SQL

Structured Query Language

il linguaggio

- SQL è un linguaggio di interrogazione per database progettato per
  - leggere,
  - modificare
  - gestire **dati** memorizzati in un sistema basato sul **modello relazionale**
  - creare e modificare **schemi** di database
  - creare e gestire strumenti di **controllo** ed **accesso** ai dati

evoluzione del linguaggio

A. Ferrari

- origini di SQL in un documento del **1970** realizzato da Edgar Codd, *"A Relational Model of Data of Large Shared Data Banks"*
- prima versione IBM inizio '70
  - SEQUEL
- primo **standard SQL-86** pubblicato da ANSI e ratificato da ISO nel 1987
  - *(ANSI e ISO sono due organismi internazionali che si occupano della standardizzazione delle tecnologie)*
- **SQL-92 (SQL 2)** è lo standard a cui fanno riferimento la maggior parte dei DBMS
- l'evoluzione del linguaggio ha portato a due ulteriori versioni: **SQL:1999** (oggetti) e **SQL:2003** (xml)

i linguaggi "dentro" SQL

A. Ferrari

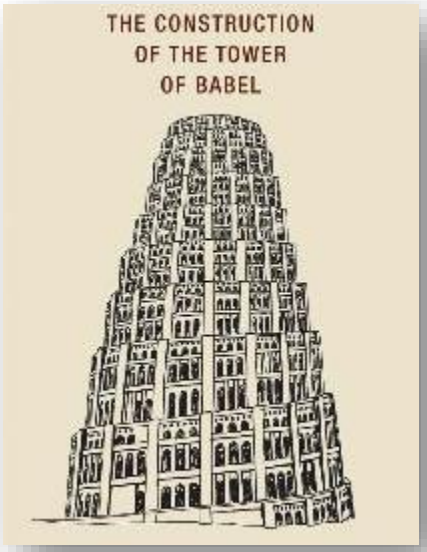
- ***DDL***
  (Data Definition Language, linguaggio di definizione dei dati)
  - descrive struttura tabelle e elementi di supporto
    - (indici, vincoli, trigger, viste ecc.)
  - utilizzato per realizzare lo ***schema logico*** e lo schema ***fisico*** del database
- ***DML***
  (Data Manipulation Language, linguaggio per la manipolazione dei dati)
  - operazioni di inserimento, modifica e cancellazione dei dati
- ***DCL***
  (Data Control Language, linguaggio di controllo dei dati)
  - limiti sui dati (permessi di accesso, vincoli di integrità)
- ***QL***
  (Query Language, linguaggio di interrogazione)
  - interrogare il database per individuare i dati che corrispondono ai parametri di ricerca dell'utente

utilizzo di SQL

- ***interattivo***
  - l'utente utilizza un software, in genere fornito con il DBMS, in cui introdurre comandi SQL che vengono inviati al DBMS
- ***all'interno di applicazioni software***
  - interazione con database scritta in SQL
  - il resto dell'applicazione in un comune linguaggio di programmazione
  - comandi SQL collegati nel programma in due modalità differenti:
    - "**ospitati**" nel codice del software e inviati al DBMS
    - **memorizzati all'interno del DBMS** e richiamati dal programma

dialetti SQL

- alcune delle critiche più frequenti rivolte ad SQL riguardano la ***mancanza di portabilità*** del codice fra implementazioni diverse
- sono spesso differenti alcuni tipi di dato e la sintassi di alcuni operatori particolari (es. LIKE)
- ***case insensitive*** (interpreting upper and lowercase letters as being the same)



