si occupa di definire gli standard nazionali, adottò SQL come standard per i linguaggi relazionali; nel 1987 SQL divenne anche standard ISO. Il linguaggio SQL è stato implementato da numerosi produttori e ovviamente esistono differenze tra un'implementazione e l'altra, ma tutte sono comunque aderenti agli standard internazionali. Attualmente, tra i prodotti più comuni che implementano SQL, vi sono Oracle, Informix, Microsoft Access, SQL Server, MySQL.
Il linguaggio SQL è progettato per:

- creare e modificare schemi di database (DDL - *Data Definition Language*);
- inserire, modificare e gestire dati memorizzati (DML - *Data Manipulation Language*);
- interrogare dati memorizzati (DQL – *Data Query Language*);
- creare e gestire strumenti di controllo e accesso ai dati (DCL - *Data Control Language*);
- creare e gestire le transazioni (TCL - *Transaction Control Language*).

Creazione di database

Per creare un database si usa la seguente sintassi:

```
CREATE DATABASE database_name
```

Volendo fare un esempio pratico, per creare il database per la gestione delle auto di una concessionaria possiamo digitare:

```
CREATE DATABASE dbAuto
```

Per eliminare un database esistente si usa il comando:

```
DROP DATABASE database_name
```

LO SAI CHE

Utilizzando linguaggi di programmazione (Java, C#, PHP, ASPX) si possono scrivere programmi che popolano una tabella, prelevando i dati memorizzati in un file di testo.

Creazione di tabelle

Create table

Per **creare una tabella** in cui registrare dati si usa l'istruzione CREATE TABLE:

```
CREATE TABLE nome-tabella
  (nome-colonna1 tipo1 {NOT NULL}
  ……
  nome-colonnaN tipoN)
```

nome-colonna1 e nome-colonnaN sono i **nomi delle colonne** della tabella. Accanto a ogni nome di colonna deve essere specificato il **tipo di dati** e, se necessario, altre **clausole**.

Le principali clausole che possono essere definite sono:

- PRIMARY KEY definisce il campo come chiave primaria (univoca);
- FOREIGN KEY definisce il campo come chiave straniera (esterna);
- UNIQUE indica che il campo non può avere valori duplicati;
- NOT NULL indica che il campo deve avere sempre un valore.

Tipi di dati

In SQL possono essere definiti dati numerici (interi e reali), alfanumerici (stringhe e date) e oggetti. Esistono poi tipi standard per l'utilizzo.

Ogni implementazione di SQL, pur nel rispetto delle norme generali, introduce piccole variazioni o nuovi tipi. Vediamo quelli più utilizzati e normalmente validi per quasi tutti i sistemi, ricordando che alcuni tipi possono essere abbreviati nella loro definizione (per esempio CHAR – CHARACTER, INT – INTEGER).

Tipo	Nome	Contenuto
Numerico	SMALLINT	Numero intero tra – 32768 e 32767 (2 byte)
	INT	Numero intero tra – 2147483648 e 2147483647 (4 byte)
	FLOAT	Numero reale in singola precisione (4 byte)
	DOUBLE	Numero reale in doppia precisione (4 byte)
	DECIMAL(p,q)	Numero con un massimo di *p* cifre di cui *q* cifre dopo il punto decimale. Occupa una dimensione che può variare da 5 a 17 byte a seconda del valore di *p*
Logico	BOOLEAN	TRUE , FALSE
Testo	CHAR(n)	Stringa di massimo *n* caratteri (dove *n* è maggiore di 0 e minore di 255); ogni valore della colonna richiede *n* caratteri
	VARCHAR(n)	Stringa di massimo *n* caratteri (dove *n* è maggiore di 0 e minore di 2048 secondo il prodotto SQL); ogni valore della colonna richiede uno spazio pari al numero di caratteri effettivi del campo
	MEMO	Stringa fino a 65536 caratteri
Data	DATE	Data nella forma AAAA/MM/GG
	TIME	Ora nella forma HH:MM

Si deve comunque tener presente che i tipi di dati e le relative occupazioni di memoria possono variare da un DBMS all'altro.

Nell'esempio è mostrata l'istruzione per creare una tabella relativa ad alcuni giocatori.

```
CREATE TABLE Giocatori
(Id_G           INT PRIMARY KEY,
Cognome         VARCHAR(15)     NOT NULL,
Sesso           CHAR(1)         NOT NULL,
Indirizzo       VARCHAR(25)
Punti           INT,
Cod_Squadra     CHAR(3),
FOREIGN KEY(Cod_Squadra) REFERENCES Squadra(Id_Squadra))
```

Ogni colonna NOT NULL deve riportare un valore in ogni riga; quindi ogni giocatore deve avere un Cognome, ma non necessariamente un Indirizzo. SQL accetta NULL come possibile valore per una colonna.
Tale valore può essere paragonato a "valore sconosciuto" oppure "valore non specificato" e non deve essere confuso con il numero zero o con gli spazi.
La clausola FOREIGN KEY indica che il campo Cod_Squadra è una chiave esterna e si riferisce alla chiave primaria Id_Squadra della tabella Squadra (Squadra(Id_Squadra)). Per definire la chiave esterna alcuni DBMS accettano anche la seguente sintassi:

```
Cod_Squadra     CHAR(3) REFERENCES Squadra(Id_Squadra)
```

I **vincoli di integrità referenziale** consentono di definire le operazioni eseguite da SQL quando un utente tenta di eliminare o aggiornare una chiave alla quale fa riferimento una chiave esterna.
Le clausole REFERENCES di CREATE TABLE supportano le direttive:

- [ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }];
- [ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }].

- ON DELETE NO ACTION (è il valore predefinito): specifica che se si tenta di eliminare una riga contenente una chiave a cui fanno riferimento chiavi esterne presenti in righe in altre tabelle, verrà generato un errore e la riga non verrà eliminata.
- ON UPDATE NO ACTION: specifica che se si tenta di aggiornare un valore di chiave in una riga e alla chiave fanno riferimento chiavi esterne in righe esistenti in altre tabelle, verrà generato un errore e verrà annullata la modifica apportata con UPDATE.
- ON DELETE CASCADE: specifica che se si tenta di eliminare una riga contenente una chiave a cui fanno riferimento chiavi esterne in righe esistenti in altre tabelle, verranno eliminate anche tutte le righe contenenti tali chiavi esterne.
- ON UPDATE CASCADE: specifica che se si tenta di aggiornare un valore di chiave in una riga e a tale valore fanno riferimento chiavi esterne in righe esistenti in altre tabelle, tutti i valori che compongono la chiave esterna verranno anch'essi aggiornati al nuovo valore specificato per la chiave.

È possibile fornire a una colonna un valore predefinito che sarà utilizzato quando, al momento di inserire i dati nella tabella, non verrà specificato un valore con la clausola DEFAULT "valore" dopo la dichiarazione del tipo di dati.

Nella tabella Giocatori dell'esempio precedente assegniamo il valore predefinito "M" al campo sesso con la seguente sintassi:

```
Sesso CHAR(1) default "M";
```

Inoltre, per verificare che il valore da inserire nel campo rientri in un certo range o abbia un determinato formato, si utilizza la clausola CHECK. Si scrive:

```
punti int CHECK (punti >= 0),
```

Per il campo punti saranno ritenuti validi valori maggiori o uguali a 0.

Creare un indice

Per reperire uno o più record in una tabella, SQL normalmente effettua un **accesso sequenziale alla tabella** leggendo una riga per volta. Se si cerca una sola riga e la tabella è composta da numerose righe, questo metodo richiede ovviamente molto tempo ed è quindi poco efficiente.

Per velocizzare la ricerca è necessario effettuare un **accesso diretto ai dati**, ma per poterlo fare bisogna definire un **indice** sul campo di ricerca.

L'istruzione CREATE INDEX serve per creare un indice su una tabella esistente. La sua sintassi è la seguente:

```
CREATE UNIQUE INDEX nome-indice
    ON nome-tabella(nome-colonna)
```

L'uso degli indici è molto importante per rendere **più veloci le operazioni di interrogazione della base di dati** anche se, durante l'elaborazione, è SQL a stabilire se utilizzare o meno uno degli indici presenti.
Quando si aggiornano, si inseriscono o si cancellano righe di una tabella, SQL deve **rivedere gli indici** definiti sulla stessa. Ciò significa che, se da un lato le operazioni di ricerca sono più veloci, dall'altro il tempo di elaborazione delle istruzioni di modifica dei dati è maggiore.
L'opzione UNIQUE specifica che non si desiderano valori duplicati per l'indice; quando è omessa, i duplicati sono ammessi.
È possibile definire più di un indice sulla stessa tabella. Se si prevede di effettuare frequentemente ricerche utilizzando due o più campi alla volta, è possibile creare un indice per quella **combinazione di campi** (indice multi-colonna) inserendo i nomi dei campi separati dalla virgola all'interno della clausola ON:

```
CREATE UNIQUE INDEX nome-indice
ON nome-tabella(nome-colonna1, nome-colonna2)
```

Come utilizzare SQL

Per scrivere le query SQL possiamo utilizzare Microsoft Access nel seguente modo:

1. nella finestra principale di Microsoft Access selezionare da menu *Crea* e scegliere *Struttura query* (fig. 1);

fig. 1

2. dalla finestra *Mostra tabella* non inserire alcuna tabella, ma selezionare *Chiudi* (fig. 2);

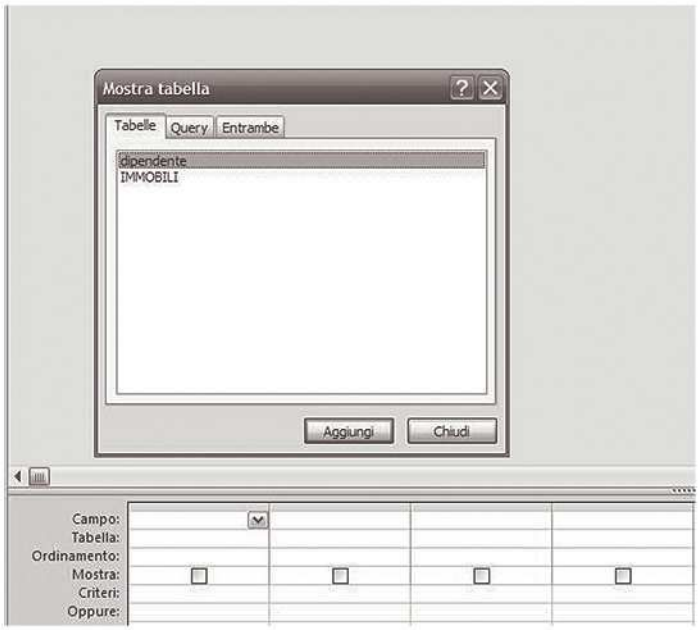


fig. 2

3. dal menu scegliere *Home* e poi *SQL* (fig. 3);

fig. 3

4. si apre una finestra dove è possibile scrivere il comando SQL (fig. 4). Una volta terminato è possibile salvare la query ed eseguirla come una normale query Access.

fig. 4

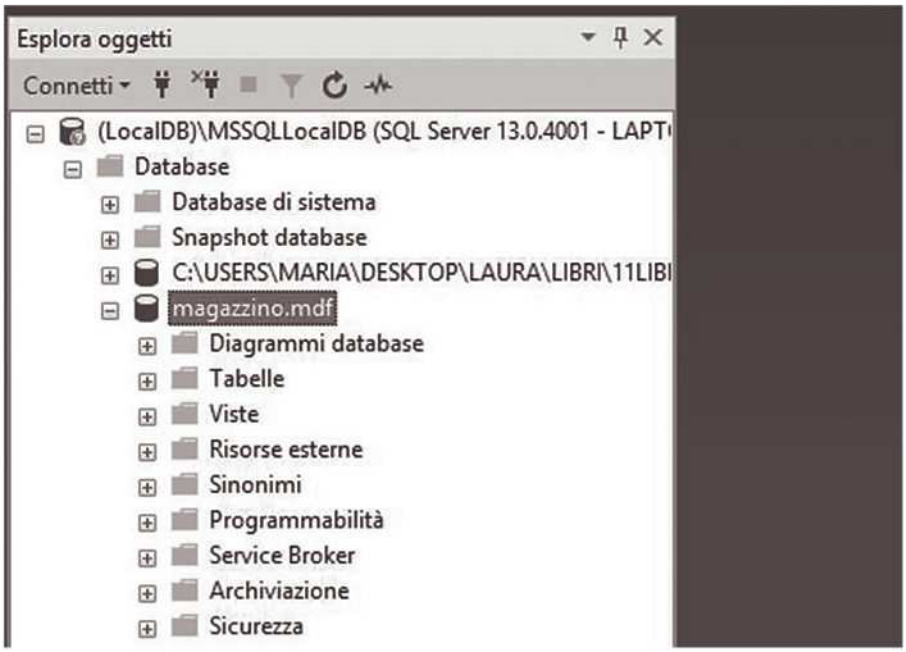
Per scrivere le query in SQL Server sono disponibili diversi strumenti, tra cui:

- SQL Operations Studio;
- SQL Server Management Studio (SSMS);
- MSSQL-cli, che è uno strumento da riga di comando interattivo per le query di SQL Server.

Vediamo come si usa SQL con SQL Server Management Studio, che dispone di una buona interfaccia utente (GUI):

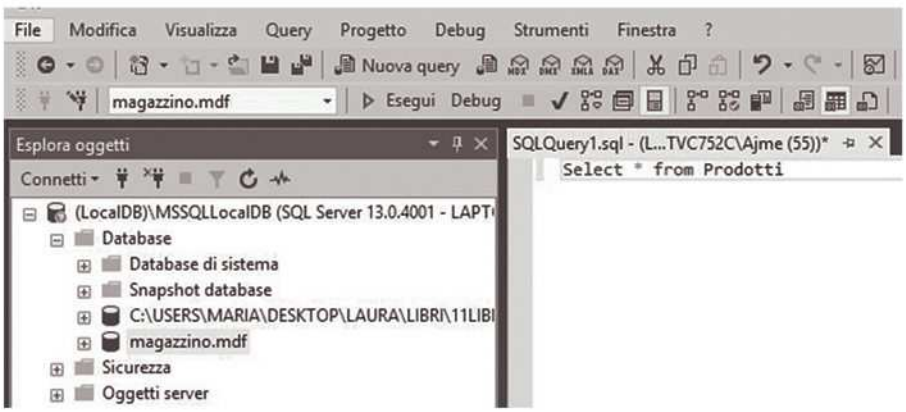
1. dopo aver selezionato il nome del server nella pagina iniziale di SSMS (che in Microsoft SQL Server 2017 è (LocalDB)\MSSQLLocalDB), compare una videata in cui è possibile scegliere il nome del database su cui lavorare (fig. 5):

fig. 5



2. selezionare il pulsante *Nuova query* per scrivere il comando SQL nella parte di destra della finestra;
3. selezionare il pulsante *Esegui* per eseguire la query (fig. 6).

fig. 6



Per scrivere le query in MySQL possiamo utilizzare PhpMyAdmin:

1. dopo aver selezionato il database dalla pagina iniziale di PhpMyAdmin, selezionare il pulsante *SQL* (fig. 7);

fig. 7

2. scrivere il comando SQL;
3. selezionare il pulsante *Esegui*, per eseguire la query (fig. 8).

