Lezione del modulo 7 SQL e database del giorno 04/05/2006 tot. ore 3

Lo Standard SQL

Il linguaggio SQL è un linguaggio non procedurale (o di tipo dichiarativo), divenuto, ormai da tempo, il linguaggio standard per creare, manipolare e interrogare basi di dati relazionali.
Il linguaggio SQL assolve alle funzioni di:

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

SQL può essere usato in:

- Modalità stand-alone o sé stante;
  può essere classificato come interactive query language (linguaggio di interrogazione interattivo). I comandi possono essere inviati al S.O. in modo interattivo o batch. Si usa una interfaccia grafica che guida l'utente. Per eseguire ogni istruzione viene invocato l'interprete SQL. Le interrogazioni complesse vengono composte sfruttando le operazioni dell'algebra relazionale, come trattati nelle precedenti dispense, ottenendo come risultato sempre una relazione.

- Modalità embedded o linguaggio ospite.
  In questo caso, dato che il linguaggio ospite può trattare le tuple anche singolarmente è possibile utilizzare comandi SQL all'interno di istruzioni C, JAVA o C++, che vengono chiamati, in questo caso, linguaggi ospite. Un programma scritto in un linguaggio ospite compilato (che incorpora istruzioni SQL al suo interno) subirà un primo processo di precompilazione seguito da una vera e propria compilazione.

Identificatori e tipi di dati

SQL non è un linguaggio case sensitive. Generalmente ogni istruzione si conclude con ; (punto e virgola).
Ogni identificatore utilizzato per il nome della tabella e dei suoi attributi deve:

- Avere lunghezza massima di 18 caratteri
- Iniziare con una lettera
- Contenere come unico carattere speciale l'underscore (_)

Nella terminologia SQL le relazioni sono chiamate **tabelle**, le tuple **righe** o **registrazioni** e gli attributi sono le **colonne** delle tabelle.
Per riferirsi a un attributo di una tabella la sintassi è la seguente

<nometabella>.<nomeattributo>

Le **costanti stringa** sono rappresentabili utilizzando gli apici (‘ ‘) o doppi apici (“ “)
Gli **operatori** utilizzati nelle espressioni sono:

- Aritmetici +, - , / , *
- Relazionali < , > , <= , >= , <>
- Logici AND , OR, NOT

I **tipi di dato** utilizzabili per gli attributi sono qui riassunti:

tipo	Descrizione	Esempi e/o Range di variabilità
CHARACTER O CHAR	Singolo carattere	"C" oppure "5"
CHARACTER(n) o CHAR(n)	Stringa di caratteri di lunghezza fissa pari a n caratteri	**n varia da 1 a 15000** esempio "ax005"
CHARACTER VARYNG(n) o VARCHAR(n)	Stringa di caratteri di lunghezza variabile con lunghezza massima pari a n	VARCHAR(5) "ER567"
BIT	Singolo bit; tipo booleano	0 corrisponde a false 1 corrisponde a true
BIT(n)	Stringa di bit di lunghezza fissa pari a n	BIT(5) "00110"
BIT VARYNG(n)	String di bit di lunghezza variabile avente lunghezza massima pari a n	BIT VARYNG(7) "0011010"
INTEGER o INT	Numero intero con precisione superiore a SMALLINT che generalmente occupa 4 byte ma dipende dall'implementazione	**-2147483648 a +2147483647**
INTEGER(n) o INT(n)	Numero intero con precisione n (numero massimo di cifre che il numero può contenere)	INT(5)
SMALLINT	Numero intero con precisione inferiore a INTEGER (occupa 2 byte di memoria)	**-32768 a +32767**
DECIMAL(p,d) o DEC(p,d)	Numero reale in fixed point con un numero massimo di cifre prima del punto decimale pari a p e un numero massimo di cifre dopo il punto decimale pari a d. Le dimensioni massime di p e d sono definite dall'implementazione.	DECIMAL(6,2) 100.3 o +0.7 o -.3
REAL	Numero reale in floating point con precisione definita dall'implementazione Generalmente vale 7 per la mantissa	**Da 1E-38 a 1E+38** 10E4 +24.7E-3 -7.897E+12
FLOAT	Numero reale in floating point con precisione definita dall'implementazione Generalmente vale 15 per la mantissa	**Da 1E-38 a 1E+38** 10E4 +24.7E-3 -7.897E+12
FLOAT(p)	Numero reale in floating point con precisione p per la mantissa	**P varia da 1 a 45**
DOUBLE PRECISION	Numero reale in floating point con precisione per la mantissa doppia rispetto alla precisione di un FLOAT	**L'esponente da -127 a +128**
DATE	Data nel formato "AAAA/MM/GG" Oppure "AAAA-MM-GG"	"2006/04/30" "2006-04-30"
TIME	Ora nel formato ora, minuti, secondi e millisecondi.	"10:34:25:42"
TIMESTAMP	Data e orario nel formato anno, mese, giorno, ora, minuti, secondi e millisecondi	"2000/04/30 10:34:25:42"