mySQL - tipi numerici

A. Ferrari

Tipo	Intervallo/Dimensioni	Memoria
TINYINT	-128 a 127 / 0 a 255	1 byte
SMALLINT	-32.768 A 32.767 / 0 a 65.535	2 byte
MEDIUMINT	-8.388.608 A 8.388.607 / 0 a 16.777.215	3 byte
INT	-2.147.483.648 A 2.147.483.647 / 0 a 4.294.967.295	4 byte
BIGINT	-9.223.372.036.854.775.808 A 9.223.372.036.854.775.807 / 0 a 18.446.744.073.709.550.615	8 byte
FLOAT(M, D)	Variabile Ad esempio FLOAT(5,3) indica a MySQL di salvare 5 cifre totali di cui 3 per la parte decimale. I numeri quindi saranno salvati fino a 99.999. Numeri con valori diversi, verranno arrotondati. Un numero come 45,7869 diventerà 45,787.	4 byte
DOUBLE(M, D)	Variabile DOUBLE e DECIMAL funzionano come FLOAT ma possono contenere valori maggiori (e quindi occupano più spazio).	8 byte
DECIMAL(M, D)	Variabile	M + 2 byte
Modificatori numerici		
AUTO_INCREMENT	Il campo numerico aumenta ogni nuova riga. Se si cancella l'ultimo record di una tabella e se ne aggiunge un altro, il valore del nuovo campo sarebbe comunque incrementato rispetto a quello cancellato. Molto utile per i codici univoci (spesso chiavi primarie) numerici.	
UNSIGNED	Accetta solo valori positivi.	
ZEROFILL	Consente di inserire tanti 0 quanti sono ammessi dal tipo di campo, prima della cifra che realmente viene salvata. Ad esempio, con un campo di tipo INT(5), il valore 78, viene memorizzato come 00078. Può essere utile per salvare dati bancari che prevedono una lunghezza fissa, come ABI, CAB o numero di conto corrente.	

mySQL - stringhe

A. Ferrari

Tipo Stringa/Testo	Intervallo/Dimensioni	
CHAR	255 Ha una lunghezza fissa, se si inserisce ‘ciao’ in un campo CHAR(9) occuperebbe 9 byte. Utile per Codice fiscale	M byte
VARCHAR	255 Scrivendo “ciao” in un campo VARCHAR(9) occuperebbe 5 byte (L+1).	L + 1 byte
TINYTEXT	255	L + 1 byte
TINYBLOB	255	L + 1 byte
TEXT	65.535 Una importante differenza tra TEXT e VARCHAR, a parte la dimensione massima dei dati, è la tecnica memorizzazione dei dati. TEXT (e BLOB) vengono memorizzati fuori dalla tabella, lasciando solo un puntatore alla memoria effettiva. VARCHAR quindi è più veloce e si usa quando la dimensione dei dati è ragionevole.	L + 2 byte
BLOB	65.535 BLOB sta per Binary Large Object e consente il salvataggio di interi file nel formato binario. Utile per nascondere file/documenti importanti direttamente nel database.	L + 2 byte
MEDIUMTEXT	16.777.215	L + 3 byte
MEDIUMBLOB	16.777.215	L + 3 byte
LONGTEXT	4.294.967.295	L + 4 byte
LONGBLOB	4.294.967.295	L + 4 byte
Modificatori Stringhe/Testo		
BINARY	Ammesso dai campi CHAR e VARCHAR: i dati salvati saranno trattati come stringhe binarie.	