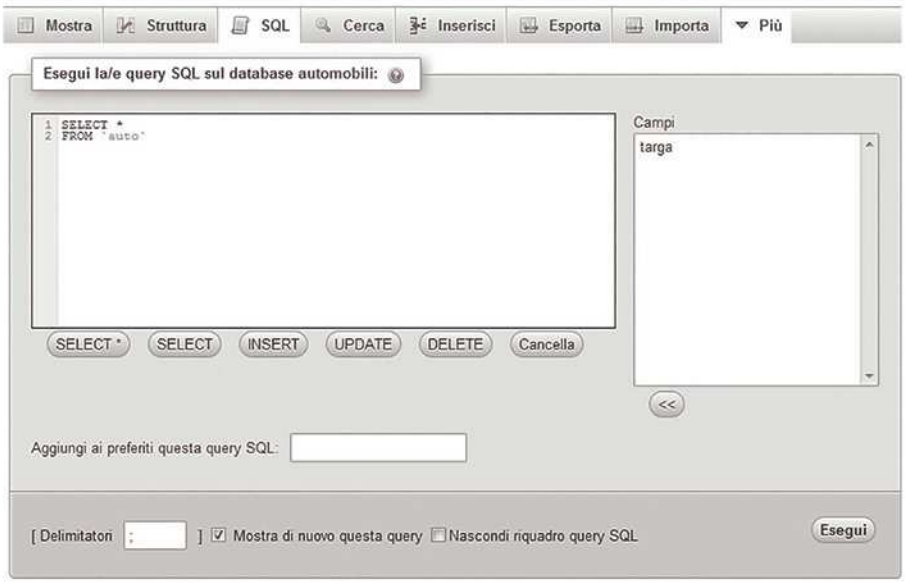


fig. 8

La base di dati per gli esercizi guidati

Negli esercizi che seguono, faremo riferimento alla base di dati DbAuto, ideata per gestire le automobili vendute. Il suo schema logico è:

- auto (targa, cilindrata, prezzo, modello, colore, alimentazione);
- modello (nomeModello, marca);
- marca (nomeMarca, sede, nazione).

targa	cilindrata	prezzo	modello	colore	alimentazione
AB001LX	1600	14000	Xsara	grigio	diesel
AB342CH	999	9000	Panda	rosso	benzina
BA666LL	1350	45000	33	blu	benzina
CB401ET	2000	15000	Punto	blu	diesel
CB453ER	1250	13000	Micra	bianco	benzina
CF786JY	1350	11000	Clio	verde	diesel
DH306HD	1350	13000	Punto	rosso	diesel
EF786JY	1600	14000	Clio	nero	benzina

Modello

nomeModello	marca
33	Alfa Romeo
Clio	Renault
Micra	Nissan
Panda	Fiat
Punto	Fiat
Xsara	Citroën
600	Fiat
Mito	Alfa Romeo

Marca

nomeMarca	sede	nazione
Alfa Romeo	Milano	Italia
BMW	Monaco	Germania
Citroën	Parigi	Francia
Fiat	Torino	Italia
Nissan	Tokyo	Giappone
Renault	Parigi	Francia

LABORATORIO

IL PROBLEMA Crea la tabella Auto che ha la seguente struttura:

Nome	Descrizione	Tipo	Lunghezza	Chiave
Targa	numero di targa	stringa	8 byte	primaria
Cilindrata	cilindrata	numerico	2 byte	
Prezzo	prezzo in euro	numerico	4 byte	
Modello	modello	stringa	20 byte	
Colore	colore	stringa	20 byte	
Alimentazione	tipo di alimentazione (diesel, benzina...)	stringa	10 byte	

Dopo averla creata, definisci:
- un indice sul campo Modello;
- un indice multi-colonna sui campi Marca e Colore.

L'ANALISI Per prima cosa occorre creare una tabella inserendo i campi previsti e definendo, oltre a una chiave primaria, due chiavi esterne.

LA CODIFICA SQL

```
CREATE TABLE Auto
(targa              VARCHAR (8) NOT NULL UNIQUE PRIMARY KEY,
 cilindrata         INT,
 prezzo             FLOAT,
 modello            VARCHAR(20),
 colore             VARCHAR(20),
 alimentazione      VARCHAR(10),
 FOREIGN KEY (modello) REFERENCES Modello(nomeModello))
```

oppure:

```
CREATE TABLE Auto
(targa          VARCHAR (8) NOT NULL UNIQUE PRIMARY KEY,
 cilindrata     INT,
 prezzo         FLOAT,
 modello        VARCHAR(20) REFERENCES Modello(nomeModello),
 colore         VARCHAR(20),
 alimentazione  VARCHAR(10))
```

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione

Dopodiché occorrerà definire un indice sul campo Modello della tabella Auto tramite il comando CREATE INDEX. Poiché potranno esistere più auto con lo stesso modello, non verrà utilizzata la clausola UNIQUE.

LA CODIFICA SQL

```
CREATE INDEX IndMod
Auto(modello)
```

Infine bisognerà definire l'indice multi-colonna sempre usando l'istruzione CREATE INDEX, ma utilizzando i campi Marca e Colore.
Poiché potranno esistere più auto con la stessa marca e lo stesso colore, non verrà utilizzata la clausola UNIQUE.

LA CODIFICA SQL

```
CREATE INDEX IndMod
ON Auto(modello,colore)
```

FISSA LE CONOSCENZE

1. Che cosa indica l'acronimo SQL?
2. Che differenza c'è tra un linguaggio procedurale e uno non procedurale?
3. Che effetto ha la clausola NOT NULL su una colonna?
4. A che cosa si riferisce la clausola REFERENCES di CREATE TABLE?
5. Come si definisce una chiave esterna?
6. A che cosa servono gli indici sui campi di ricerca?

2. MODIFICARE LO SCHEMA DI UNA BASE DATI

Lo schema dei dati può essere modificato sia **eliminando tabelle** o **indici**, sia modificando la **struttura stessa della tabella**.

Modificare la struttura di una tabella

L'istruzione ALTER TABLE permette all'utente di modificare la struttura di una tabella. La clausola successiva indica il tipo di modifica che si vuole effettuare sulla tabella (aggiunta, eliminazione e modifica).
Per inserire una colonna in una tabella già creata, si usa:

```
ALTER TABLE identificatore-tabella
ADD nome-colonna tipo
```

Per esempio, il comando:

```
ALTER TABLE Giocatori
ADD Nazione CHAR(6)
```

inserisce il campo Nazione nella tabella Giocatori.

Per cancellare una colonna da una tabella già esistente, si usa:

```
ALTER TABLE identificatore-tabella
DROP nome-colonna
```

Per esempio, il comando:

```
ALTER TABLE Giocatori
DROP Nazione
```

cancella il campo Nazione nella tabella Giocatori.

Per modificare una colonna cambiandone il tipo, si usa:

```
ALTER TABLE identificatore-tabella
CHANGE nome-colonna nome-colonna nuovotipo
```

Per esempio:

```
ALTER TABLE Giocatori
CHANGE Punti Punti DECIMAL(3,2)
```

Eliminare tabelle e indici

Per cancellare una tabella si usa il comando:

```
DROP TABLE identificatore-tabella
```

Il comando DROP elimina sia la struttura della tabella sia tutte le righe in essa contenute.
Per cancellare un indice definito su una tabella, si usa:

```
DROP INDEX nome-indice ON identificatore-tabella
```

LABORATORIO

IL PROBLEMA Inserisci il nuovo campo km (kilometri percorsi dall'auto) nella tabella Auto descritta nella Lezione 1.

L'ANALISI Occorre modificare la tabella utilizzando l'istruzione ALTER TABLE e indicando il nuovo campo da inserire (km).

LA CODIFICA SQL

```
ALTER TABLE Auto
ADD km INT
```

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione	km

LABORATORIO

IL PROBLEMA Cancella dalla tabella Auto il campo km.

L'ANALISI Occorre modificare la tabella utilizzando l'istruzione ALTER TABLE e indicando il campo da eliminare (km).

LA CODIFICA SQL

```
ALTER TABLE Auto
DROP km
```

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione

FISSA LE CONOSCENZE

1. Quale comando si usa per aggiungere un campo in una tabella?
2. Quale comando si usa per eliminare una tabella?
3. Quale comando si usa per eliminare un campo di una tabella?
4. Quale comando si usa per modificare la lunghezza di un campo in una tabella?
5. Quale comando si usa per eliminare un indice?

3. MODIFICARE I DATI

Nelle lezioni precedenti abbiamo esaminato i comandi SQL che implementano le funzioni di DDL (*Data Definition Language*) ovvero quelle istruzioni che permettono di progettare, modificare e distruggere le strutture che contengono i dati. Esaminiamo ora la parte di SQL che assolve alle funzioni del DML (*Data Manipulation Language*) per inserire, modificare e cancellare i dati che sono memorizzati nelle tabelle.

Inserimento

È possibile inserire nuove righe in una tabella usando il comando:

```
INSERT INTO identificatore-tabella
VALUES([costante1, costante2…])
```

Per esempio, per inserire una nuova riga nella tabella Giocatori, definita nella Lezione 1, si scrive:

```
INSERT INTO Giocatori
VALUES (10, 'Rossi', 'M', 'Via Torino 10', 4, 'Sq1')
```

Se si vuole inserire una riga nella tabella riga, ma non tutti i campi hanno un valore, bisogna specificare i nomi dei campi di cui si conosce il valore:

```
INSERT INTO identificatore-tabella
(nome-campoX, nomecampoY…) VALUES (valoreX, valoreY…)
```

Se, per esempio, si vuole inserire un nuovo giocatore del quale si conoscono solo il codice, il cognome e il codice della squadra in cui gioca, occorre scrivere:

```
INSERT INTO Giocatori
(idG, Cognome, CodSquadra) VALUES (14, 'Bruni', 'sq3')
```

Alcuni ambienti, come per esempio MySQL e SQL Server, danno la possibilità di inserimenti multipli, vale a dire di inserire più record utilizzando un solo comando INSERT:

```
INSERT INTO Giocatori
(idG, cognome, sesso, indirizzo, punti, codSquadra)
VALUES
(12, 'Verdi', 'F', 'Via Genova 22', 15, sq2),
(15, 'Bianchi', 'M', 'Via Roma14', 5, sq1)
```

Modifica

Per modificare i valori contenuti in una tabella, si usa l'istruzione UPDATE:

```
UPDATE identificatore-tabella
SET nome-colonna = espressione
{WHERE condizione}
```

Quando è specificata la clausola WHERE, la modifica interessa solo quelle righe della tabella che soddisfano il criterio di selezione. In sua assenza, la modifica avrà effetto su tutte le righe della tabella.

Per modificare l'indirizzo del giocatore Rossi si ha:

```
UPDATE Giocatori
SET Indirizzo = 'Via Milano 20'
WHERE Cognome = 'Rossi'
```

Per incrementare di 1 il punteggio di tutti i giocatori, si scrive:

```
UPDATE Giocatori
SET Punti = Punti + 1
```

Cancellazione

È possibile eliminare le righe di una tabella con l'istruzione DELETE:

```
DELETE
    FROM identificatore-tabella
    {WHERE condizione}
```

Anche in questo caso la presenza della clausola WHERE permette di eliminare solo i record che rispettano certe condizioni. L'assenza della clausola WHERE provoca lo svuotamento (non la cancellazione) di tutta la tabella.

Per cancellare le informazioni relative al giocatore Rossi, si scrive:

```
DELETE
    FROM Giocatori
    WHERE Cognome = 'Rossi'
```

LABORATORIO

IL PROBLEMA Inserisci i dati di una nuova automobile nella tabella Auto.

L'ANALISI Occorre ricordare che i dati saranno inseriti nei campi della tabella Auto nell'ordine in cui si presentano nella clausola VALUES. I dati alfanumerici devono inoltre essere racchiusi tra apici.

LA CODIFICA SQL

```
INSERT INTO Auto
(targa, cilindrata, prezzo, modello, colore, alimentazione)
VALUES ('CD564BA', 1200, 11000, 'Punto', 'bianco', 'benzina')
```

Al termine dell'esecuzione di questo comando non viene mostrato nulla. Per verificare il corretto inserimento dei dati, bisogna aprire la tabella Auto.

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione
AB001LX	1600	14000	Xsara	grigio	diesel
AB342CH	999	9000	Panda	rosso	benzina
BA666LL	1350	45000	33	blu	benzina
CB401ET	2000	15000	Punto	blu	diesel
CB453ER	1250	13000	Micra	bianco	benzina
CF786JY	1350	11000	Clio	verde	diesel
CD564BA	1200	11000	Punto	bianco	benzina
DH306HD	1350	13000	Punto	rosso	diesel
EF786JY	1600	14000	Clio	nero	benzina

LABORATORIO

IL PROBLEMA Modifica il colore dell'auto con targa CB453ER.

L'ANALISI Setta a rosso il campo colore del record in cui Targa = 'CB453ER'.

LA CODIFICA SQL

```
UPDATE Auto
SET colore = 'rosso'
WHERE targa = 'CB453ER'
```

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione
AB001LX	1600	14000	Xsara	grigio	diesel
AB342CH	999	9000	Panda	rosso	benzina
BA666LL	1350	45000	33	blu	benzina
CB401ET	2000	15000	Punto	blu	diesel
CB453ER	1250	13000	Micra	rosso	benzina
CF786JY	1350	11000	Clio	verde	diesel
CD564BA	1200	11000	Punto	bianco	benzina
DH306HD	1350	13000	Punto	rosso	diesel
EF786JY	1600	14000	Clio	nero	benzina

LABORATORIO

IL PROBLEMA Incrementa del 10% il prezzo delle auto Punto.