Si ricordi che questi sono i tipi di dati della versione ISO 9075 e che in alcune versioni tali tipi sono differenti. Ecco perché di seguito ho riportato le tabelle dei tipi che si riferisco a MySQL e ad ACCESS.

MySQL			
Tipo	**Descrizione**		**Dimensione max/formato**
BIGINT	Intero lunghissimo	INTERI	-2^{63} a $+2^{63}-1$
INTEGER	Intero lungo		-2^{31} a $+2^{31}-1$
SMALLINT	Intero		-32768 a +32767
TINYINT	Intero ridotto		-128 a +127
DOUBLE	Reale a doppia precisione	REALI	$\pm 2.225*10^{-308}$ a $\pm 1.798*10^{308}$
FLOAT	Reale a singola precisione		$\pm 1.176*10^{-38}$ a $\pm 3.403*10^{38}$
DECIMAL	Decimale memorizzato come stringa		$\pm 2.225*10^{-308}$ a $\pm 1.798*10^{308}$
DATE	Data in formato AAAA-MM-GG	DATA	Dal 1000-01-01 al 9999-12-31
TIME	Orario in formato HH:MM:SS		HH:MM:SS
DATETIME	Data con ora		aaaa-mm-gg hh:mm:ss
YEAR	Anno		AAAA
CHARACTER	Stringa a lunghezza fissa	STRINGA	Da 0 a 255 caratteri
VARCHAR	Stringa a lunghezza variabile		Da 0 a 255 caratteri
TEXT	Campo testo a lunghezza fissa		Da 0 a 65535 caratteri
MEDIUMTEXT	Campo testo (memo)		16 mb di caratteri – 1
BLOB	Immagini jpeg, bmp, gif oppure file binary in esadecimale o di testo	OGGETTI	Fino a 64 Kb
MEDIUM BLOB			Fino a 16 mb
LONG BLOB			Circa 4 Gb

ACCESS			
Tipo	**Descrizione**		**Dimensione max/formato**
BOOLEAN	Un bit memorizzato in 1 byte	INTERI	0 (false) oppure 1 (true)
BYTE	Intero ridotto		Da 0 a 255
INTEGER	Intero		-32768 a +32767
LONG	Intero Lungo 4 byte		-2147483648 a +2147483647
DOUBLE	Reale a doppia precisione a 8 byte	REALI	-1,79E308 a -14,94E-324 per negativi +4,94E-324 a 1,79E308 per positivi
SINGLE	Reale a singola precisione a 4 byte		-3,40E38 a -1,40E-45 per negativi 1,40E-45 a 3,40E38 per positivi
VARIANT	Con numeri a 16 byte		Vedi DOUBLE
CURRENCY	Decimale a 8 byte per valuta		Da -922.337.203.685.477,5808 A +922.337.203.685.477,5807
DATE	Data 8 byte	DATA	Dal 01/01/100 al 31/12/9999
STRING	Stringa a lunghezza fissa	STRINGA	Da 0 a 255 caratteri
MEMO	Campo testo a lunghezza fissa		Da 0 a 65535 caratteri
VARIANT	Con caratteri		22 + lunghezza
OBJECT	Immagini jpeg, bmp, gif oppure file binary in esadecimale o di testo	OGGETTI	4 byte

Istruzioni DDL di SQL

I comandi di DDL creano e modificano lo schema di una base di dati:

1. Creazione di un database

CREATE DATABASE [**IF NOT EXISTS]** <nomedatabase>
[[DEFAULT] CHARACTER SET <NOMESETCARATTERI>**[,…]]**

Questo comando crea un database di nome <nomedatabase> solo se non esiste già che è compatibile con il set di caratteri <NOMESETCARATTERI> . Lo standard è **latin1.** Le caratteristiche del database vengono salvate nel file db.opt.