mySQL – dateTime

A. Ferrari

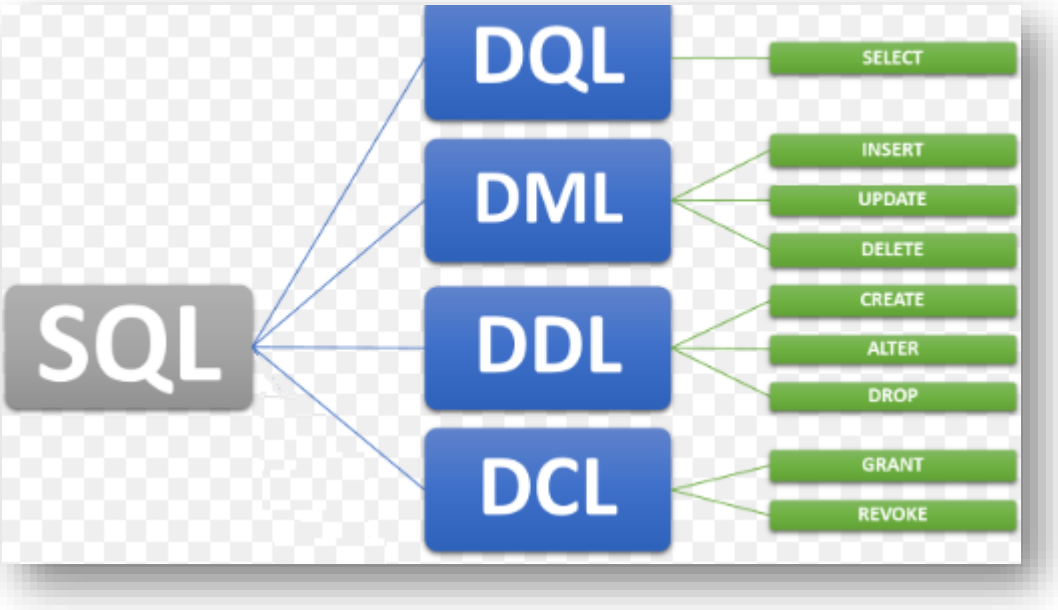
Tipo Data/Ora	
DATETIME	1000-01-01 00:00:00 a 9999-12-31 23:59:59
DATE	1000-01-01 a 9999-12-31
TIME	-838:59:59 a 838:59:59 Salva l'ora. I valori vanno da 00:00:00 a 23:59:59. E' possibile però salvare intervalli di valore tra un evento e un altro e quindi, ammettere ore differenti. In questo caso, i dati vanno da -838:59:59 a 838:59:59. MySQL legge i valori partendo da destra, quindi, salvando un campo con il contenuto 8:32, nel database verrà interpretato come 00:08:32. L'immissione di un valore sbagliato sarà salvato come mezzanotte (00:00:00).
YEAR	1901 a 2155 Salva l'anno ed è il campo più leggero: 1 solo byte. Ammette valori dal 1901 al 2155. Gli anni possono essere salvati a due o a quattro cifre. MySQL, in caso di anni a due cifre, interpreterà i valori da 70 a 99 come dal 1970 al 1999. Quelli dall'1 al 69, come dal 2001 al 2069.
TIMESTAMP	Variabile Salva (nel formato scelto dal numero tra le parentesi) il momento esatto in cui la tabella viene modificata. Quindi può essere utile per visualizzare il momento dell'ultima modifica del record a cui il campo appartiene. Ammette anni compresi tra il 1970 e il 2037. Tutti i tipi di TIMESTAMP occupano lo stesso spazio: 4 byte perché comunque salva tutti i dati e poi ne visualizza solo quelli richiesti. Ad esempio, con TIMESTAMP(2) il database visualizza solo due cifre dell'anno, ma in memoria ha tutti gli altri dati (anno a 4 cifre, mese, giorno, ora, minuti e secondi). Quando infatti modifichiamo il tipo di TIMESTAMP, ad esempio con TIMESTAMP(8) ci sono tutti i dati.

modificatori

- ***UNIQUE***
  - i valori devono essere diversi uno dall'altro
  - se si tenta di aggiungere un valore duplicato MySQL genera un errore (1062 – Duplicate entry 'N' for key N)
- ***DEFAULT***
  - imposta valore predefinito nel caso il campo sia lasciato vuoto
- ***NOT NULL***
  - impone che il campo non sia lasciato vuoto
- ***NULL***
  - se il campo non contiene un valore, sarà vuoto
- ***PRIMARY KEY***
  - imposta un indice, i dati non devono essere vuoti

operatori

+	Addizione
-	Sottrazione
*	Prodotto
/	Divisione
%	Modulo
<	Minore
>	Maggiore
<=	Minore o Uguale
>=	Maggiore o Uguale
=	Uguaglianza
<>	Disuguaglianza
AND	E logico
OR	O logico
NOT	Negazione



DDL

Data Definition Language

creazione database

A. Ferrari

- **CREATE DATABASE <NomeDB>**
- esempio
  - **CREATE DATABASE Cinema**

creazione tabella

A. Ferrari

```
CREATE TABLE <NomeTabella> (
      <NomeCampo1> <Tipo1> [NOT NULL],
      <NomeCampo2> <Tipo2> [NOT NULL],
      …
      <NomeCampoN> <TipoN> [NOT NULL],
);
```

modifica tabella

- ***aggiungere*** un nuovo ***campo*** ad una tabella:

```
ALTER TABLE <NomeTabella>
    ADD <NomeCampo1> <Tipo1> [NOT NULL];
```