L'ANALISI Individua il modello e poi effettua un calcolo sul campo prezzo aggiornandolo.

LA CODIFICA SQL

```
UPDATE Auto
SET prezzo = prezzo*1.1
WHERE modello='Punto'
```

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione
AB342CH	999	9000	Panda	rosso	benzina
BA666LL	1350	45000	33	blu	benzina
CB401ET	2000	16500	Punto	blu	diesel
CB453ER	1250	13000	Micra	rosso	benzina
DH306HD	1350	14300	Punto	rosso	diesel
EF786JY	1600	14000	Clio	nero	benzina

* alcune righe sono state omesse

LABORATORIO

IL PROBLEMA Cancella l'auto con targa AB001LX.

L'ANALISI Cancella dalla tabella Auto il record relativo all'auto in cui Targa = 'AB001LX'.

LA CODIFICA SQL

```
DELETE *
FROM AUTO
WHERE Targa = 'AB001LX'
```

Viene chiesta conferma dell'intenzione di cancellare un record. Al termine dell'esecuzione di questo comando non viene mostrato nulla; per verificare la cancellazione, bisogna aprire la tabella Auto e si noterà che non c'è più l'auto con targa AB001LX.

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione
AB342CH	999	9000	Panda	rosso	benzina
BA666LL	1350	45000	33	blu	benzina
CB401ET	2000	22000	Punto	blu	diesel
CB453ER	1250	13000	Micra	rosso	benzina
CF786JY	1350	16500	Clio	verde	diesel
CD564BA	1200	11000	Punto	rosso	benzina
DH306HD	1350	30000	Punto	rosso	diesel
EF786JY	1600	15400	Clio	nero	benzina

4. L'ISTRUZIONE SELECT

Il DQL (*Data Query Language* – linguaggio di interrogazione dei dati) comprende i comandi per **leggere** ed **elaborare** i dati presenti in un database strutturato secondo il modello relazionale.

L'istruzione SQL che consente di **interrogare la base di dati** è **SELECT**: il suo risultato è una tabella che successivamente può essere ancora elaborata. Il formato completo dell'istruzione SELECT è:

```
SELECT identificatore-colonna
FROM identificatore-tabella
{WHERE condizione}
{GROUP BY [nome-colonna]…}
{HAVING condizione}
{ORDER BY [identificatore-colonna]…}
```

Questa istruzione permette di selezionare, dalla tabella specificata nella clausola FROM, tutte le righe che soddisfano la condizione specificata nella clausola WHERE. Il significato delle altre opzioni (GROUP BY, HAVING, ORDER BY) sarà analizzato in seguito.
Per esempio, per selezionare due campi (Cognome e Indirizzo) appartenenti alla tabella Giocatori, l'istruzione è la seguente:

```
SELECT Cognome, Indirizzo
FROM Giocatori
```

L'utilizzo del carattere * indica la selezione di tutti i campi. Con l'istruzione:

```
SELECT *
FROM Giocatori
```

vengono selezionati tutti i campi appartenenti alla tabella Giocatori.

LABORATORIO

IL PROBLEMA Visualizza la targa, il modello e il prezzo di tutte le auto.

L'ANALISI Nel comando SELECT inserisci il nome delle colonne che vuoi visualizzare (**operazione di proiezione**).

LA CODIFICA SQL

```
SELECT targa, modello, prezzo
FROM Auto
```

IL RISULTATO

targa	modello	prezzo
AB001LX	Xsara	14000
AB342CH	Panda	9000
BA666LL	33	45000
CB401ET	Punto	15000
CB453ER	Micra	13000
CF786JY	Clio	11000
DH306HD	Punto	14300
EF786JY	Clio	14000

Clausola WHERE

La clausola WHERE definisce la **condizione** a cui devono sottostare le righe da ricercare. Una condizione può essere di confronto semplice (sono utilizzabili i classici operatori =, <, >, >=, <=, <>) o di confronto composto quando sono presenti gli operatori AND, OR, NOT.

```
SELECT Cognome, Indirizzo
FROM Giocatori
WHERE Sesso = 'M'
```

In questo esempio viene selezionato il cognome e l'indirizzo dei maschi (Sesso = 'M') presenti nella tabella Giocatori.

LABORATORIO

IL PROBLEMA Visualizza tutte le auto con prezzo minore di 20.000 euro.

L'ANALISI Nel comando SELECT imposta la condizione WHERE Prezzo < 20000 che specifica il criterio di selezione (**operazione di selezione**).

LA CODIFICA SQL

```
SELECT *
FROM Auto
WHERE prezzo < 20000
```

IL RISULTATO

targa	cilindrata	prezzo	modello	colore	alimentazione
AB001LX	1600	14000	Xsara	grigio	diesel
AB342CH	999	9000	Panda	rosso	benzina
CB401ET	2000	15000	Punto	blu	diesel
DH306HD	1350	14300	Punto	rosso	diesel
EF786JY	1600	14000	Clio	nero	benzina
CF786JY	1350	11000	Clio	verde	diesel
CB453ER	1250	13000	Micra	bianco	benzina

Il **predicato** della clausola WHERE può essere anche:

```
LIKE         : è similare al valore
NOT LIKE     : non è similare al valore
BETWEEN      : è compreso tra i due valori specificati
NOT BETWEEN  : non è compreso tra i due valori specificati
IS NULL      : è il valore null
IS NOT NULL  : non è il valore null
IN           : corrisponde a uno dei valori specificati
NOT IN       : non corrisponde ad alcuno dei valori specificati
```

LIKE

L'operatore LIKE permette un **confronto con stringhe** che contengono valori speciali, detti **caratteri jolly**, che sono i simboli percento '%' e underscore '_':

- _ (underscore) per indicare un qualsiasi carattere singolo in quella posizione della stringa;
- % (percento) per indicare una sequenza qualsiasi di caratteri in quella posizione della stringa.

N.B.: in Access, i caratteri % e _ sono sostituiti rispettivamente da * e ?.
Per esempio, se si vogliono ottenere tutti gli elementi che iniziano per una certa lettera si utilizza l'operatore LIKE come segue:

```
SELECT Nome_prodotto, Prezzo
FROM Prodotti
WHERE Nome_prodotto LIKE 'P%'
```

Il risultato è la visualizzazione dei campi Nome_prodotto e Prezzo, dei record appartenenti alla tabella Prodotti, ma solo quelli in cui Nome_prodotto inizia con la lettera P seguita da un qualunque insieme di caratteri (in Access il carattere % dovrebbe essere sostituito da *).
Vediamo un altro esempio. La query:

```
SELECT Nome, Cognome FROM Giocatori WHERE Nome LIKE 'Francesc_'
```

restituisce rispettivamente le righe della tabella Giocatori, in cui il campo Nome inizia per Francesc ed è seguito da una singola lettera: quindi include sia Francesco sia Francesca, ma non Franceschino.

LABORATORIO

IL PROBLEMA Visualizza la targa, il modello e il prezzo per tutte le auto la cui targa presenta la lettera B in seconda posizione.

L'ANALISI Nel comando SELECT imposta la clausola LIKE sul campo Targa. Le targhe estratte potranno presentare qualsiasi carattere in prima posizione (_, o ? in Access), la lettera B in seconda posizione seguita da una qualsiasi sequenza di caratteri (%, o * in Access).

LA CODIFICA SQL

```
SELECT targa, modello,marca, prezzo
FROM Auto
WHERE targa Like '_B%';
```

in ambiente Access:

```
SELECT targa, modello, prezzo
FROM Auto
WHERE targa LIKE '?B*';
```

IL RISULTATO

targa	modello	prezzo
AB001LX	Xsara	14000
AB342CH	Panda	9000
CB401ET	Punto	15000
CB453ER	Micra	13000

BETWEEN

L'operatore BETWEEN controlla se un valore è **compreso all'interno di un intervallo di valori**, inclusi gli estremi. È possibile specificare, anteponendolo a BETWEEN, anche l'operatore logico NOT per valutare la condizione opposta, cioè per controllare se il valore non rientra nell'intervallo specificato. Nota che il campo può essere di qualsiasi tipo: numerico, stringa, data… .

```
SELECT Nome_prodotto, Prezzo
FROM Prodotti
WHERE Prezzo BETWEEN 10 and 15
```

Nell'esempio illustrato vengono visualizzati i campi Nome_prodotto e Prezzo della tabella Prodotti, selezionando solo i prodotti che hanno il prezzo unitario compreso tra 10 e 15 euro.

LABORATORIO

IL PROBLEMA Visualizza la targa, il modello e il prezzo delle auto con prezzo compreso tra 10.000 e 13.000 euro.

L'ANALISI Puoi risolvere questo problema utilizzando l'operatore BETWEEN all'interno del comando SELECT.

LA CODIFICA SQL

```
Select targa, modello, prezzo
From Auto
WHERE prezzo BETWEEN 10000 and 13000
```

IL RISULTATO

targa	modello	prezzo
DH306HD	Punto	13000
CF786JY	Clio	11000
CB453ER	Micra	13000

Operatori logici

Come già accennato, le condizioni possono essere **complesse** se vengono congiunte condizioni semplici con gli operatori AND, OR e NOT. Nel seguito sono mostrati due esempi di condizioni complesse.

```
SELECT Nome_prodotto, Prezzo
FROM Prodotti
WHERE Nome_prodotto LIKE 'C%' AND Prezzo < 15
```

Visualizza i campi Nome_prodotto e Prezzo, appartenenti alla tabella Prodotti, selezionando solo i record che hanno il nome che incomincia con la lettera C e un prezzo inferiore ai 15 euro.

```
SELECT Nome_prodotto, Prezzo
FROM Prodotti
WHERE (Nome_prodotto LIKE 'C%' OR
       Nome_prodotto LIKE 'P%') AND Prezzo < 10
```

Visualizza i campi Nome_prodotto e Prezzo, appartenenti alla tabella Prodotti, selezionando solo i record che hanno il nome che inizia con la lettera C o con la lettera P (criterio OR) e prezzo inferiore a 10 euro (criterio AND).

IN

L'operatore IN controlla **se un valore appartiene a un insieme specificato di valori**.
Anche IN può essere preceduto da NOT per indicare la condizione opposta, cioè la non appartenenza del valore all'insieme dei valori.

```
SELECT *
FROM Fornitori
WHERE Città IN ('Londra', 'Roma', 'Parigi')
```

Il risultato di questa query visualizza tutti i campi della tabella Fornitori, selezionando solo i record residenti a Londra, Roma o Parigi.

DISTINCT

Per **eliminare i valori ripetuti** viene utilizzata la clausola DISTINCT. I record duplicati verranno mostrati una sola volta.

```
SELECT DISTINCT Ruolo
FROM Giocatori
```

Il risultato dell'esempio visualizza tutti i ruoli dei giocatori presenti nella tabella Giocatori: le righe duplicate vengono eliminate.

LABORATORIO

IL PROBLEMA Visualizza tutte le cilindrate presenti nel database.

L'ANALISI Si tratta di visualizzare **una sola volta** il valore di un campo che si ripete all'interno della tabella. Per visualizzare le cilindrate una sola volta eliminando i duplicati occorre inserire nel comando SELECT la clausola DISTINCT.

LA CODIFICA SQL

```
SELECT DISTINCT (cilindrata)
FROM Auto
```

IL RISULTATO

cilindrata
999
1250
1350
1600
2000

Ordinare i risultati di una query

La clausola ORDER BY **ordina i dati visualizzati** secondo uno o più campi specificati, in ordine crescente o decrescente. L'ordinamento è crescente per default; se lo si vuole decrescente, bisogna specificare DESC accanto agli attributi sui quali si vuole eseguire l'ordinamento.
Nell'esempio seguente il risultato della query sarà una tabella con i nomi e i relativi prezzi dei prodotti ordinati in modo crescente in base al nome del prodotto.

```
SELECT Nome_prodotto, Prezzo
FROM Prodotti
ORDER BY Nome_prodotto
```

Nella clausola ORDER BY possono essere specificati più campi: in questo caso l'ordinamento avviene sul primo campo e nel caso in cui diverse righe presentino lo stesso valore si ha anche l'ordinamento sul secondo campo.
Scrivendo, per esempio, "ORDER BY cognome,nome" si ha l'ordinamento in base al cognome e, a parità di cognome, si ha l'ordinamento sul campo nome.

LABORATORIO

IL PROBLEMA Visualizza la targa, il modello e il colore per tutte le auto di modello Xsara o Punto o Clio ordinate per targa.

L'ANALISI Nel comando SELECT imposta la condizione WHERE per estrarre le righe che presentano il valore di modello compreso nell'insieme 'Xsara , Punto , Clio'.
Con l'aggiunta della clausola ORDER BY le targhe sono ordinate alfabeticamente.

LA CODIFICA SQL

```
SELECT targa, modello, colore
FROM Auto
WHERE modello IN ('Xsara','Punto','Clio')
ORDER BY targa
```

La condizione:

```
modello IN ( 'Xsara','Punto','Clio')
```

è equivalente a:

```
modello = 'Xsara' or modello = 'Punto' or modello = 'Clio'
```

IL RISULTATO

targa	modello	colore
AB001LX	Xsara	grigio
CB401ET	Punto	blu
CF786JY	Clio	verde
DH306HD	Punto	rosso
EF786JY	Clio	nero

FISSA LE CONOSCENZE

1. Quando si usa l'istruzione SELECT?
2. Quali dati posso essere estratti con l'istruzione SELECT * FROM Auto?
3. A quali condizioni è equivalente la clausola NOT BETWEEN?
4. A quali condizioni è equivalente il predicato NOT IN?
5. A che cosa servono e quali sono i caratteri jolly?
6. Quando si usa la clausola DISTINCT?

5. ALTRI USI DELL'ISTRUZIONE SELECT

Nelle applicazioni può capitare di dover fare calcoli al volo sui campi dei record estratti: per esempio, se occorre visualizzare un prezzo convertito in vari cambi (euro, dollari, sterline...), oppure quando bisogna calcolare una percentuale o una frazione.
Vediamo due esempi.

LABORATORIO

IL PROBLEMA Visualizza la targa, il modello e il prezzo in euro e in dollari di tutte le auto Clio con prezzo maggiore di 1.200 euro.

L'ANALISI Per risolvere questo problema occorre inserire una **condizione doppia** nella clausola WHERE del comando SELECT, in modo da estrarre dalla tabella tutte le righe che soddisfano le condizioni richieste. Inoltre, per calcolare il prezzo in dollari devi moltiplicare il prezzo per 1.17 e assegnare il nome Dollari al campo calcolato utilizzando la clausola AS.