MYSQL: il comando è identico
ACCESS: i database devono essere creati all'interno dell'ambiente GUI

2. Cancellazione di un database

DROP DATABASE [IF EXISTS] <nomedatabase>

Il comando seguente mi permette di eliminare definitivamente un database.

MYSQL: il comando è identico
ACCESS: i database devono essere eliminati dal file system.

3. Creare una tabella

MYSQL:
CREATE [IF NOT EXISTS] TABLE <NOMETABELLA>
({nomecampo tipo **[**(lunghezza[1])**] [UNSIGNED[2]] [ZEROFILL[3]] [NOT NULL | NULL[4]]**
[DEFAULT valoredidefault[5]**] [AUTO_INCREMENT[6]] [[PRIMARY] KEY] [UNIQUE]**}**[,…]**
[PRIMARY KEY (nomecampo**)]**
[INDEX [nomeindice**] ((**nomecampo**)]**
[UNIQUE [index**]]**
[FOREIGN KEY [nomeindice**]])**
[TYPE=[HEAP | ISAM | InnoDB | MYISAM]];

> **HEAP**: è una tabella temporanea, in cui I dati risiedono direttamente in memoria. Non deve essere utilizzata per memorizzare i dati in modo permanente in quanto ad ogni riavvio di MySQL server tutti i dati vengono persi. I tempi di accesso sono più bassi rispetto alle tabelle MYISAM. Non possiedono il tipo Text e Blob.
> **InnoDB**: è un tipo di tabella che consente di operare con transazioni e foreign Key non supportate dalle altre tabelle. Il motore di gestione delle tabelle InnoDB mantiene le tabelle entro l'area tabellare che può essere composta anche da diversi file. La dimensione massima è comunque di 64 Tb.
> **MYISAM**: è il tipo di tabella di default. Se creata in modo statico i record sono di lunghezza prefissata. Sono assicurate maggiore velocità di accesso e maggiore sicurezza in caso di crash. Se create in modo dinamico ogni record occupa solo lo spazio necessario. Successivi update possono portare alla frammentazione in più pezzi del medesimo archivio. Se create in modo compresso viene ridotto lo spazio occupato dai file .MYI e .MYD. Le tabelle ottenute sono in solo lettura. La dimensione massima è di 4 Gb.

[1] Determina la lunghezza massima del campo.
[2] I numeri possono essere con segno o senza segno
[3] Riempie di zeri tutta la lunghezza del campo. Usato per i codici.
[4] Determina se il campo può essere lasciato vuoto (NULL) oppure no (NOT NULL).
[5] Determina quale valore o stringa deve contenere il campo prima che avvenga l'immissione di un nuovo dato.
[6] È un campo numerico il cui valore viene incrementato di 1 ad ogni nuovo record. A differenza di Access il valore può essere reinserito se il record viene eliminato.

ACCESS:

CREATE TABLE <NOMETABELLA>
({nomecampo tipo **[**(lunghezza*)**] [NOT NULL][WITH COMPRESSION]**
[[PRIMARY] KEY] [UNIQUE] }
[,…]
[PRIMARY KEY (nomecampo**)]**
[[UNIQUE] INDEX [nomeindice**] ((**nomecampo**)]**
[FOREIGN KEY [nomeindice**]]);**

***** in Access non è possibile indicare, nella creazione di una tabella, la lunghezza di un campo numerico.

4. I vincoli di integrità

Nella definizione di una tabella sono presenti vincoli:

- Per un singolo attributo;

Impostano limitazioni da specificare su singolo attributo.
NOT NULL il corrispondente attributo deve avere necessariamente un valore e non può non rimanere non specificato.
DEFAULT assegna all'attributo il valore di default.

- Per un gruppo di attributi detti anche vincoli di ennupla

Impostano limitazioni da specificare sui valori di più attributi.
PRIMARY KEY indica le colonne che fanno parte della chiave primaria.
UNIQUE indica che i valori degli attributi specificati negli attributi (che non formano una chiave primaria) devono essere distinti all'interno della tabella (formano una chiave candidata).

- Per l'integrità referenziale.