- ***modificare*** il tipo di un ***campo***:

```
ALTER TABLE <NomeTabella>
    ALTER COLUMN <NomeCampo> <NuovoTipo>;
```

- ***eliminare*** un ***campo***:

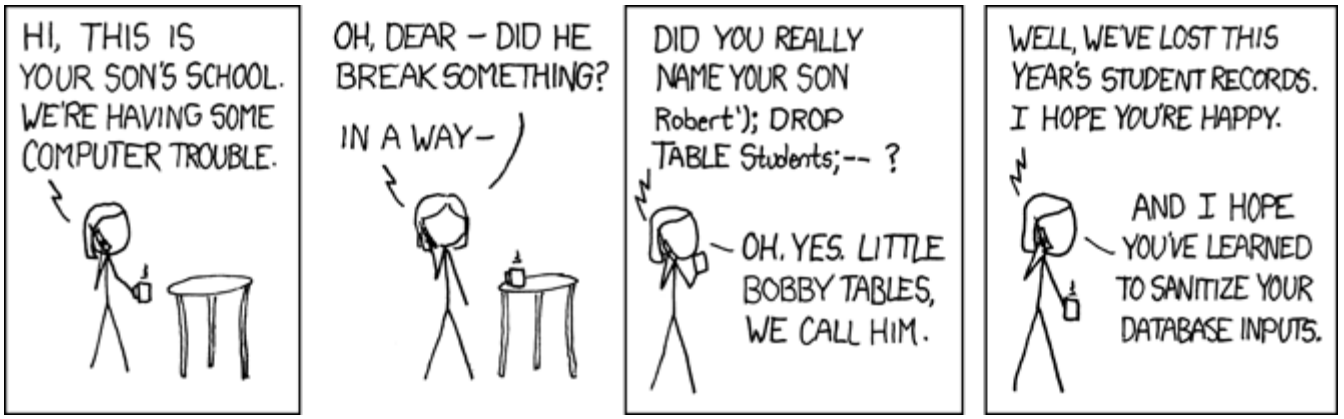
```
ALTER TABLE <NomeTabella>
    DROP COLUMN <NomeCampo1>;
```

eliminazione tabella

A. Ferrari

```
DROP TABLE <NomeTabella>;
```

- ***attenzione***: non è possibile eliminare una tabella a cui fa riferimento un vincolo FOREIGN KEY
  - è prima necessario eliminare il vincolo FOREIGN KEY o la tabella di riferimento



- i vincoli (***constraints***) consentono di specificare **controlli** sui dati, al fine di assicurare la **correttezza** e **consistenza** dell'informazione
- i vincoli possono essere:
  - **interni** (o intrarelazionali) specificano controlli sulla singola tabella intesa come entità a se stante
  - di **integrità referenziale** riguardano i rapporti tra una tabella e l'altra

vincoli interni

- **NOT NULL**
  - ***impedisce*** di inserire un dato nullo nel campo in cui viene specificato
  - **<NomeCampo> <Tipo> NOT NULL;**
- **PRIMARY KEY**
  - imposta un campo (o più campi) come ***chiave primaria*** della tabella
  - **PRIMARY KEY (<NomeCampo>);**
- **CHECK**
  - indica un controllo su un'espressione tra i campi della tabella
  - **CHECK (<NomeCampo> VALUE IN (<valori>));**
  - **CHECK (<NomeCampo> VALUE BETWEEN (<valore1> AND <valore2>));**

vincoli di integrità referenziale

A. Ferrari

- **FOREIGN KEY**
  - imposta una chiave esterna in una tabella, con campi che fanno riferimento ad un'altra tabella del DataBase
- **FOREIGN KEY (<ElencoCampi>) REFERENCES <NomeTabella>(<ElencoCampiTabella>);**
  - **<ElencoCampi>**
    - elenco dei campi della tabella corrente
  - **<NomeTabella>**
    - tabella in cui sono presenti i campi esterni
  - **<ElencoCampiEsterni>**
    - elenco dei campi della tabella di riferimento