LA CODIFICA SQL

```
SELECT targa, modello, prezzo, prezzo * 1.17 AS dollari
FROM Auto
WHERE modello = 'Clio' and prezzo > 12000
```

IL RISULTATO

targa	modello	prezzo	dollari
EF786JY	Clio	14000	16380

LABORATORIO

IL PROBLEMA Visualizza la targa, il modello, il prezzo e il prezzo aumentato del 10% per tutte le auto la cui targa presenta la lettera B in seconda posizione, che sono di colore blu o rosso e che hanno prezzo compreso tra 10.000 e 20.000 euro.

L'ANALISI Nel comando SELECT, bisogna impostare la condizione WHERE per estrarre le righe che presentano targa LIKE '_B%', Colore = 'blu' o Colore = 'rosso' e Prezzo BETWEEN 10000 e 20000. Il nuovo prezzo è calcolato con la formula prezzo + (prezzo * 10/100)

LA CODIFICA SQL

```
SELECT targa, modello, prezzo, (prezzo + prezzo*10/100) AS nuovoPrezzo
FROM Auto
WHERE targa LIKE '_B%'
AND colore IN ('blu','rosso')
AND prezzo BETWEEN 10000 AND 20000
```

In ambiente Access la codifica è la seguente:

```
SELECT targa, modello, prezzo, (prezzo + prezzo*10/100) AS nuovoPrezzo
FROM Auto
WHERE targa LIKE '?B*'
AND colore IN ('blu','rosso')
AND prezzo BETWEEN 10000 AND 20000
```

La condizione:

```
prezzo BETWEEN 10000 AND 20000
```

è equivalente a:

```
prezzo >=10000 AND prezzo = 20000
```

IL RISULTATO

targa	modello	prezzo	nuovoPrezzo
CB401ET	Punto	15000	16500

Generare nuove tabelle

Come abbiamo già detto, il risultato di una query è una tabella di tipo temporaneo, che scompare al termine dell'esecuzione. Se si desidera conservarla, è possibile **creare una tabella permanente** a partire dai risultati della query, usando la clausola INTO.

```
SELECT ListaCampi INTO NuovaTabella
FROM TabellaOrigine
{WHERE condizione}
```

Questa query genera la tabella NuovaTabella che contiene le colonne della tabella TabellaOrigine che compaiono in ListaCampi.
La tabella NuovaTabella potrà essere poi utilizzata all'interno delle query.

LABORATORIO

IL PROBLEMA Genera la tabella TabPunto contenente le targhe delle auto del modello Punto.

L'ANALISI Per risolvere questo problema, utilizza il comando SELECT con la clausola INTO.

LA CODIFICA SQL

```
SELECT targa, modello INTO TabPunto
FROM Auto
WHERE modello = 'Punto'
```

IL RISULTATO

targa	modello
CB401ET	Punto
DH306HD	Punto

6. L'OPERAZIONE JOIN

In un database le informazioni sono generalmente suddivise in più tabelle e, in molti casi, per **ricostruire l'informazione complessiva** occorre valutare insieme le tabelle. L'interrogazione deve quindi coinvolgere più tabelle. Per comprendere appieno come avviene un'interrogazione su più tabelle bisogna aver ben chiaro il concetto di **prodotto cartesiano**.

Il prodotto cartesiano

Il **prodotto cartesiano** è il modo più semplice per **combinare due** (o più) **tabelle diverse**. Esso corrisponde a una regola matematica che mette in relazione gli elementi di un insieme con quelli di un altro.

> Il **prodotto cartesiano di due tabelle** restituisce una tabella con un numero di colonne pari alla somma del numero delle colonne delle tabelle. Ciascuna riga è ottenuta unendo ciascuna riga della prima tabella con ciascuna riga della seconda.

La tabella risultato di questa operazione ha un numero di righe pari al prodotto del numero di righe delle due tabelle.
Facciamo un esempio, ipotizzando di avere le seguenti tabelle.

Nazioni

nome	codContinente
Italia	1
Algeria	2
Francia	1

Continenti

idContinente	nome
1	Europa
2	Africa

Il loro prodotto cartesiano è il seguente:

nome	codContinente	idContinente	nome
Italia	1	1	Europa
Italia	1	2	Africa
Algeria	2	1	Europa
Algeria	2	2	Africa
Francia	1	1	Europa
Francia	1	2	Africa

Il risultato si ottiene prendendo la prima riga di Nazioni e affiancando tutte le righe di Continenti (si ottengono le prime due righe). Poi si ripete per la seconda riga di Nazioni e così via.
Otterremo quindi sei righe (poiché 3 × 2 = 6).

Il comando SQL che produce il prodotto cartesiano è:

```
SELECT *
FROM Nazioni, Continenti
```

JOIN

Non tutte le righe del prodotto cartesiano sono significative: nell'esempio infatti hanno significato solo quelle per cui il campo codContinente è uguale al campo idContinente, perché indicano il nome del continente in cui si trovano le nazioni:

nome	codContinente	idContinente	nome
Italia	1	1	Europa
Italia	1	2	Africa
Algeria	2	1	Europa
Algeria	2	2	Africa
Francia	1	1	Europa
Francia	1	2	Africa

Un modo per **filtrare i dati del prodotto cartesiano** così da ottenere solo le righe che hanno significato è quello di impostare un vincolo nella clausola WHERE:

```
SELECT *
FROM Nazioni, Continenti
WHERE Nazioni.codContinente = Continenti.idContinente
```

La giunzione interna (INNER JOIN) è quella che coincide con la query sopra descritta. Questa query (fondamentale) può essere espressa anche nel seguente modo:

```
SELECT *
FROM Nazioni INNER JOIN Continenti
ON Nazioni.codContinente = Continenti.idContinente
```

Le due query proposte sono equivalenti; la seconda però è più esplicita e più semplice da leggere. In caso di presenza di valori nulli in uno dei due campi non si mostrano le righe del prodotto cartesiano.

Per semplificare la scrittura e per rendere più leggibile il codice, è possibile far uso degli **Alias**. Nell'esempio si può indicare con N la tabella Nazioni e con C la tabella Continenti:

```
SELECT N.nome, C.nome
FROM Nazioni AS N, Continenti As C
WHERE N.codContinente = C.idContinente
```

Riepilogando, le possibili sintassi per un'operazione di INNER JOIN sono:

```
SELECT *
FROM Tabella1, Tabella2
WHERE campoTabella1 = campoTabella2
```

e

```
SELECT *
FROM Tabella1 INNER JOIN Tabella2
ON campoTabella1 = campoTabella2
```

JOIN su più tabelle

Quando le informazioni da elaborare sono contenute su tre o più tabelle, si può estendere la JOIN a più tabelle:

```
SELECT *
FROM Tabella1, Tabella2, Tabella3
WHERE
   campoTabella1 = campoTabella2
AND
   campoTabella2 = campoTabella3
```

Rimane comunque sempre valida la seguente sintassi:

```
SELECT *
FROM Tabella1 INNER JOIN (Tabella2 INNER JOIN Tabella3
                                   ON campoTabella2 = campoTabella3)
ON campoTabella1 = campoTabella2
```

Facciamo subito un esempio pratico, considerando anche la tabella Capitali.

Capitali

nazione	città
Italia	Roma
Algeria	Algeri
Francia	Parigi

Se vogliamo conoscere il nome del continente in cui sono situate le capitali, dobbiamo considerare tutte e tre le tabelle. La tabella Continenti è legata alla tabella Nazioni attraverso il codice del continente, mentre la tabella Nazioni è collegata con la tabella Capitali attraverso il nome della nazione:

```
SELECT Continenti.nome, Capitali.citta
FROM Continenti, Nazioni, Capitali
WHERE Continenti.idContinente = Nazioni.codContinente
AND Capitali.Nazione = Nazioni.Nome
```

LABORATORIO

IL PROBLEMA Visualizza la targa, il modello e la marca di tutte le auto.

L'ANALISI I dati da visualizzare sono contenuti in due tabelle diverse. Pertanto è necessario congiungere le due tabelle interessate (Auto, Modello), usando la chiave esterna Modello per collegare la tabella Auto con la tabella Modello che ha come chiave primaria nomeModello.

LA CODIFICA SQL

```
SELECT targa, modello, marca
FROM Auto, Modello
WHERE Auto.modello = Modello.nomeModello
```

oppure:

```
SELECT targa, modello, marca
FROM Auto INNER JOIN Modello
        ON Auto.modello = Modello.nomeModello
```

IL RISULTATO

targa	modello	marca
AB001LX	Xsara	Citroën
AB342CH	Panda	Fiat
BA666LL	33	Alfa Romeo
CB401ET	Punto	Fiat
DH306HD	Punto	Fiat
EF786JY	Clio	Renault
CF786JY	Clio	Renault
CB453ER	Micra	Nissan

Se invece si volessero conoscere solo i dati delle auto di marca Fiat, basterebbe aggiungere una condizione alla query precedente:

```
SELECT targa, modello, marca
FROM Auto, Modello
WHERE Auto.modello = Modello.nomeModello
AND Modello.marca = 'Fiat'
```

targa	modello	marca
AB342CH	Panda	Fiat
CB401ET	Punto	Fiat
DH306HD	Punto	Fiat

LABORATORIO

IL PROBLEMA Per ogni auto visualizza la targa, il modello, la marca e la nazione dove è stata prodotta.

L'ANALISI I dati da visualizzare sono contenuti in tre tabelle diverse. Pertanto è necessario congiungere le tre tabelle interessate (Auto, Modello, Marca) utilizzando la chiave esterna Modello per collegare le tabelle Auto e Modello e la chiave esterna Marca per collegare le tabelle Modello e Marca.

LA CODIFICA SQL

```
SELECT A.targa, A.cilindrata, A.modello, MR.NomeMarca, MR.sede
FROM Auto AS A, Modello AS ML, Marca AS MR
WHERE A.modello = ML.nomeModello
AND ML.Marca = MR.NomeMarca
```

oppure:

```
SELECT A.targa, A.cilindrata, A.modello, MR.NomeMarca, MR.sede
FROM (Auto AS A INNERN JOIN Modello AS ML
        ON A.modello = ML.nomeModello)
INNER JOIN Marca AS MR ON ML.Marca = MR.NomeMarca
```

IL RISULTATO

targa	cilindrata	modello	NomeMarca	sede
AB001LX	1600	Xsara	Citroën	Parigi
AB342CH	999	Panda	Fiat	Torino
BA666LL	1350	33	Alfa Romeo	Milano
CB401ET	2000	Punto	Fiat	Torino
DH306HD	1350	Punto	Fiat	Torino
EF786JY	1600	Clio	Renault	Parigi
CF786JY	1350	Clio	Renault	Parigi
CB453ER	1250	Micra	Nissan	Tokio

FISSA LE CONOSCENZE

1. Che cos'è il prodotto cartesiano di due tabelle?
2. Quale funzione ha la clausola WHERE in una JOIN?
3. Come si estende una JOIN su più tabelle?
4. Che cosa sono gli alias e a che cosa servono?
5. Quando si presenta la necessità di effettuare un'operazione JOIN tra due o più tabelle?
6. Dal prodotto cartesiano di due tabelle come si ottiene una JOIN?

7. TIPI DI JOIN

Abbiamo visto nella Lezione precedente che quando dobbiamo selezionare dati da due o più tabelle, per avere dei risultati completi occorre effettuare delle operazioni JOIN.

Se con la INNER JOIN selezioniamo tutti i valori che accomunano le tabelle, nella OUTER JOIN avremo come risultato anche le righe che non trovano corrispondenza. A sua volta l'OUTER JOIN può essere di tre tipi:

- LEFT JOIN;
- RIGHT JOIN;
- SELF JOIN.

LEFT JOIN

LEFT JOIN visualizza tutte le righe della prima tabella (quella a sinistra dell'operatore). Inoltre congiunge le righe della prima tabella con quelle della seconda tabella, che hanno valori corrispondenti.

In altre parole, se c'è una corrispondenza tra le due tabelle vengono riportati i dati di entrambe le tabelle, se non c'è vengono comunque riportati i dati della tabella di sinistra.

```
SELECT *
FROM Tabella1 LEFT JOIN Tabella2
ON campoTabella1 = campoTabella2
```

Consideriamo le due tabelle Studenti e Voti.
La prima contiene i dati di alcuni studenti, la seconda i voti degli studenti in alcune prove:

- **Studenti** (matricola, nome, cognome);
- **Voto** (matricola, voto, materia).

Studenti

matricola	nome	cognome
m1	Mario	Rossi
m3	Sara	Bianchi
m4	Andrea	Bruni

Voti

matricola	voto	materia
m1	8	italiano
m1	6	matematica
m3	7	fisica
m3	8	italiano

La query seguente riporta tutti i record della tabella Studenti, anche se gli studenti non hanno avuto valutazioni:

```
SELECT *
FROM Studenti as S LEFT JOIN Voti as V
ON S.matricola = V.matricola
```

S.matricola	nome	cognome	V.matricola	voto	materia
m1	Mario	Rossi	m1	6	matematica
m1	Mario	Rossi	m1	8	italiano
m3	Sara	Bianchi	m3	8	italiano
m3	Sara	Bianchi	m3	7	fisica
m4	Andrea	Bruni	null	null	null

Se poi si vogliono ottenere solo le righe relative agli studenti che non hanno valutazioni, bisogna aggiungere alla query precedente la clausola che estrae le righe per le quali il campo V.matricola è vuoto (IS NULL):

```
SELECT *
FROM Studenti as S Left JOIN Voti as V
ON S.matricola = V.matricola
WHERE V.matricola IS NULL
```

S.matricola	nome	cognome	V.matricola	voto	materia
m4	Andrea	Bruni			

LABORATORIO

IL PROBLEMA Visualizza i nomi dei modelli per i quali non sono state vendute auto.

L'ANALISI Per risolvere questo problema bisogna individuare il nome dei modelli che sono presenti nella tabella Casa, ma non sono presenti nella tabella Auto: si tratta di effettuare un'operazione LEFT JOIN tra le tabelle Auto e Casa e poi selezionare solo le righe che ci interessano (il campo modello della tabella Auto deve essere nullo).

LA CODIFICA SQL

```
SELECT M.nomeModello
FROM Modello AS M LEFT JOIN Auto AS A
ON M.nomeModello = A.modello
WHERE A.modello IS NULL
```

IL RISULTATO

nomeModello
600
Giulietta

RIGHT JOIN

RIGHT JOIN visualizza tutte le righe della seconda tabella (quella a destra dell'operatore), congiungendo con le righe della prima solo le righe della seconda per le quali si trovano valori corrispondenti nella prima tabella.