Questi vincoli permettono di creare un legame permanente tra diverse tabelle. È un vincolo mediante un dato presente in un'altra tabella. Il vincolo alla tabella esterna viene detto FOREIGN.

MYSQL: questi vincoli posso essere attivati soltanto sulle tabelle di tipo InnoDB. Se si utilizza una tabella MYISAM, il codice viene letto senza errori ma modificando la tabella l'integrità referenziale non viene verificata.

…
[INDEX (<nomecampotabellainterna>**)],**
[FOREIGN KEY ((<nomecampotabellainterna>**)**
REFERENCES <nometabellamadre> **(**<nomecampochiaveprimaria>**)]**

In questo caso si definiscono due tabelle poste in relazione tra loro. La tabella madre è in relazione con la tabella figlia tramite un campo, chiave primaria nella madre e foreign key nella figlia.

ACCESS: Access permette di eseguire questa operazione tramite l'ambiente GUI, che chiama relazioni.

…
<nomecampotabellainterna> tipo **NOT NULL CONSTRAINT m REFERENCES** <nometabellamadre> **(**<nomecampochiaveprimaria>**)]**

Ma impostare i campi all'integrità referenziale significa anche prevedere cosa succede se il record della tabella madre dovesse essere cancellato.
Le operazioni da eseguire possono essere introdotti in base ai seguenti eventi:

- Aggiornamento (UPDATE)
- Cancellazioni (DELETE)

Le possibili reazioni a questi eventi sono:

- CASCADE propaga la modifica anche alla tabella figlia
- SET NULL annulla l'attributo portandolo al valore NULL
- SET DEFAULT assegna all'attributo il valore di default
- RESTRICT impedisce che la modifica possa avvenire

…
[INDEX (<nomecampotabellainterna>**)],**
[FOREIGN KEY ((<nomecampotabellainterna>**)**
REFERENCES <nometabellamadre> **(**<nomecampochiaveprimaria>**)]**
[ON {DELETE | UPDATE}
{CASCADE | SET NULL | SET DEFAULT | RESTRICT}]

MYSQL:

Esempio:

```
CREATE TABLE mansioni
( id_mansione INT(4) DEFAULT '0'  NOT NULL PRIMARY KEY,
Nome CHAR(20) NOT NULL) TYPE=INNODB;

CREATE TABLE dipendenti
(matricola CHAR(6) DEFAULT '0' NOT NULL PRIMARY KEY,
nome CHAR(20) NOT NULL,
cognome CHAR(20)  NOT NULL,
mansione INT(4) NOT NULL,
INDEX (mansione),
FOREIGN KEY (mansione) REFERENCES mansioni(id_mansione) ON DELETE CASCADE,
UNIQUE(nome,cognome) ) TYPE=INNODB;
```

Nel caso in cui si dovesse cancellare il record id_mansione = 1

Id_mansione	Nome
1	Impiegato
2	Quadro
3	Operaio

matricola	Nome	Cognome	mansione
12345	Alfa	Neri	1
12355	Beta	Verdi	3
12366	Delta	Rossi	1
12377	Gamma	Bianchi	2
12388	Teta	Gialli	1

Il risultato sarebbe:

matricola	Nome	Cognome	mansione
12355	Beta	Verdi	3
12377	Gamma	Bianchi	2

CREATE TABLE mansioni
(id_mansione INT(4) DEFAULT ‘0’ NOT NULL PRIMARY KEY,
Nome CHAR(20) NOT NULL) TYPE=INNODB;

CREATE TABLE dipendenti
(matricola CHAR(6) DEFAULT ‘0’ NOT NULL PRIMARY KEY,
nome CHAR(20) NOT NULL,
cognome CHAR(20) NOT NULL,
mansione INT(4) NOT NULL,
INDEX (mansione),
FOREIGN KEY (mansione) REFERENCES mansioni(id_mansione) ON DELETE SET DEFAULT,
UNIQUE(nome,cognome)) TYPE=INNODB;

Nel caso in cui si dovesse cancellare il record id_mansione = 1

matricola	Nome	Cognome	mansione
12345	Alfa	Neri	0
12355	Beta	Verdi	3
12366	Delta	Rossi	0
12377	Gamma	Bianchi	2
12388	Teta	Gialli	0

CREATE TABLE mansioni
(id_mansione INT(4) DEFAULT ‘0’ NOT NULL PRIMARY KEY,
Nome CHAR(20) NOT NULL) TYPE=INNODB;