integrità referenziale

- l'integrità referenziale viene controllata anche dalle parole chiave **`RESTRICT`**, **`CASCADE`** e **`SET NULL`**, che consentono di controllare la risposta del database a un vincolo
- **`RESTRICT`**
  - il database ***rifiuta*** le modifiche violano un vincolo
- **`CASCADE`**
  - il database ***propaga*** a cascata le modifiche
- **`SET NULL`**
  - è consentita la modifica alla tabella principale, eventuali riferimenti in altre tabelle non più validi vengono posti a **`NULL`**

QL

Query Language

SELECT

- per estrarre informazioni dalla base di dati si utilizza l'istruzione **`SELECT`**
- la sintassi completa dell'istruzione **`SELECT`** è complessa perché l'istruzione implementa varie funzionalità

SELECT (proiezione)

A. Ferrari

```
SELECT [DISTINCT]
<Campo1> [AS "Alias1"],
<Campo2> [AS "Alias2"],
 …
<CampoN> [AS "AliasN"]
FROM <Tabella1>, <Tabella2>, … <TabellaN>
```

- **DISTINCT** - permette di ottenere solo tuple differenti tra loro
- **<Campo>** - elenco dei campi da estrarre
- **<Tabella>** - tabella in cui sono contenuti i campi da estrarre
- **"Alias"** - etichetta da assegnare al campo nella selezione (*facoltativa*)
- *****
  - sostituendolo ai nomi dei campi implica la selezione di tutti i campi della tabella specificata

esempi

- selezione di un'***intera*** tabella

```
SELECT *
FROM Genere
```

- selezione di alcuni campi di una tabella (***proiezione***)

```
SELECT titolo, durata
FROM Film
```

- selezione (***senza duplicazione***)

```
SELECT DISTINCT titolo
FROM Film
```

SELECT (restrizione)

A. Ferrari

- per estrarre informazioni dal DB, ***limitate da una condizione***:

```
SELECT [DISTINCT]
<Campo1>, <Campo2>, … <CampoN>
FROM <Tabella>
[WHERE <Condizione>]
```

- **<Condizione>**
  - indica la condizione che devono soddisfare le tuple estratte
  - all'interno di questa espressione è possibile specificare:
    - nomi dei campi della tabella
    - operatori di confronto, come =, <>, >, >=, <=, <
    - operatori logici come **NOT, AND, OR**
    - l'operatore **LIKE**
    - la parola chiave **IS NULL** o **IS NOT NULL**

esempi

- selezione delle righe che soddisfano una condizione (***restrizione***)

```
SELECT *
FROM Film
WHERE durata>100
```

- selezione con condizione composta

```
SELECT *
FROM Film
WHERE durata>100 AND titolo LIKE 'M%'
```

esempi

- selezione di alcuni campi delle righe che soddisfano una condizione (***restrizione*** e ***proiezione***)

```
SELECT titolo, durata
FROM Film
WHERE titolo LIKE '%K‘
```

- ***alias*** per le colonne

```
SELECT titolo, regia AS Regista
FROM Film
WHERE titolo LIKE '_L%'
```

esempi

- selezione di valori NULL

```
SELECT  *
FROM Film
WHERE titoloOriginale IS NULL
```

- selezione di valori NOT NULL

```
SELECT *
FROM Film
WHERE titoloOriginale IS NOT NULL
```

SELECT (join)

A. Ferrari

- per ***concatenare*** due tabelle in base ad un campo comune (JOIN) può essere utilizzata l'istruzione SELECT-WHERE, con una particolare condizione:

```
SELECT [DISTINCT]
<Campo1>, <Campo2>, … <CampoN>
FROM <Tabella1>, <Tabella2>, … <TabellaN>
WHERE <Tabella1>.<Campo1> = <Tabella2>.<Campo2> …
```

- primo formato

```
SELECT *
FROM Film, Genere
WHERE Film.genere = Genere.codice
```

- formato esplicito

```
SELECT *
FROM Film INNER JOIN Genere
    ON Film.genere = Genere.codice
```