In altre parole, se c'è una corrispondenza tra le due tabelle vengono riportati i dati di entrambe le tabelle, se non c'è vengono comunque riportati i dati della tabella di destra:

```
SELECT *
FROM Tabella1 RIGHT JOIN Tabella2
ON campoTabella1 = campoTabella2
```

Si possono ottenere i dati degli studenti senza valutazioni con il comando:

```
SELECT *
FROM Voti as V RIGHT JOIN Studenti as S
ON S.matricola = V.matricola
WHERE V.matricola IS NULL
```

V.matricola	voto	materia	S.matricola	nome	cognome
null	null	null	m4	Andrea	Bruni

SELF JOIN

SELF JOIN consente di unire una tabella a se stessa.

Si può utilizzare una SELF JOIN quando si desidera creare un insieme di risultati che unisca record in una tabella ad altri record nella stessa tabella. Per elencare una tabella due volte nella stessa query, è necessario specificare un alias di tabella, come nel seguente codice:

```
SELECT T1.campo1, T2.campo2
FROM tabella_uno AS T1, tabella_uno AS T2
WHERE T1.campo1 = T2.campo2
```

Facciamo un esempio. Supponiamo di avere la seguente tabella Genitori:

genitore	figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

Se si vogliono conoscere i nonni di Anna, si può impostare la query seguente, che comporta impostare gli alias G1 e G2 della tabella Genitori.

```
SELECT G1.Genitore AS Nonni
FROM Genitori AS G1, Genitori AS G2
WHERE G1.figlio = G2.genitore
AND G2.figlio = 'Anna'
```

Il risultato sarà quello mostrato nella tabella.

Nonni
Giorgio
Enzo
Silvia

LABORATORIO

IL PROBLEMA Visualizza il nome e il cognome del capo del dipendente di cognome Verdi. La tabella Dipendenti contiene codice, cognome, nome e codice del capo dei dipendenti di una ditta.

Codice	Cognome	Nome	Codice_capo
1	Rossi	Mario	6
2	Verdi	Luigi	5
3	Bianchi	Maria	6
4	Bruni	Clara	5
5	Neri	Luca	4
6	Conti	Filippo	6

L'ANALISI Per risolvere questo problema bisogna utilizzare un'operazione SELF JOIN, perché il capo di Verdi è a sua volta un dipendente presente nella tabella Dipendenti. Viene quindi eseguita un'operazione SELF JOIN tra la tabella Dipendenti alias T1 e la tabella Dipendenti alias T2. Viene poi selezionata solo la riga cui si è interessati (T1.cognome = 'Verdi').

LA CODIFICA SQL

```
SELECT T2.Nome, T2.Cognome
FROM Dipendenti AS T1, Dipendenti AS T2
WHERE T1.Codice_capo = T2.Codice
AND T1.Cognome = 'Verdi'
```

IL RISULTATO

Nome	Cognome
LUCA	NERI

FISSA LE CONOSCENZE

1. Quali sono i tipi dell'operazione JOIN?
2. Che differenza c'è tra INNER JOIN e OUTER JOIN?
3. Quali dati sono selezionati con la LEFT JOIN?
4. Quale clausola bisogna definire perché siano estratti solo i dati che compaiono nella tabella di destra, ma non in quella di sinistra?
5. Quando si usa la SELF JOIN?
6. Che cosa visualizza l'operazione RIGHT JOIN?
7. Quando è necessaria la clausola IS NULL nelle OUTER JOIN?

8. FUNZIONI DI AGGREGAZIONE

All'interno del comando SELECT possono essere utilizzate alcune funzioni predefinite, che agiscono sui valori contenuti in insiemi di righe della tabella (da cui il nome **funzioni di aggregazione**):

- COUNT() conta il numero di righe di una tabella;
- SUM() calcola la somma dei valori assunti da una colonna di una tabella;
- AVG() calcola la media dei valori assunti da una colonna di una tabella;
- MAX() calcola il massimo dei valori assunti da una colonna di una tabella;
- MIN() calcola il minimo dei valori assunti da una colonna di una tabella.

Funzione COUNT

La funzione COUNT **conta il numero di righe** presenti selezionate. La sua sintassi è la seguente:

```
SELECT COUNT (*)
FROM NomeTabella
[WHERE condizione]
```

oppure:

```
SELECT COUNT (nomeColonna)
FROM NomeTabella
[WHERE condizione]
```

La sintassi del linguaggio SQL richiede di specificare come argomento della funzione il carattere * (asterisco), oppure il nome di un attributo: nel primo caso la funzione calcola il numero delle righe della tabella, incluse quelle con campi di tipo NULL; nel secondo caso non vengono conteggiate le righe che hanno valore NULL nella colonna specificata tra le parentesi.

Il seguente comando restituisce il numero di tutte le righe presenti nella tabella Dipendenti:

```
SELECT COUNT (*)
FROM Dipendenti
```

Specificando invece il nome dell'attributo (cellulare) come argomento della funzione COUNT, si ottiene il numero di persone per le quali è stato memorizzato il numero di cellulare:

```
SELECT COUNT (cellulare)
FROM Dipendenti
```

Se si utilizza una clausola WHERE, la funzione COUNT restituisce il numero delle righe che soddisfano la condizione specificata. Con la query:

```
SELECT COUNT(*)
FROM Dipendenti
WHERE nome = 'Giovanni'
```

viene restituito il numero di dipendenti che si chiamano Giovanni.
Il risultato del conteggio può essere anche descritto con un'opportuna intestazione aggiungendo la clausola AS:

```
SELECT COUNT(*) As 'Nome Giovanni'
FROM Dipendenti
WHERE nome = 'Giovanni'
```

Si può ottenere anche il numero di righe diverse usando DISTINCT.
Il comando seguente restituisce quanti nomi diversi sono presenti nella tabella Dipendenti:

```
SELECT COUNT(DISTINCT nome)
FROM Dipendenti
```

Attenzione: non tutti i DBMS supportano quest'ultima sintassi per la funzione COUNT.

Funzione SUM

La funzione SUM restituisce la **somma di tutti i valori contenuti in una colonna** (naturalmente di tipo numerico) specificata come argomento della funzione.
La sintassi nel linguaggio SQL è:

```
SELECT SUM(nomeColonna)
FROM NomeTabella
```

La query seguente restituisce la somma degli stipendi di tutti i dipendenti:

```
SELECT SUM(stipendio)
FROM Dipendenti
```

Nel caso siano presenti valori NULL, vengono considerati come 0.
Se si utilizza una clausola WHERE, la funzione prenderà in esame solo le righe che soddisfano la condizione specificata.

La seguente interrogazione restituisce la somma degli stipendi di tutti i dipendenti che lavorano nel dipartimento vendite:

```
SELECT SUM(stipendio)
FROM Dipendenti
WHERE dipartimento = 'Vendite'
```

Funzione AVG

La funzione AVG (dall'inglese *average*) calcola la **media aritmetica** dei valori numerici contenuti in una colonna. La sintassi nel linguaggio SQL è:

```
SELECT AVG(nomeColonna)
FROM NomeTabella
```

La funzione non include nel calcolo i valori di tipo NULL presenti nella

colonna.
Nell'esempio viene calcolata la media degli stipendi del reparto vendite:

```
SELECT AVG(stipendio)
FROM Dipendenti
WHERE dipartimento = 'Vendite'
```

Funzione MIN

La funzione MIN restituisce il **valore minimo** tra i valori della colonna.
La sintassi nel linguaggio SQL è:

```
SELECT MIN(nomeColonna)
FROM nomeTabella;
```

Anche in questo caso, specificando la clausola WHERE, si restringe il calcolo alle righe che soddisfano la condizione.

La seguente query estrae lo stipendio minimo per i dipendenti che lavorano nel reparto vendite:

```
SELECT MIN(stipendio) AS 'Stipendio minimo'
FROM Dipendenti
WHERE dipartimento = 'Vendite'
```

L'argomento della funzione MIN può essere anche di tipo stringa:

```
SELECT MIN(cognome)
FROM Dipendenti
```

In questo caso è estratto il primo cognome in ordine alfabetico.

Funzione MAX

La funzione MAX cerca il più grande dei valori numerici contenuti in una determinata colonna di una tabella. La sintassi nel linguaggio SQL è:

```
SELECT MAX(nomeColonna)
FROM nomeTabella;
```

La seguente query estrae lo stipendio massimo per i dipendenti che lavorano nel reparto vendite:

```
SELECT MAX(stipendio) AS 'Stipendio massimo'
FROM Dipendenti
WHERE dipartimento = 'Vendite'
```

È anche possibile utilizzare più funzioni di aggregazione contemporaneamente.
Con la seguente query vengono estratti i valori massimo e minimo degli stipendi, nonché lo stipendio medio dei dipendenti della ditta:

```
SELECT MAX(stipendio), MIN(stipendio), AVG(stipendio)
FROM Dipendenti
```

LABORATORIO

IL PROBLEMA Stampa il numero totale delle auto.

L'ANALISI Per risolvere questo problema va utilizzata la funzione di aggregazione COUNT. Il risultato di questa query è una tabella che ha un'unica riga e un'unica colonna (Numero).

LA CODIFICA SQL

```
SELECT COUNT(*) AS Numero
FROM Auto
```

IL RISULTATO

Numero
8

LABORATORIO

IL PROBLEMA Visualizza la media dei prezzi, il prezzo più alto e quello più basso.

L'ANALISI Per risolvere questa interrogazione vengono utilizzate le funzioni AVG(Prezzo), MAX(Prezzo) e MIN(Prezzo). Il risultato di questa query è una tabella che ha una riga e tre colonne: Media, Massimo e Minimo.

LA CODIFICA SQL

```
SELECT AVG(prezzo) AS Media, MAX(prezzo) AS Massimo, MIN(prezzo) AS Minimo
FROM Auto
```

IL RISULTATO

Media	Massimo	Minimo
16750	45000	9000

LABORATORIO

IL PROBLEMA Visualizza la somma dei prezzi delle auto Fiat.

L'ANALISI Viene utilizzata la funzione di aggregazione SUM(prezzo) insieme alla clausola WHERE marca = 'Fiat'. Il risultato di questa query è una tabella che ha una sola riga e una sola colonna, che si chiama Somma.

LA CODIFICA SQL

```
SELECT SUM(prezzo) AS Somma
FROM Auto, Modello
WHERE Auto.modello = Modello.nomeModello
AND Modello.Marca = 'Fiat'
```

IL RISULTATO

Somma
37000

9. RAGGRUPPAMENTI

La clausola GROUP BY **raggruppa le righe in base ai valori uguali** delle colonne specificate. Questa opzione produce una riga di risultato per ogni raggruppamento.

```
SELECT elencoCampi
FROM NomeTabella
[WHERE condizione]
GROUP BY elencoCampi
[ORDER BY elencoCampi]
```

Scrivendo quindi:

```
SELECT reparto
FROM Dipendenti
GROUP BY reparto
```

si ottiene una tabella che ha tante righe quanti sono i reparti contenuti nella tabella Dipendenti.
In effetti, nella sua forma di utilizzo più semplice, la clausola GROUP BY produce un risultato analogo a SELECT DISTINCT. Lo stesso risultato si sarebbe ottenuto scrivendo:

```
SELECT DISTINCT reparto
FROM Dipendenti
```

GROUP BY e le funzioni di aggregazione

La clausola GROUP BY viene di frequente usata con le funzioni di aggregazione: insieme consentono di effettuare calcoli su campi numerici raggruppati, per estrarre il valore medio, massimo, minimo, la somma, ecc.
Per conoscere il numero di dipendenti per ciascun reparto scriviamo:

```
SELECT codReparto, COUNT (*)
FROM Dipendenti
GROUP BY codReparto
```

Questa query fornisce per ogni reparto il codice e il numero di dipendenti che lavorano nel reparto. La tabella risultante ha tante righe quanti sono i reparti e due colonne (codice del reparto e numero dipendenti).

In alcuni ambienti, come per esempio Access e SQL Server, nella clausola GROUP BY devono comparire tutti i nomi dei campi che compaiono accanto alla parola SELECT. Scrivendo:

```
SELECT codReparto, nomeReparto COUNT (*)
FROM Dipendenti
GROUP BY codReparto
```

viene generato un errore, perché il campo nomeReparto non compare nella clausola GROUP BY.

All'interno della query possono comparire più funzioni di aggregazione. Per esempio, scrivendo:

```
SELECT codReparto, MAX(stipendio), MIN(stipendio),
FROM Dipendenti
GROUP BY codReparto
```

viene visualizzato lo stipendio massimo e minimo per ciascun reparto.

LABORATORIO

IL PROBLEMA Visualizza, per ogni modello di auto, la media dei prezzi.

L'ANALISI Viene utilizzata la funzione di aggregazione AVG(prezzo) che, combinata alla clausola GROUP BY modello, calcola la media dei prezzi delle auto per ogni modello. Il risultato di questa query è una tabella che ha tante righe quanti sono i modelli e due colonne: modello e Media.

LA CODIFICA SQL

```
SELECT modello, AVG(prezzo) AS Media
FROM Auto
GROUP BY modello
```

IL RISULTATO

modello	Media
33	45000
Clio	12500
Micra	13000
Panda	9000
Punto	14000
Xsara	14000
Renault	22000

LABORATORIO

IL PROBLEMA Per ogni marca di automobile visualizza il numero di esemplari venduti.

L'ANALISI Si usa la funzione COUNT che, in JOIN tra le tabelle Auto e Modello e combinata alla clausola GROUP BY marca, determina il numero di auto vendute per ogni modello.

LA CODIFICA SQL

```
SELECT Modello.marca, Count(*) As Numero
FROM Auto, Modello
WHERE Modello.nomeModello = Auto.modello
GROUP BY Modello.marca
```

oppure:

```
SELECT Modello.marca, Count(*) As Numero
FROM Auto INNER JOIN Modello
ON Modello.nomeModello = Auto.modello
GROUP BY Modello.marca
```

IL RISULTATO

marca	Numero
Alfa Romeo	1
Citroën	1
Fiat	3
Nissan	1
Renault	2

LABORATORIO

IL PROBLEMA Visualizza quanto è stato incassato per la vendita delle auto, raggruppando dati per nazione e ordinamento in ordine decrescente di importo.