CREATE TABLE dipendenti
(matricola CHAR(6) DEFAULT ‘0’ NOT NULL PRIMARY KEY,
nome CHAR(20) NOT NULL,
cognome CHAR(20) NOT NULL,
mansione INT(4) NOT NULL,
INDEX (mansione),
FOREIGN KEY (mansione) REFERENCES mansioni(id_mansione) ON DELETE SET NULL,
UNIQUE(nome,cognome)) TYPE=INNODB;

Nel caso in cui si dovesse cancellare il record id_mansione = 1

matricola	Nome	Cognome	mansione
12345	Alfa	Neri	Null
12355	Beta	Verdi	3
12366	Delta	Rossi	Null
12377	Gamma	Bianchi	2
12388	Teta	Gialli	Null

CREATE TABLE mansioni
(id_mansione INT(4) DEFAULT ‘0’ NOT NULL PRIMARY KEY,
Nome CHAR(20) NOT NULL) TYPE=INNODB;

CREATE TABLE dipendenti
(matricola CHAR(6) DEFAULT ‘0’ NOT NULL PRIMARY KEY,
nome CHAR(20) NOT NULL,
cognome CHAR(20) NOT NULL,
mansione INT(4) NOT NULL,
INDEX (mansione),
FOREIGN KEY (mansione) REFERENCES mansioni(id_mansione) ON DELETE RESTRICT,
UNIQUE(nome,cognome)) TYPE=INNODB;

Nel caso in cui si dovesse cancellare il record id_mansione = 1

Viene segnalato un messaggio di errore

5. La visualizzazione dei database e delle tabelle

Mediante il costrutto SHOW si possono conoscere i nomi dei database esistenti, l'elenco delle tabelle di un database oppure le caratteristiche di una particolare tabella.

SHOW TABLES | DATABASES | COLUMNS
IN | FROM
Nometabella | nomedatabase

SHOW TABLES IN <nomedatabase> visualizza le tabelle in un database
SHOW COLUMNS FROM <nometabella> **IN** <nomedatabase> visualizza le colonne e i tipi
SHOW DATABASES visualizza tutti i database

Se si volesse interrogare una tabella senza specificare il database eseguo due istruzioni:
USE <nomedatabase>;
SHOW COLUMNS <nometabella>

6. La modifica dello schema di una tabella.

La struttura di una tabella può essere modificata mediante il costrutto ALTER.

ALTER TABLE <NOMETABELLA> **ADD | CHANGE | DROP**
[COLUMN] VECCHIONOMECAMPO NUOVONOMECAMPO
[FIRST | AFTER] NOMECAMPO TIPO
[INDEX NOMEINDICE **(**NOMECAMPO**)]**
[PRIMARY KEY NOMEINDICE **(**NOMECAMPO**)**
[UNIQUE (NOMECAMPO**)]**
[FOREIGN KEY NOMEINDICE (NOMECAMPO**)]**
[REFERENCES…]

ALTER TABLE mansioni **ADD COLUMN** data_mansione DATA; aggiunge il campo data_mansione
ALTER TABLE mansioni **DROP COLUMN** data_mansione; elimina il campo data_mansione
ALTER TABLE dipendenti **ADD INDEX** cog_nom(cognome,nome) si crea un indice nella tabella dipendenti
ALTER TABLE mansioni **ADD COLUMN** nato VARCHAR(20) **AFTER** data_mansione;
aggiunge il campo nato dopo il campo data_mansione
ALTER TABLE voti **CHANGE COLUMN** voto voto float(4,1); aggiorna il tipo del campo voto

7. Cancellazione e modifica di una tabella.

DROP TABLE nometabella eliminazione di una tabella

ALTER TABLE nomevecchiotabella **RENAME** nomenuovotabella rinomina una tabella

ACCESS: Access individua sempre una chiave primaria come un indice, assegnandogli un nome; quindi diversamente da MySQL per l'eliminazione di una chiave primaria o unica si deve far riferimento al nome associato all'indice.

ALTER TABLE <NOMETABELLA> **ADD | DROP**
[COLUMN] tipo NOMECAMPO (lunghezza) **[CONSTRAINT** nome indice**]**

Istruzioni DML di SQL

Sono operazioni di manipolazione dei dati ossia inserimento di nuovi record e l'eliminazione e la modifica di tuple.