Left Outer Join

A. Ferrari

- oltre alle righe che soddisfano la condizione vengono anche incluse ***tutte*** le ***righe*** della ***prima tabella***

```
SELECT *
FROM Film LEFT OUTER JOIN Premio
     ON Premio.film = Film.codice
```

- in questo caso anche i film che non hanno vinto premi
- esistono anche
  - **RIGHT OUTER JOIN ...**
  - **FULL OUTER JOIN ...**

unione di due tabelle

- per ***accodare*** le righe di due tabelle compatibili (con campi omogenei):

```
SELECT
  <Campo1>
    FROM <Tabella1>
UNION
  SELECT
  <Campo2>
  FROM <Tabella2>;
```

esempio


```
SELECT titolo, durata
FROM   Film
WHERE Film.durata>300

UNION

SELECT titolo, durata
FROM   Film INNER JOIN Premio
          ON Premio.film = Film.codice
WHERE  Premio.anno='1975';
```

differenza

- per estrarre da due tabelle compatibili (con campi omogenei) solo i record ***presenti*** nella ***prima*** ma ***non*** nella ***seconda***:

```
SELECT
    <Campo1>
  FROM <Tabella1>
EXCEPT
 SELECT
    <Campo2>
  FROM <Tabella2>;
```

esempio


```
SELECT titolo, durata
FROM   Film
WHERE Film.durata>300

EXCEPT

SELECT titolo, durata
FROM   Film INNER JOIN Premio
          ON Premio.film = Film.codice
WHERE  Premio.anno='1975';
```

intersezione

- per estrarre da due tabelle compatibili (con campi omogenei) i record che ***entrambe*** le tabelle hanno in comune:

```
SELECT
    <Campo1>
  FROM <Tabella1>
INTERSECT
 SELECT
    <Campo2>
  FROM <Tabella2>;
```

esempio


```
SELECT titolo, durata
FROM   Film
WHERE Film.durata>300

INTECEPT

SELECT titolo, durata
FROM   Film INNER JOIN Premio
          ON Premio.film = Film.codice
WHERE  Premio.anno='1975';
```

funzioni di aggregazione

- SQL dispone di alcune modalità per effettuare calcoli sui dati, ***senza modificare*** i dati in tabella: il calcolo di espressioni e l'utilizzo di funzioni predefinite

funzioni per i calcoli sui dati

A. Ferrari

- **`COUNT([DISTINCT] <Campo>)`**
  - conta il ***numero di elementi*** del campo indicato
- **`MIN(<Campo>)`**
  - restituisce il valore ***minimo*** del campo indicato
- **`MAX(<Campo>)`**
  - restituisce il valore ***massimo*** del campo indicato
- **`SUM([DISTINCT] <Campo>)`**
  - calcola e restituisce la ***somma*** dei valori presenti nel campo indicato
- **`AVG([DISTINCT] <Campo>)`**
  - calcola e restituisce la ***media*** aritmetica dei valori presenti nel campo indicato

ordinamento

- per ordinare i risultati ottenuti in base al valore di uno o più campi:

```
SELECT [DISTINCT]
    <Campo1>, <Campo2>, … <CampoN>
FROM <Tabella1>, <Tabella2>, … <TabellaN>
[WHERE <Condizione>]
[ORDER BY <CampoOrdine1> [ASC|DESC], <CampoOrdine2>
[ASC|DESC], … <CampoOrdineN> [ASC|DESC]];
```

- **<CampoOrdine>** - campo(i) in base al(ai) quale(i) ordinare il risultato ottenuto dalla SELECT
- **ASC|DESC** - indicano l'ordinamento crescente **[ASC]** o decrescente **[DESC]** dei campi
  - default **ASC**

- **`GROUP BY`** raggruppa le righe sulla base del valore di uno o più attributi, in genere per effettuare calcoli aggregati su dati omogenei

raggruppamento (esempio)