L'ANALISI Viene utilizzata la funzione SUM all'interno di un'operazione JOIN tra le tabelle Auto, Modello e Marca. Con la clausola GROUP BY si ottengono poi i raggruppamenti per nazione, mentre, con la clausola ORDER BY SUM(Prezzo) DESC, i dati sono ordinati secondo valori decrescenti di incasso totale.

LA CODIFICA SQL

```
SELECT nazione, SUM(prezzo) As Incasso
FROM Auto, Modello, Marca
WHERE Modello.nomeModello = Auto.modello
AND Modello.marca = Marca.nomeMarca
GROUP BY nazione
ORDER BY SUM(prezzo) DESC
```

oppure:

```
SELECT nazione, SUM(prezzo) As Incasso
FROM (Auto INNERN JOIN Modello
        ON Auto.modello = Modello.nomeModello)
INNERN JOIN Marca R ON Modello.Marca = Marca.NomeMarca
GROUP BY nazione
ORDER BY SUM(prezzo) DESC
```

IL RISULTATO

nazione	Incasso
Italia	82000
Francia	39000
Giappone	13000

Clausola HAVING

La clausola HAVING è simile alla clausola WHERE nell'istruzione SELECT, ma, mentre WHERE opera sulla singola riga di una tabella, HAVING agisce su **gruppi di righe**, che prima sono stati selezionati con una clausola GROUP BY.

```
SELECT codReparto, COUNT (*)
FROM Dipendenti
GROUP BY codReparto
HAVING COUNT (*) < 10
```

Questo esempio visualizza il codice dei reparti in cui lavorano meno di dieci dipendenti.

Per decidere se specificare le condizioni usando WHERE o HAVING, la regola è semplice:

- se devi impostare la condizione su un campo della tabella, allora usa la clausola WHERE;
- se devi impostare la condizione su un campo di raggruppamento, la condizione concerne gli insiemi di tuple; allora devi usare la clausola HAVING.

Questo esempio visualizza il codice dei reparti dove lavorano più di 10 dipendenti che hanno lo stipendio maggiore di 1.500 euro. In questo caso si è applicata la clausola WHERE sul campo della tabella stipendio e la clausola HAVING sul campo di aggregazione COUNT(*).

```
SELECT CodReparto, COUNT (*)
FROM Dipendenti
WHERE stipendio > 1500
GROUP BY reparto
HAVING COUNT (*) >10
```

LABORATORIO

IL PROBLEMA Visualizza il numero di auto per i modelli che hanno venduto almeno due esemplari.

L'ANALISI Per risolvere questo problema si utilizza la funzione di aggregazione COUNT con la clausola GROUP BY modello per calcolare il numero delle auto per ogni modello. Poi, applicando al risultato il criterio HAVING, si estraggono solo le righe che soddisfano il criterio di selezione. Il risultato di questa query è una tabella che ha tante righe quante sono i modelli che hanno almeno due auto e due colonne: Modello e Numero.

LA CODIFICA SQL

```
SELECT modello, COUNT(*) AS Numero
FROM Auto
GROUP BY modello
HAVING COUNT(*) > 1
```

L'ESECUZIONE

Modello	Numero
Clio	2
Punto	2

FISSA LE CONOSCENZE

1. A che cosa serve la clausola GROUP BY?
2. Che differenza c'è tra la clausola HAVING e la clausola WHERE?
3. Se in una SELECT sono presenti due campi, quanti campi dovranno comparire nella clausola GROUP BY?
4. È possibile usare la clausola GROUP BY all'interno di un'operazione JOIN?

10. QUERY COMPLESSE

Interrogazioni nidificate

Un'**interrogazione nidificata** o subquery è una query inclusa in un'altra, ovvero un'interrogazione all'interno di altre interrogazioni.

Una subquery restituisce dati (tabelle o dati singoli) necessari all'esecuzione della query a un livello più alto: viene sempre eseguita prima la query più interna e, una volta completata, quelle situate al livello superiore.
Il caso più semplice di subquery è quella che restituisce un singolo valore che sarà confrontato con il risultato fornito dalla query esterna.

```
SELECT Campo1
FROM Tabella1
WHERE Tabella1.Campo1 [operatore di confronto]
        (SELECT CampoY
        FROM Tabella2
        WHERE [condizione])
```

Questa query estrae i valori di Campo1 dalla tabella Tabella1 per i quali è soddisfatta la condizione di confronto con il valore di CampoY della tabella Tabella2. Per esempio la query:

```
SELECT Nome_prodotto, Prezzo
FROM Prodotti
WHERE Prezzo <
        (SELECT AVG(Prezzo)
        FROM Prodotti)
```

visualizza i campi Nome_prodotto e Prezzo appartenenti alla tabella Prodotti, selezionando solo quelli con il prezzo inferiore al prezzo medio di tutti i prodotti. In questo caso, prima è eseguita la query interna che restituisce un singolo valore (il prezzo medio), poi viene eseguita la query esterna che estrae tutti i prodotti che hanno un prezzo inferiore.

LABORATORIO

IL PROBLEMA Visualizza targa, modello e prezzo delle auto il cui prezzo è inferiore al prezzo medio.

L'ANALISI Devi definire una query per estrarre tutte le auto il cui prezzo è minore del risultato di un'altra query, che calcola la media dei prezzi.

LA CODIFICA SQL

```
SELECT targa, modello, prezzo
FROM Auto
WHERE prezzo < (SELECT AVG(prezzo) FROM Auto)
```

IL RISULTATO

targa	modello	prezzo
AB001LX	Xsara	14000
AB342CH	Panda	9000
CB401ET	Punto	15000
DH306HD	Punto	13000
EF786JY	Clio	14000
CF786JY	Clio	11000
CB453ER	Micra	13000

LABORATORIO

IL PROBLEMA Visualizza i dati dell'auto che ha prezzo massimo.

L'ANALISI Questa query usa le query annidate: la query interna determina il prezzo massimo, quella esterna estrae i dati delle auto che hanno il prezzo uguale al prezzo massimo.

LA CODIFICA SQL

```
SELECT A.modello, prezzo
FROM Auto AS A, Modello as M
WHERE A.modello = M.nomeModello
AND A.prezzo = (SELECT MAX(prezzo)
                FROM Auto)
```

IL RISULTATO

modello	prezzo
33	45000

LABORATORIO

IL PROBLEMA Visualizzare per quale modello è stata venduta l'auto diesel più cara.

L'ANALISI Questa query usa query annidate, impostando la condizione alimentazione = 'diesel' sia nella query esterna, sia in quella interna.
Infatti, se eliminassimo la condizione dalla query interna, troveremmo il prezzo maggiore tra tutte le auto e non solo tra quelle alimentate a diesel, mentre se la togliessimo dalla query esterna potrebbe venire inclusa un'auto non diesel, con quel prezzo.

LA CODIFICA SQL

```
SELECT A.modello
FROM Auto AS A, Modello as M
WHERE A.modello = M.nomeModello
AND A.prezzo = (SELECT MAX(prezzo)
                FROM Auto
                WHERE alimentazione = 'diesel')
AND alimentazione = 'diesel'
```

oppure:

```
SELECT A.modello
FROM Auto AS A, Modello as M
ON A.modello = M.nomeModello
WHERE A.prezzo = (SELECT MAX(prezzo)
                  FROM Auto
                  WHERE alimentazione = 'diesel')
AND alimentazione = 'diesel'
```

IL RISULTATO

modello
Punto

FISSA LE CONOSCENZE

1. In quale ordine viene eseguita una query complessa che contiene una query di primo livello e una query annidata?
2. Che cosa può restituire la query interna?
3. È corretta la query: Select Modello, Max(Prezzo) from Auto?
4. Definisci la query per visualizzare il modello e il prezzo dell'auto più costosa.
5. Definisci la query per visualizzare il modello e il prezzo dell'auto meno costosa tra quelle di marca Fiat.
6. Definisci la query per visualizzare quante auto costano meno del costo medio.

11. SUBQUERY COMPLESSE

Nel caso delle subquery è necessario fare una distinzione: esse infatti possono restituire un valore singolo (scalare), una singola riga, una singola colonna, oppure una tabella. Gli esempi della Lezione precedente rientrano nel primo caso, perché la query interna ha sempre restituito **un singolo valore** (la media dei prezzi, il prezzo massimo). Quando invece la subquery restituisce **tanti valori**, per esempio una colonna, essi possono essere usati per fare confronti attraverso gli operatori **IN**, **ANY** e **ALL**.

Predicato IN

Il predicato IN serve a controllare se il valore di un attributo è compreso tra quelli restituiti dalla subquery nidificata.

Il seguente esempio produce l'elenco delle province che appartengono a regioni che hanno più di cinque province:

```
SELECT NomeProvincia, NomeRegione
FROM Provincia
WHERE NomeRegione IN (SELECT NomeRegione
                      FROM Provincia
                      GROUP BY NomeRegione
                      HAVING Count(*) > 5)
```

È possibile utilizzare NOT IN per estrarre solo le righe della tabella principale per le quali nessuna riga della tabella ottenuta con la subquery contiene un valore uguale.

LABORATORIO

IL PROBLEMA Per i modelli per i quali sono state vendute almeno due auto produci l'elenco delle auto (targa, modello e prezzo).

L'ANALISI Questa query può essere svolta utilizzando le query annidate: nella subquery vengono selezionati i modelli per i quali sono state vendute almeno due auto, nella query esterna vengono poi estratte le auto appartenenti a quei modelli.

LA CODIFICA SQL

```
SELECT targa, modello, prezzo
FROM Auto
WHERE modello IN    (SELECT modello
                     FROM Auto
                     GROUP BY modello
                     HAVING Count(*) >= 2)
```

IL RISULTATO

targa	modello	prezzo
CB401ET	Punto	15000
DH306HD	Punto	13000
EF786JY	Clio	14000
CF786JY	Clio	11000

Predicato ANY

Il predicato ANY restituisce **vero** se almeno uno dei valori restituiti dalla subquery soddisfa la condizione. Il predicato ANY invece risulta **falso** se la subquery restituisce un insieme vuoto, oppure se il confronto è falso per ciascuno dei valori restituiti dalla subquery.

Il seguente esempio produce la lista degli operai che guadagnano più di almeno un dirigente.

```
SELECT Nome, Stipendio
FROM Operai
WHERE Stipendio > ANY (SELECT Distinct Stipendio
                       FROM Dirigenti)
```

LABORATORIO

IL PROBLEMA Visualizza le informazioni delle auto del modello Clio con prezzo superiore a quello di almeno un'auto del modello Punto.

L'ANALISI Questa query può essere svolta utilizzando le query annidate: nella subquery vengono estratti i prezzi delle auto del modello Clio, nella query esterna vengono poi estratte le auto del modello Punto con prezzo superiore ad almeno uno dei prezzi selezionati nella subquery.

LA CODIFICA SQL

```
SELECT targa, prezzo, cilindrata
FROM Auto
WHERE modello = 'Clio'
AND prezzo > ANY (SELECT prezzo
                  FROM Auto
                  WHERE modello = 'Punto');
```

Predicato ALL

Il predicato ALL restituisce **vero** se il confronto è vero per ciascuno dei valori restituito dalla subquery, è **falso** se il confronto è falso per almeno uno tra i valori dell'elenco restituito dalla subquery.

Il seguente esempio produce la lista degli operai che non vivono in province in cui vive un dirigente.

```
SELECT Nome, Provincia
FROM Operai
WHERE Provincia <> ALL (SELECT Distinct Provincia
                        FROM Dirigenti)
```

Il predicato NOT IN è equivalente a <> ALL.

Consideriamo le query:

```
SELECT Targa
FROM Auto
WHERE Prezzo > ANY
                (SELECT Prezzo
                FROM Auto)
```

```
SELECT Targa
FROM Auto
WHERE Prezzo >= ALL
                (SELECT Prezzo
                FROM Auto)
```

definite sulla tabella Auto di fig. 9. La tabella in fig. 10 definisce il valore di verità dei predicati ANY e ALL.

Targa	Prezzo
AS111EE	50000
BK446RF	100000
AJ767DH	90000

fig. 9 Tabella Auto

ANY	ALL
F	F
V	V
V	F

fig. 10 Predicati ANY e ALL

- ANY specifica che il risultato del confronto è **vero se è vero per una qualunque riga** del risultato dell'interrogazione.
- ALL specifica che il risultato del confronto è **vero se è vero per tutte le righe del risultato** dell'interrogazione.

LABORATORIO

IL PROBLEMA Visualizza targa e prezzo delle Punto che hanno un prezzo superiore a quello di tutte le CLIO.

L'ANALISI Questa query può essere svolta utilizzando le query annidate: nella subquery vengono estratti i prezzi delle auto del modello Clio, nella query esterna vengono poi estratte le auto del modello Punto con prezzo superiore a tutti i prezzi delle auto del modello Clio selezionato nella subquery.

LA CODIFICA SQL

```
SELECT targa, prezzo, cilindrata
FROM Auto
WHERE modello = 'Punto'
AND prezzo > ALL (SELECT prezzo
                  FROM Auto
                  WHERE modello = 'Clio');
```

IL RISULTATO

targa	prezzo	cilindrata
CB401ET	15000	2000

Predicato EXISTS

Il predicato EXISTS controlla se vengono restituite righe dall'esecuzione della subquery: la condizione di ricerca è vera se la SELECT nidificata produce una o più righe come risultato, è falsa se la subquery restituisce un insieme vuoto.

Il seguente esempio riporta i nomi di tutti i lavoratori che sono anche nella tabella proprietari. Il predicato EXISTS riporta true se l'insieme di righe risultanti dalla subselect ne contiene almento una, se invece l'insieme è vuoto, EXISTS riporta false.

```
SELECT Lavoratore
FROM Aziende WHERE EXISTS
                    (SELECT * FROM Proprietari
                    WHERE Proprietari.Nome = Aziende.Lavoratore)
```

Facendo uso di NOT EXISTS è possibile verificare se la subquery non restituisce alcuna tupla.

LABORATORIO

IL PROBLEMA Visualizza per quali modelli non sono state vendute auto.

L'ANALISI Questa query può essere svolta usando le query annidate collegate con EXISTS, che restituirà un valore booleano TRUE se e solo se la subquery interna seleziona almeno una riga.