1. **Inserimento di nuove righe**

Detta operazione avviene mediante il costrutto INSERT e secondo la seguente sintassi

INSERT INTO <nometabella> (nomi campi) **VALUES** (lista di valori);

MYSQL: e ACCESS:
Ci sono dei casi in cui il comando INSERT viene usato senza tenere conto di tutte le sue istruzioni.
Caso 1: uso completo
INSERT INTO alunni(matricola,nome,cognome,classe) **VALUES** ("1234","Carlo","Rossi","3A");

Caso 2: inserimento del record senza specificare i campi
INSERT INTO alunni **VALUES** ("1234","Carlo","Rossi","3A");

Caso 3: inserimento di più alunni senza specificare i campi
INSERT INTO alunni **VALUES** ("1234","Carlo","Rossi","3A"), ("2678","Ugo","Bianchi","4A");

Caso 4: inserimento di soli tre campi con rispettivi valori
INSERT INTO alunni(matricola,nome,cognome) **VALUES** ("1234","Aldo","Neri");

Per i campi auto-increment non occorre specificare alcun valore basterà sostituirlo con apici vuoti.

MYSQL: solo in MySQL esiste la possibilità di carica tramite comando dei dati da un file esterno di formato diverso a quello del database, unica condizione è che il file deve avere il cosiddetto formato CSV (Comma Separated Values).

LOAD DATA INFILE "nomefile.txt" **[REPLACE | IGNORE]**
INTO TABLE nometabella
FIELDS TERMINATED BY "simbolo"
LINES TERMINATED BY "simbolo";

Dove simbolo può essere per FIELDS (, . spazio) e per LINES (\n)

2. **Sostituzione delle righe**

La sostituzione dei record di una tabella consente di modificare il contenuto di alcuni record

REPLACE
INTO nome tabella (nomi dei campi)
VALUES (elenco valori)
SET nome campo=espressione

È necessario qualora si presenti il caso di sostituire alcuni valori di specificare tutti i campi perché si rischierebbe di azzerare i dati già presenti.

3. Eliminazione delle righe

Con questo sistema si possono cancellare più di un record.

MYSQL:
DELETE FROM nometabella
WHERE condizione
LIMIT massimo_numeri_di_righe_ammesse;
ACCESS:
DELETE FROM nometabella
WHERE condizione;

Il comando si presenta di una semplicità applicativa, ma si trascura il fatto che alcune tabelle possono avere dei vincoli di integrità referenziale e la cancellazione di un record dell'entità più forte potrebbe procurare i seguenti problemi:

a) se nello schema il vincolo è RESTRICT l'eliminazione viene impedita
b) se nello schema il vincolo è SET NULL l'eliminazione viene consentita ed i valori della tabella figlia diventa NULL
c) se nello schema il vincolo è CASCADE l'eliminazione è possibile e nella tabella figlia vengono cancellate tutti i record
d) se nello schema il vincolo è SET DEFAULT l'eliminazione viene consentita ed i valori della tabella figlia diventano di default.

4. Aggiornamento delle righe

Si modifica il contenuto di uno o più record della tabella.

UPDATE nometabella
SET nomecampo= valore o espressione
WHERE condizione;

Istruzioni QL di SQL

L'interrogazione di un database è permessa tramite il costrutto SELECT. Le tre operazioni che tale costrutto esprime sono le seguenti:

- Selezione tramite una condizione WHERE
- Proiezione tramite i campi che si vogliono estrarre
- Congiunzione tramite i costrutti JOIN oppure mediante la clausola WHERE

Il comando SELECT serve per estrarre dati da una tabella, in pratica vengono estratte tutte e solo le righe da una tabella che soddisfano una certa condizione.
Sintassi (la sintassi qui data è ovviamente incompleta e semplificata):

```
SELECT {Campo1, Campo2, ...}
FROM {Tabella 1}
WHERE {Condizione1 AND|OR Condizione2 AND|OR ...}
```

Es: Sia la tabella Anagrafica

Cognome	**Età**
Verdi	15
Rossi	25
Bianchi	85

La query : *SELECT Cognome FROM Anagrafica WHERE Età=25* dà come risultato:

Cognome
Rossi

La sintassi vista finora del comando SELECT permette interrogazioni molto semplici, spesso si cercano dati in maniera molto più complessa, come quando sono presenti relazioni fra tabelle (delle relazioni tra tabelle parleremo più avanti su queste pagine).