A. Ferrari

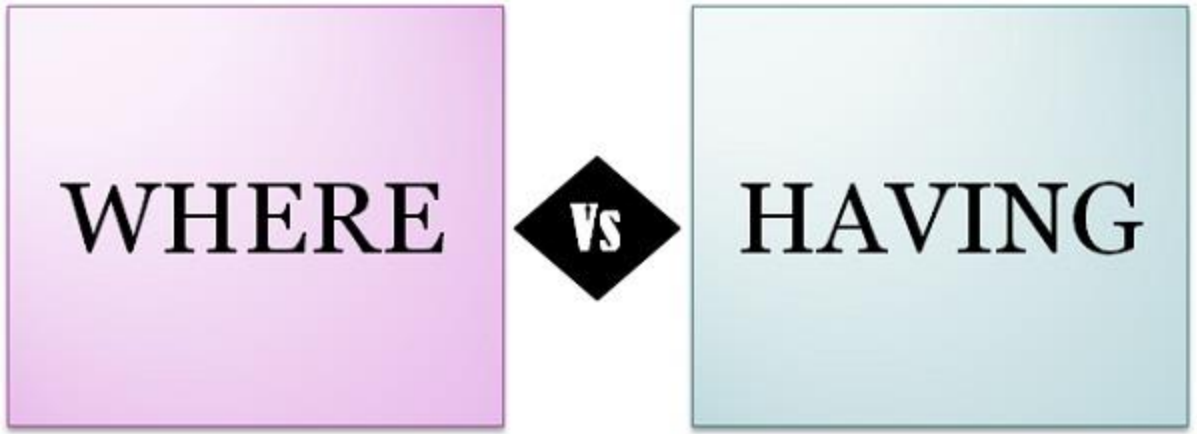
- per ***raggruppare*** in base al valore dei campi selezionati:

```
SELECT [DISTINCT]
    <Campo1>, <Campo2>, … <CampoN>
FROM <Tabella1>, <Tabella2>, … <TabellaN>
[WHERE <Condizione>]
[GROUP BY <CampoGruppo1>, < CampoGruppo2>, …
<CampoGruppoN>
    [HAVING <CondizioneGruppo>]];
```

- **<CampoGruppo>** - campo(i) in base al(ai) quale(i) raggruppare tutti i record ottenuti dalla SELECT
- **<CondizioneGruppo>** - specifica la condizione secondo la quale verranno raggruppati i record

HAVING

- è anche possibile restringere il risultato specificando una condizione che può considerare sia i campi sia il valore di funzioni di aggregazione