LA CODIFICA SQL

```
SELECT M.nomeModello
FROM Modello AS M
WHERE NOT EXISTS (SELECT *
                  FROM Auto AS A
                  WHERE A.modello = M.nomeModello)
```

IL RISULTATO

nomeModello
600
Giulietta

Per conoscere per quali modelli è stata venduta almeno un'auto, usiamo il predicato EXISTS:

```
SELECT M.nomeModello
FROM Modello AS M
WHERE EXISTS (SELECT *
              FROM Auto AS A
              WHERE A.modello = M.nomeModello)
```

IL RISULTATO

nomeModello
Xsara
Panda
Punto
Micra
Clio
33

LABORATORIO

IL PROBLEMA Visualizza, per ogni modello, la percentuale di auto vendute.

L'ANALISI In questo caso la subquery non compare all'interno della clausola WHERE, ma al denominatore del calcolo percentuale. Nella subquery si determina il numero totale di auto vendute.

LA CODIFICA SQL

```
SELECT Modello, Count(*) /
(SELECT Count(*) FROM Auto)
*100 AS Percentuale
FROM Auto
GROUP BY Modello
```

IL RISULTATO

Modello	Percentuale
33	12,5
Clio	25
Micra	12,5
Panda	12,5
Punto	25
Xsara	12,5

LABORATORIO

IL PROBLEMA Per ogni auto, determina lo scostamento tra il suo prezzo e il prezzo medio delle auto dello stesso modello.

L'ANALISI Bisogna sottrarre dal costo di ogni auto il prezzo medio delle auto della stessa categoria. Quest'ultimo è calcolato nella subquery e bisogna impostare due alias sulla tabella Auto (A1 e A2) perché nella subquery si possa far riferimento allo stesso campo (A2.modello) della query esterna (A1.modello).

LA CODIFICA SQL

```
SELECT A1.targa, A1.modello, A1.prezzo,
A1.prezzo - (SELECT AVG(A2.prezzo)
             FROM Auto AS A2
             WHERE A1.modello
             = A2.modello) AS scostamento
FROM Auto AS A
```

IL RISULTATO

targa	modello	prezzo	scostamento
AB001LX	Xsara	14000	0
AB342CH	Panda	9000	0
BA666LL	33	45000	0
CB401ET	Punto	15000	1000
DH306HD	Punto	13000	-1000
EF786JY	Clio	14000	1500
CF786JY	Clio	11000	-1500
CB453ER	Micra	13000	0

12. UNIONE, INTERSEZIONE E DIFFERENZA

Unione (UNION)

Il commando SQL UNION **combina insieme i risultati di due tabelle** o **interrogazioni**. Affinché questo operatore possa essere utilizzato correttamente è necessario che:

- le tabelle siano interrogate sul medesimo numero di colonne;
- le colonne richieste abbiano lo stesso nome;
- le colonne richieste nelle due tabelle abbiano dati omogenei.

La sintassi è la seguente:

```
[Istruzione SQL 1]
UNION
[Istruzione SQL 2]
[Order by campo]
```

LABORATORIO

IL PROBLEMA Date la tabella VenditeNegozio contenente i dati delle vendite effettuate nei negozi e la tabella VenditeInternet contenente i dati delle vendite effettuate tramite Internet, visualizza tutte le date (ordinate in modo decrescente) in cui è stata realizzata una vendita.

Tabella VenditeNegozio

Negozio	Importo	Data
Torino	1500	05-gen
Milano	250	07-gen
Roma	300	08-gen
Venezia	700	08-gen

Tabella VenditeInternet

Data	Importo
07-gen	250
10-gen	535
11-gen	320
12-gen	750

L'ANALISI Per realizzare questa funzione bisogna prelevare i dati dalle due tabelle VenditeNegozio e VenditeInternet con un'operazione UNION.

LA CODIFICA SQL

```
SELECT Data FROM VenditeNegozio
UNION
SELECT Data FROM VenditeInternet
ORDER BY Data DESC
```

IL RISULTATO

Data
12-gen
11-gen
10-gen
08-gen
07-gen
05-gen

Intersezione (INTERSECT)

Come con il comando UNION, anche mediante INTERSECT è possibile intervenire su due istruzioni SQL. La differenza consiste nel fatto che, mentre UNION agisce fondamentalmente come un operatore di tipo OR, in cui il valore viene selezionato, sempre che questo sia presente nella prima o nella seconda istruzione, il comando INTERSECT funziona come un operatore AND, in cui il **valore viene selezionato solo se questo appare in entrambe le istruzioni**.

La sintassi è la seguente:

```
[Istruzione SQL 1]
INTERSECT
[Istruzione SQL 2]
```

LABORATORIO

IL PROBLEMA Utilizzando le tabelle dell'esercizio precedente, trova tutte le date in cui sono state effettuate vendite sia in negozio sia via Internet.

L'ANALISI Per realizzare questa operazione si esegue un'operazione INTERSECT sul risultato delle interrogazioni che forniscono le date dei giorni in cui si sono verificate delle vendite.

LA CODIFICA SQL

```
SELECT Data FROM VenditeNegozio
INTERSECT
SELECT Data FROM VenditeInternet
```

IL RISULTATO

Data
07-gen

Differenza (MINUS)

Questo comando esegue la **differenza tra i risultati prodotti da due interrogazioni**. Mediante questo comando vengono presi tutti i risultati restituiti dalla prima query ai quali vengono sottratti tutti quelli che sono presenti nella seconda query. Se nella seconda istruzione SQL sono inclusi risultati che non sono presenti nella prima istruzione, tali risultati vengono ignorati.
La sintassi è la seguente:

```
[Istruzione SQL 1]
MINUS
[Istruzione SQL 2]
```

In alcuni DBMS può essere utilizzato EXCEPT anziché MINUS. Per un corretto utilizzo, si consiglia di consultare la documentazione relativa al proprio DBMS.

LABORATORIO

IL PROBLEMA Utilizzando le tabelle VenditeNegozio e VenditeInternet di cui sopra, trova tutte le date in cui sono state registrate vendite in negozio, ma non via Internet.

L'ANALISI Per realizzare questa funzione bisogna utilizzare l'operatore MINUS tra la tabella VenditeNegozio e la tabella VenditeInternet, escludendo dalle righe estratte dalla prima interrogazione quelle estratte dalla seconda.

LA CODIFICA SQL

```
SELECT Data FROM VenditeNegozio
MINUS
SELECT Data FROM VenditeInternet
```

IL RISULTATO

Data
05-gen
08-gen

FISSA LE CONOSCENZE

1. Che differenza c'è tra il comando UNION e il comando INTERSECT?
2. In quali circostanze è possibile applicare il comando UNION?
3. Quando è possibile applicare il comando INTERSECT?
4. Quale risultato fornisce il comando MINUS?

13. LE VISTE

SQL riconosce **due tipi di tabelle**:

- le **tabelle base**, che sono le uniche in cui è possibile registrare i dati;
- le **tabelle derivate**, dette anche **viste** (VIEW), che sono formule per combinare in una tabella virtuale i dati contenuti nella tabella base.

Grazie alle viste un utente ha una visione solo parziale dei dati contenuti nella tabella, e non può agire su dati che non lo riguardano.
La creazione di una vista avviene con la seguente istruzione:

```
CREATE VIEW nome-vista AS
        istruzione-select
```

Il **contenuto di una vista** non viene memorizzato, ma **derivato** nel momento in cui la vista compare in un'istruzione. Esso, quindi, **coincide sempre con il contenuto delle tabelle base**: qualsiasi modifica ai dati base si riflette immediatamente sulla vista che accede ai dati.
Per cancellare una vista si ricorre all'istruzione:

```
DROP VIEW nome-vista
```

LABORATORIO

IL PROBLEMA Crea una vista con le auto prodotte a Torino.

L'ANALISI Bisogna creare una vista basata su una selezione e una congiunzione (tutte le auto le cui case costruttrici hanno sede a Torino).

LA CODIFICA SQL

```
CREATE VIEW AutoTorino AS
    SELECT targa, prezzo, modello
    FROM Auto, Modello, Marca
    WHERE Auto.modello = Modello.nomeModello
    AND Modello.Marca = Marca.NomeMarca
    AND Marca.sede = 'Torino'
```

IL RISULTATO

targa	prezzo	modello
AB342CH	9000	Panda
CB401ET	15000	Punto
DH306HD	13000	Punto

LABORATORIO

IL PROBLEMA Usando la vista AutoTorino creata nell'esercizio precedente calcola il prezzo medio.

LA CODIFICA SQL

```
SELECT AVG(prezzo) AS Prezzo_Medio
from AutoTorino
```

IL RISULTATO

Prezzo_Medio
12333,3333333333

LABORATORIO

IL PROBLEMA Visualizza per quale modello sono state vendute più automobili.

L'ANALISI Quando le query sono piuttosto complesse, la creazione di una vista può semplificare il lavoro, come nel caso proposto. Infatti per risolvere la query si può procedere creando prima una vista che contenga per ogni modello il numero di automobili vendute e poi cercare su questa vista il modello che ne ha vendute di più.

LA CODIFICA SQL

```
CREATE VIEW AutoTorino AS
SELECT Modello.marca, count (*) as numero
FROM Auto, Modello
WHERE Auto.modello = Modello.nomeModello
GROUP BY Modello.marca
```

IL RISULTATO

marca	numero
Alfa Romeo	1
Citroën	1
Fiat	3
Nissan	1
Renault	2

e successivamente:

LA CODIFICA SQL

```
SELECT nomeMarca
FROM VistaMarca
WHERE numero = (SELECT MAX(numero) FROM VistaMarca)
```

IL RISULTATO

marca
Fiat

FISSA LE CONOSCENZE

1. Che cosa sono le viste?
2. Con quale comando SQL si crea una vista?
3. Con quale comando SQL si elimina una vista?
4. Che differenza c'è tra una vista e una tabella?
5. Per quale motivo alle volte usare una vista può semplificare il lavoro di interrogazione di una base di dati?
6. Per quale motivo una vista mostra sicuramente i dati aggiornati?
7. I dati di una vista occupano spazio?

14. SICUREZZA DEI DATI

I comandi di tipo DCL (*Data Control Language*) controllano il livello di accesso che gli utenti hanno sugli oggetti del database.

Tali comandi sono:
- GRANT;
- REVOKE.

GRANT

Il comando GRANT permette all'amministratore della base di dati di fornire **accessi diversificati** alle tabelle in funzione del **tipo di utente** che ne fa richiesta.
La sintassi di questa istruzione utilizzata per assegnare determinati privilegi è la seguente:

```
GRANT [SELECT
       INSERT
       DELETE
       UPDATE {nome colonna}
       INDEX
       ALTER
       ALL]
ON identificatore-tabella
TO [nome-utente].... | PUBLIC
```

Il privilegio può essere:
- accordato a un solo utente;
- accordato a più utenti;
- concesso come PUBLIC. In quest'ultimo caso il privilegio è concesso a tutti gli utenti SQL.

REVOKE

L'istruzione REVOKE è concepita per rimuovere i privilegi di un utente assegnati con il comando GRANT.
La sua sintassi, prevedibilmente, è la seguente:

```
REVOKE [SELECT
        INSERT
        DELETE
        UPDATE {nome colonna}
        INDEX
        ALTER
        ALL]
ON identificatore-tabella
FROM [nome-utente].... | PUBLIC
```

LABORATORIO

IL PROBLEMA Definisci i privilegi di accesso in lettura e scrittura per l'utente Rossi alla tabella Auto.

L'ANALISI Vanno definiti tramite il comando GRANT i privilegi per l'utente "Rossi" (INSERT, SELECT, UPDATE) sulla tabella Auto.

LA CODIFICA SQL

```
GRANT INSERT, SELECT, UPDATE
ON Auto
TO Rossi;
```

LABORATORIO

IL PROBLEMA Togli all'utente "Rossi" i privilegi di scrittura sulla tabella Auto.

L'ANALISI Bisogna eliminare tramite il comando REVOKE i privilegi per l'utente "Rossi" (INSERT, UPDATE) sulla tabella Auto.

LA CODIFICA SQL

```
REVOKE INSERT, UPDATE
ON Auto
FROM Rossi;
```

I TRIGGER

Un TRIGGER (**regola attiva**) è un tipo speciale di procedura che viene **eseguita automaticamente** quando si verifica un evento nel database.

Spesso, soprattutto nei sistemi di grandi dimensioni, la modifica di determinati dati richiede un'azione automatica su altri dati registrati sullo stesso database o su database differenti e i TRIGGER (*trigger* in inglese vuol dire "grilletto") sono gli strumenti adatti a questo scopo. I TRIGGER vengono eseguiti, per esempio, quando un utente tenta di modificare dati tramite un comando di aggiornamento (istruzioni INSERT, UPDATE o DELETE) eseguito su una tabella o una vista.
Questi TRIGGER vengono attivati quando viene generato un evento **valido**, indipendentemente dal fatto che esistano o meno righe di tabella interessate.
I TRIGGER possono essere utilizzati in molte situazioni per:

1. **mantenere l'integrità dei dati**, controllando la validità dei valori inseriti;
2. segnalare automaticamente ad altri programmi che **devono essere effettuate azioni** quando vengono apportate modifiche a una tabella o vengono inseriti nuovi record (per esempio: aggiungere il contatto alla *mailing list* quando si registra un nuovo utente, inviare una email di conferma all'utente appena registrato);

3. **creare tabelle** di *auditing* **per registrare i record che vengono modificati o eliminati**, in modo da tenere traccia di chi ha eseguito la modifica e in quale data. Per esempio, può essere utile mantenere la storia delle modifiche apportate allo stipendio dei dipendenti di una ditta: tutte le volte che viene modificato uno stipendio viene registrata in una tabella del database una riga contenente il codice del dipendente, il vecchio stipendio, il nuovo stipendio, la data e l'ora dell'operazione.