Vediamo ora qualche clausola WHERE un po' più complessa:
nella parte che segue il WHERE del comando SELECT può essere inserita una qualsiasi espressione booleana (cioè una qualsiasi espressione che, risolta, dia come risultato uno dei due valori booleani (Vero, Falso)) che sia calcolabile, ovviamente nel risultato della query verranno inserite tutte e sole le righe della tabella che rendono vera l'espressione che segue la clausola WHERE (*Where Condition*).

Es: Sia la tabella Dipendenti:

Cognome	Nome	Età	Data_assunzione
Rossi	Mario	40	10/5/1995
Verdi	Giorgina	20	1/3/1999
Bianchi	Paolo	50	1/6/1975
Gialli	Loredana	35	24/9/1989
Rossi	Giorgio	46	15/7/1979

SELECT * FROM *Dipendenti* ***WHERE*** *True* ritorna tutta la tabella,
mentre la query ***SELECT * FROM*** *Dipendenti* ***WHERE*** *False* non ritorna nulla.

SELECT *Cognome* ***FROM*** *Dipendenti* ***WHERE*** *Età>35* ***AND*** *(Cognome='Rossi'* ***OR*** *Nome Like 'Paolo')* ***AND*** *Età<>46* ritorna solamente le seguenti righe:

Cognome	Nome	Età	Data_assunzione
Rossi	Mario	40	10/5/1995
Bianchi	Paolo	50	1/6/1975

CONDIZIONI dopo la clausola WHERE

= uguale
< minore
> maggiore
<> diverso
<= minore o uguale
>= maggiore o uguale
LIKE contiene
BETWEEN ... AND compreso tra…. e …..

Nella clausola WHERE se il record deve essere uguale a del testo, allora il testo deve essere incluso tra gli apici ', in questo modo:
SELECT nome FROM anagrafici WHERE citta = 'Milano'

mentre se il record deve essere uguale ad un numero, gli apici non devono essere utilizzati.
SELECT nome FROM anagrafici WHERE eta = 17

Like
Like permette di ricercare nei records parole, numeri o parti di essi (iniziali, centrali e finali). La sintassi è leggermente diversa, ma molto semplice:
SELECT nome FROM anagrafici WHERE nome LIKE 'm'

In questo caso vengono dati come risultati tutti i dati, che sono presenti nella colonna nome e che iniziano per M.

Per cercare invece nomi che finiscono per M si utilizza la seguente query:
SELECT nome FROM anagrafici WHERE nome LIKE '%m'

Infine per cercare nomi che finiscono, iniziano o che contengono la M, si utilizza la seguente query:
SELECT nome FROM anagrafici WHERE nome LIKE '%m%'

AND e OR

AND e OR permettono di indicare due o più condizioni di WHERE. L'operatore AND indica che tutte le condizioni devono essere vere, mentre OR indica che ne basta una vera.
Sintassi
SELECT records **FROM** tabella **WHERE** records **CONDIZIONE** valore OPERATORE **WHERE** records CONDIZIONE valore OPERATORE...

La tabella "anagrafici"

Nome	**Cognome**	**Eta**	**Citta**
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

Utilizzo di AND
SELECT * FROM anagrafici WHERE eta = 17 AND citta = '%Oriolo%'
Risultato
Lorenzo Pascucci, 17, Oriolo Romano

Utilizzo di OR
SELECT * FROM anagrafici WHERE eta = 21 OR citta = '%Oriolo%'
Risultato
Lorenzo Pascucci, 17, Oriolo Romano
Marcello Tansini, 21, Milano

Utilizzo combinato di AND e OR
SELECT * FROM anagrafici WHERE (nome = 'Lorenzo' OR nome = 'Marcello') AND eta = 17
Risultato
Lorenzo Pascucci, 17, Oriolo Romano

BETWEEN ... AND
Between ... And serve a prelevare dei dati compresi tra due valori.
Sintassi
SELECT records **FROM** tabella **WHERE** records **BETWEEN** valore1 **AND** valore2
Esempio
La tabella "anagrafici"

Nome	**Cognome**	**Eta**	**Citta**
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