interrogazioni nidificate

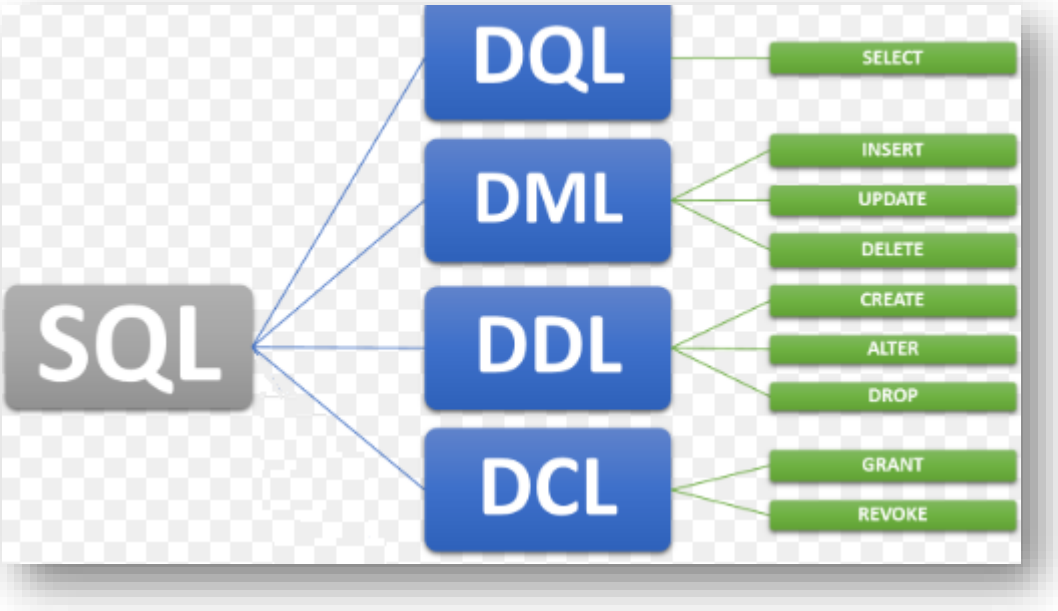
A. Ferrari

- talvolta le operazioni di interrogazione si rivelano particolarmente complesse
  - in questo caso, è necessario utilizzare più istruzioni SELECT al fine di ottenere tutti i dati voluti

```
SELECT
<Campo1>
FROM <Tabella1>
WHERE <Campo1> = (
            SELECT
            <Campo2>
            FROM <Tabella2>
            WHERE <Condizione2>);
```

ANY - ALL

- **`ANY`** ritorna vero se il confronto indicato è ***vero per almeno uno*** degli elementi identificati dalla query nidificata
- **`ALL`** ritorna vero se il confronto indicato è ***vero per tutti*** gli elementi individuati dalla query nidificata
- **`ANY`** e **`ALL`** sono più potenti di **`IN`**, in quanto consentono di utilizzare operatori di confronto >, >=, <= e <



DML

Data Manipulation Language

inserimento dati

A. Ferrari

```
INSERT INTO <NomeTabella>
    [(<Campo1>, <Campo2>, … <CampoN>)]
VALUES
    (<Valore1>, <Valore2>, … <ValoreN>);
```

- **<NomeTabella>** - nome della tabella in cui inserire i dati
- **<Campo>** - lista dei campi della tabella in cui inserire i valori specificati di seguito
- **<Valore>** - lista dei valori da inserire nei rispettivi campi
- *l'elenco dei campi è opzionale; se non viene specificato è necessario inserire un valore per tutti i campi della tabella*

modifica dati


```
UPDATE
     <NomeTabella>
SET
     <Campo1> = <Valore1>,
     <Campo2> = <Valore2>,
     …
     <CampoN> = <ValoreN>
[WHERE <Condizione>];
```

- **<NomeTabella>** - nome della tabella in cui modificare i dati
- **<Campo>** - lista dei campi della tabella in cui modificare i dati esistenti con i valori seguenti
- **<Valore>** - lista dei valori da sostituire a quelli dei rispettivi campi
- *se non viene specificata alcuna condizione WHERE, il valore inserito viene sostituito ai valori di ogni campo*

eliminazione dati


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

- **<NomeTabella>** - nome della tabella dalla quale verranno eliminati i dati
- **<Condizione>** - condizione che deve essere soddisfatta dai campi che verranno eliminati
- *se non viene specificata alcuna condizione WHERE, viene eliminato il valore di ogni campo*

SQL come linguaggio ospitato

A. Ferrari

- per eseguire comandi SQL da un programma scritto in un linguaggio differente è necessario effettuare alcune operazioni aggiuntive:
  - ***connessione***: per ottenere un oggetto che consentirà di eseguire uno o più comandi SQL
    - la connessione è necessaria per stabilire con quale database si vuole operare e per fornire dati di autenticazione (in genere utente/password)
  - ***creazione*** di un ***comando*** SQL: viene creato un oggetto che rappresenta un'istruzione SQL e che viene impostato con uno specifico comando
  - ***esecuzione*** del ***comando***: comporta il passaggio dell'oggetto che rappresenta il comando a quello che rappresenta la connessione, in modo che il comando venga eseguito

SQL come linguaggio ospitato

A. Ferrari

- ***iterazione*** sulla ***risposta***: il risultato di un'istruzione SELECT è una tabella e in genere un programma deve scorrere le righe del risultato per elaborarle
- ***chiusura*** della ***risposta***: l'oggetto che rappresenta la risposta, una volta utilizzato, deve essere chiuso e rilasciato dalla memoria
- ***chiusura*** del ***comando***: l'oggetto che rappresenta il comando SQL, una volta utilizzato, deve essere chiuso e rilasciato dalla memoria
- ***chiusura*** della ***connessione***: l'oggetto che rappresenta la connessione SQL, prima della conclusione del programma, deve essere chiuso e rilasciato dalla memoria

SQL ospitato