Da un punto di vista generale, in un TRIGGER vengono specificati i tre elementi: EVENTO – CONDIZIONE – AZIONE.
A seguito della modifica specificata in EVENTO, se è vera la condizione CONDIZIONE, vengono specificate le azioni di AZIONE. La sintassi per creare un trigger in ambiente MySQL è la seguente:

1. Specifica il nome del TRIGGER.
2. Specifica se il TRIGGER deve essere attivato prima o dopo un'operazione.
3. Specifica l'evento che attiva il TRIGGER.
4. Specifica se il TRIGGER è set-oriented (attivato un'unica volta) oppure tuple-oriented (attivato tante volte quante sono le tuple interessate dall'operazione).
5. Specifica l'eventuale condizione che deve essere verificata per attivare il TRIGGER.
6. Specifica le azioni da eseguire.

```
1. CREATE TRIGGER <Nometrigger>
2. <BEFORE|AFTER>
3. <SELECT|DELETE|INSERT|UPDATE[(<Colonne>)]>
   ON <Nometabella>
4. [FOR EACH STATEMENT|FOR EACH ROW]
5. [WHEN <condition>]
6. <action>
```

Per un TRIGGER in update si può accedere al vecchio valore di attributo utilizzando la sintassi OLD.<colonna> e si può accedere al nuovo attributo utilizzando NEW.<colonna>.
La sintassi di CREATE TRIGGER può variare a seconda del DBMS utilizzato.
Per eliminare un TRIGGER si usa il comando:

```
DROP TRIGGER trigger_name;
```

Per creare un TRIGGER in ambiente SQL Server la sintassi è la seguente:

1. Specifica il nome del TRIGGER.
2. Specifica il nome della tabella.
3. Opzioni che specificano quando il TRIGGER deve scattare: FOR indica un'azione contemporanea all'aggiornamento dei dati; AFTER indica un'azione successiva; INSTEAD OF indica che deve essere eseguita l'istruzione del TRIGGER al posto dell'istruzione SQL predefinita specificata dopo (INSERT, DELETE o UPDATE).
4. Specifica le azioni da eseguire.

```
1. CREATE TRIGGER nomeTrigger
2. ON nome Tabella
3. FOR / AFTER/ INSTEAD OF (INSERT, DELETE o UPDATE)
4. AS azione
```

Nel caso di una operazione di DELETE la tabella *Deleted* contiene le righe che sono state appena eliminate, mentre con una INSERT la tabella *Inserted* contiene le righe appena inserite. Nel caso di una operazione di UPDATE entrambe le tabelle contengono valori, perché la *Deleted* contiene i dati prima della modifica (le vecchie righe), mentre la *Inserted* contiene i dati dopo la modifica (le nuove righe).

LABORATORIO

IL PROBLEMA Crea un TRIGGER che permetta di effettuare il controllo sulla cilindrata quando una nuova auto è inserita nel database, ponendo Cilindrata = 900 se il valore inserito è minore di 0 e Cilindrata = 4800 se, invece, è stato inserito un valore maggiore di 4.800.

L'ANALISI In questo esempio il TRIGGER deve essere eseguito **prima** di inserire la riga nella tabella Auto e, se la cilindrata appena inserita (NEW.Cilindrata) è minore di 900, viene assegnato 900. Lo stesso procedimento viene adottato per mantenere il valore della cilindrata minore di 4.800.

LA CODIFICA MYSQL

```
CREATE TRIGGER NuovaAuto
BEFORE INSERT ON Auto
FOR EACH ROW
BEGIN
    IF NEW.Cilindrata < 0 THEN
        SET NEW.Cilindrata = 900
  END IF
    IF NEW.Cilindrata > 4800 THEN
        SET NEW.Cilindrata = 4800
    END IF
END
```

LA CODIFICA SQL SERVER

```
CREATE TRIGGER NuovaAuto
   ON Auto
   INSTEAD OF INSERT
AS
BEGIN
  DECLARE @targa as VARCHAR(20)
  DECLARE @cilindrata as VARCHAR(20)
  DECLARE @prezzo as float
  DECLARE @modello as VARCHAR(20)
  DECLARE @colore as VARCHAR(20)
  DECLARE @marca as VARCHAR(20)
  DECLARE @alimentazione as VARCHAR(20)
  DECLARE @codProprietario as smallint

  SET @cilindrata = (select cilindrata from inserted)
  SET @targa = (select targa from inserted)
  SET @colore = (select colore from inserted)
  SET @prezzo = (select prezzo from inserted)
  SET @modello = (select modello from inserted)
  SET @marca = (select marca from inserted)
  SET @alimentazione = (select alimentazione from inserted)
  SET @codProprietario = (select codProprietario from inserted)
```

```
if(@cilindrata < 0)
      set @cilindrata = 900

if(@cilindrata > 4800)
set @cilindrata = 4800

INSERT INTO Auto
        Values ( @targa , @cilindrata, @prezzo, @modello, @colore, @marca,
          @alimentazione,@codProprietario )
END;
```

In questo esempio, essendo specificata l'opzione INSTEAD OF INSERT, le istruzioni del TRIGGER vengono eseguite al posto dell'istruzione di INSERT di SQL; pertanto anche nel TRIGGER deve comparire l'istruzione INSERT.

LABORATORIO

IL PROBLEMA Aggiorna la tabella di log, inserendo nella tabella Storia il codice dell'auto, il vecchio prezzo e il nuovo prezzo tutte le volte che si aggiorna il prezzo di un'auto.

L'ANALISI In questo caso il TRIGGER deve essere eseguito dopo che è stata modificata una riga nella tabella Auto; in tal caso, se è stato modificato il prezzo e quindi il nuovo prezzo è diverso da quello vecchio, viene inserita una nuova riga nella tabella Storia contenente il codice, il prezzo vecchio e il prezzo nuovo.

LA CODIFICA MYSQL

```
CREATE TRIGGER AggiornaPrezzo
AFTER UPDATE ON Auto
FOR EACH ROW
BEGIN
    IF NEW.Costo <> OLD.Costo THEN
        INSERT INTO Storia
            Values (OLD.Codice, OLD.Costo, NEW.Costo)
  END IF
END
```

LA CODIFICA SQL SERVER

```
Create TRIGGER [dbo].[AgiornaPrezzo]
ON [dbo].[Auto] AFTER UPDATE
AS
   INSERT INTO Storico (targa, oldColore,newColore)
      (SELECT i.codice, d.costo, i.pre costo xxo
      FROM Inserted i
      INNER JOIN Deleted d ON i.targa = d.targa
      where i.costo <> d.costo)
```

15. LE TRANSAZIONI

Per **transazione** si intende una serie di istruzioni SQL che vengono trattate come se fossero una singola unità.

Esempio di transazione è il trasferimento di una somma da un conto corrente a un altro, che può essere implementato con i seguenti comandi SQL:

```
UPDATE CC
SET Saldo = Saldo – 50
WHERE Conto = 123
```

```
UPDATE CC
SET Saldo = Saldo + 50
WHERE Conto = 235
```

Per realizzare questa operazione sono necessari **due aggiornamenti separati**; i funzionari di banca vorranno essere sicuri o che entrambi questi aggiornamenti si verifichino, o che non si verifichi nessuno di essi: non vorrebbero certamente che, per un guasto di sistema, risultasse nel conto 235 un accredito di 50 euro e nessun addebito sul conto 123. C'è bisogno di una **garanzia** che se qualcosa va male nell'operazione, nessuno dei passi eseguiti abbia effetto.
Raggruppando gli aggiornamenti in una transazione abbiamo questa garanzia, perché una transazione può:

- andare completamente a buon fine;

oppure

- fallire completamente, lasciando la base di dati intatta.

Le istruzioni di una transazione devono essere racchiuse tra i comandi Begin Transaction e End Transaction:

```
Begin Transaction
        UPDATE CC
                SET Saldo = Saldo – 50
                WHERE Conto = 123
        UPDATE CC
                SET Saldo = Saldo + 50
                WHERE Conto = 235
End Transaction
```

Una transazione può avere solo due esiti:

1. **terminare correttamente**: questo avviene solo quando l'applicazione, dopo aver eseguito tutte le operazioni, esegue una particolare istruzione SQL, detta COMMIT, che garantisce la conclusione delle operazioni;
2. **terminare non correttamente** (anticipatamente) e quindi sono possibili 2 casi:
   - la **transazione** determina che è avvenuto un errore e interrompe le operazioni, eseguendo l'istruzione SQL ROLLBACK;
   - il **sistema** non è in grado di garantire la corretta prosecuzione della transazione (per esempio, per un guasto o a causa della violazione di un vincolo); la transazione viene quindi interrotta.

Se per qualche motivo la transazione non può terminare correttamente, il DBMS deve annullare (UNDO) le eventuali modifiche da essa apportate alla base di dati. Per esempio, può capitare che, dopo aver detratto 50 euro dal conto di codice 123, per un errore o in seguito a una cancellazione, il conto di codice 235 non sia presente: in questo caso si verificherebbe una situazione di incongruenza e, quindi, la modifica sul primo conto corrente dovrebbe essere annullata con un'operazione di ROLLBACK.
Il modello di transazioni usato dai DBMS è in realtà più articolato; in particolare è possibile definire dei cosiddetti SAVEPOINT, che vengono utilizzati da una transazione per annullare solo parzialmente il lavoro svolto.

Un SAVEPOINT serve a specificare un preciso punto all'interno di una transazione, giunti al quale sarà poi possibile effettuare un ROLLBACK.

Per definire un SAVEPOINT in MySQL si usa la sintassi:

```
SAVEPOINT <nome savepoint> ON ROLLBACK RETAIN CURSORS
```

e per eseguire un rollback parziale:

```
ROLLBACK TO SAVEPOINT <nome savepoint>
```

In SQL Server i rispettivi comandi sono:

```
SAVE TRANSACTION NomeSavePoint
```

e

```
ROLLBACK TRANSACTION NomeSavePoint
```

I comandi che agiscono sulle transazioni (COMMIT, ROLLBACK E SAVEPOINT) appartengono al TCL (*Transaction Control Language*).

Per chiarire meglio il funzionamento di questi comandi, utilizziamo come esempio la tabella Anagrafica, che presenta la seguente situazione:

Cognome	Stipendio
Rossi	20000
Bianchi	20000

Supponiamo ora di avere le seguenti istruzioni in ambiente MySQL:

```
INSERT INTO Anagrafica
VALUES ('Neri', 35000);
SAVEPOINT sp1;

INSERT INTO Anagrafica
VALUES ('Viola', 35000);
SAVEPOINT sp2;

INSERT INTO Anagrafica
VALUES ('Marrone', 25000);
```

Le stesse istruzioni in ambiente SQL Server si scrivono:

```
INSERT INTO Anagrafica
VALUES ('Neri',35000)
SAVE TRANSACTION SavePointsp1

INSERT INTO Anagrafica
VALUES ('Viola',35000)
SAVE TRANSACTION SavePointsp2

INSERT INTO Anagrafica
VALUES ('Marrone',25000)
```

A questo punto, la nostra tabella si presenterà così:

Cognome	Stipendio
Rossi	20000
Bianchi	20000
Neri	35000
Viola	35000
Marrone	25000

Se ora noi scrivessimo

```
ROLLBACK TO sp1;    (MySQL)
```

```
Rollback Transaction SavePointsp1       (Sql Server)
```

avremmo:

Cognome	Stipendio
Rossi	20000
Bianchi	20000
Neri	35000

Se invece scrivessimo:

```
ROLLBACK TO sp2;    (MySQL)
```

```
Rollback Transaction SavePointsp2       (Sql Server)
```

avremmo:

Cognome	Stipendio
Rossi	20000
Bianchi	20000
Neri	35000
Viola	35000

Se invece scrivessimo:

```
ROLLBACK;
```

avremmo come esito che l'intera transazione sarebbe annullata, quindi la nostra tabella si presenterebbe così:

Cognome	Stipendio
Rossi	20000
Bianchi	20000

Infine, scrivendo:

```
COMMIT;
```

l'intera transazione sarebbe consolidata (*committata*), tutti i SAVEPOINT sarebbero rimossi e, quindi, la nostra tabella si presenterebbe così:

Cognome	Stipendio
Rossi	20000
Bianchi	20000
Neri	35000
Viola	35000
Marrone	25000

FISSA LE CONOSCENZE

1. Che cos'è una transazione?
2. Quali esiti può avere una transazione?
3. Che cosa provoca il comando COMMIT?
4. Che cosa provoca il comando ROLLBACK?
5. Che cosa specifica un SAVEPOINT?

RIPASSIAMO INSIEME

DIDATTICA INCLUSIVA

Creazione di database e tabelle

L'SQL è un **linguaggio non procedurale** utilizzato per creare, visualizzare, modificare tabelle di un **database relazionale**.
Attraverso l'istruzione CREATE TABLE è possibile creare una tabella in cui registrare dati, che possono essere **numerici** (interi e reali), **alfanumerici** (stringhe e date) oppure **oggetti**.
In una tabella è possibile **definire un indice** su un dato attributo, allo scopo di rendere più veloce il reperimento dei dati.
È inoltre possibile definire più indici sulla stessa tabella.
Per farlo si usa l'istruzione CREATE INDEX, che agisce su tabelle esistenti creando l'indice.

Modificare lo schema di una base di dati

Attraverso l'istruzione ALTER TABLE è possibile **modificare la struttura di una tabella** definita in precedenza aggiungendo campi, modificandone le caratteristiche, o cancellandoli.
Per **eliminare** una tabella (oppure un indice creato in precedenza) si usa l'istruzione DROP. Nel caso sia applicato a una tabella, il comando elimina sia la struttura sia tutte le righe in essa contenute.

Modificare i dati

È possibile **modificare il contenuto** dei campi di una tabella, oppure anche inserire o cancellare una riga della tabella stessa.
Per **inserire nuove righe** in una tabella si usa il comando INSERT.
Per **modificare i dati** precedentemente inseriti in un record si usa il comando UPDATE.
Per **cancellare i dati** di un record inserito si usa il comando DELETE.

L'istruzione SELECT

L'istruzione SELECT del linguaggio SQL consente di **interrogare la base di dati**.
La sua sintassi è la seguente:

```
SELECT identificatore-colonna
FROM identificatore-tabella
{WHERE condizione}
```

Una SELECT, quindi, preleva i dati di una colonna da una tabella se una condizione viene soddisfatta.
Il **risultato** di una SELECT è una **tabella** che successivamente può essere ancora elaborata usando la clausola ORDER BY.

Altri usi dell'istruzione SELECT

L'istruzione SELECT può essere usata per **implementare query complesse** e per visualizzare dati ottenuti da semplici calcoli effettuati sui campi della tabella.

JOIN

L'operatore JOIN serve a **unire le righe di due o più tabelle** quando i dati richiesti sono contenuti in tabelle diverse.
Per poter utilizzare questo operatore le tabelle interessate **devono avere un attributo in comune**.

Tipi di JOIN

Quando si devono estrarre dati da due tabelle, ma si vogliono considerare quelli che compaiono nella **prima tabella**, ma non nella **seconda**, si usano le operazioni di LEFT JOIN.
Quando si devono estrarre dati da due tabelle, ma si vogliono considerare quelli che compaiono nella **seconda tabella**, ma non nella **prima**, si usano le operazioni di RIGHT JOIN.
Con una SELF JOIN si uniscono le righe di una tabella ad altre righe della stessa tabella.