SELECT nome FROM anagrafici WHERE nome BETWEEN Lorenzo AND Michele

Risultato
Marcello

Dopo la ***WHERE*** *condition* possono essere presenti ancora alcuni comandi:

```
GROUP BY
HAVING
ORDER BY {Campo1, Campo2, ...} [ASC | DESC]
```

Il più semplice è ovviamente **ORDER BY**, che serve per ordinare una tabella rispetto ad uno o più campi, le parole chiave opzionali in fondo significano il tipo di ordinamento: **ASC** (o niente) sta per ascendente, **DESC** sta per discendente.
Vediamo subito con un esempio come funziona:

Es: Con la tabella dell'esempio precedente, la query

SELECT * FROM Dipendenti **WHERE** Età<=40 **ORDER BY** Cognome **DESC**

ritorna

Cognome	**Nome**	**Età**	**Data_assunzione**
Verdi	Giorgina	20	1/3/1999
Rossi	Mario	40	10/5/1995
Gialli	Loredana	35	24/9/1989

Se invece non avessimo messo alcunché in fondo o avessimo messo ASC la tabella sarebbe stata ordinata in ordine crescente rispetto al Cognome.

Si può ordinare anche rispetto a più campi, i campi che vengono dopo il primo sono usati per l'ordinamento nel caso in cui i valori del primo sono uguali:

SELECT * FROM Dipendenti **ORDER BY** Cognome, Età

Cognome	Nome	Età	Data_assunzione
Bianchi	Paolo	50	1/6/1975
Gialli	Loredana	35	24/9/1989
Rossi	Mario	40	10/5/1995
Rossi	Giorgio	46	15/7/1979
Verdi	Giorgina	20	1/3/1999

Uno dei più utili comandi all'interno della clausola **WHERE** è sicuramente il **GROUP BY**, che serve per aggregare i dati secondo alcuni criteri; nelle query con questa clausola si possono usare funzioni che permettono di calcolare medie, somme, massimi, minimi etc; per esempio la funzione **COUNT** effettua il conteggio delle righe restituite, mentre la funzione **AVG** effettua la media dei valori dell'intervallo.

Nella parte dopo la parola chiave **SELECT**, che deve contenere i nomi dei campi da estrarre, possono essere messi solo campi contenuti nella clausola **GROUP BY** o funzioni di calcolo di valori su intervalli.

Es: Prendiamo ora invece la tabella Progetti:

Nome_progetto	Ore_uomo	Ambito
fax	400	Comunicazioni
mail	700	Comunicazioni
contabilità	3500	Gestionale
stipendi	7000	Gestionale
SMTP	400	Comunicazioni
POP	7000	Comunicazioni
inventario	10500	Gestionale

La query

SELECT Ambito, Count(*) As TOT, **AVG**(Ore_uomo) AS MEDIA
FROM Progetti **GROUP BY** Ambito

Ritorna

Ambito	TOT	MEDIA
Comunicazioni	4	2125
Gestionale	3	7000

La funzione **AVG** effettua la media sui valori contenuti nel campo Ore_uomo, raggruppati secondo il valore contenuto nel campo Ambito.

La clausola **HAVING** pone delle condizioni sulle clausole di gruppo, per esempio per considerare solo i raggruppamenti che hanno più di un certo numero di elementi:

Es: con la stessa tabella dell'esempio precedente, la query

SELECT Count(*) As NUM, Ore_uomo
FROM Progetti
GROUP BY Ore_uomo
HAVING Count(*)>1

NUM	Ore_uomo
2	400
2	7000

Che si riferiscono ovviamente alla 1°, 4°, 5° e 6° riga della tabella.

La clausola **DISTINCT** serve a non ripetere nei risultati lo stesso valore.

SELECT DISTINCT records **FROM** tabella

Esempio

La tabella "anagrafici"

Nome	Cognome	Eta	Citta
Lorenzo	Pascucci	17	Oriolo Romano
Marcello	Tansini	21	Milano
Michele	Basso	17	Udine

SELECT DISTINCT eta **FROM** anagrafici

Risultato

17
21

L'età di Michele (17 anni come quella di Lorenzo) non viene data come risultato in quanto già esiste un risultato dello stesso valore. La stessa cosa vale per le parole...

è differente dal semplice:

SELECT eta **FROM** anagrafici

in quanto questa istruzione sql avrebbe dato come risultato:

17
21
17