

SQL

Lo *Structured Query Language* - di **SQL** è l'acronimo - è un linguaggio utilizzato per l'interrogazione e la gestione di basi di dati. I comandi SQL si distinguono principalmente in tre categorie:

- DDL - Data Definition Language
- DML - Data Manipulation Language
- DCL - Data Control Language

Oggetto di questo capitolo sono i **comandi DDL**, cioè i comandi del tipo:

- Create
- Alter
- Drop

ovvero i comandi che gestiscono la creazione, l'a modifica e la cancellazione delle strutture.

CREATE

Supponiamo di voler costruire una tabella di anagrafica degli impiegati di un'azienda. Volendo farlo senza ricorrere alla funzionalità automatica vista in precedenza, dovremo utilizzare una query del genere:

CREATE

Supponiamo di voler costruire una tabella di anagrafica degli impiegati di un'azienda. Volendo farlo senza ricorrere alla funzionalità automatica vista in precedenza, dovremo utilizzare una query del genere:

```
CREATE TABLE Anagrafica
(
  Nome VARCHAR2(80 BYTE),
  Cognome VARCHAR2(80 BYTE),
  Stipendio Number
);
```

Attraverso questa istruzione otterremo che la nostra tabella sarà costituita dalle 3 colonne (Nome, Cognome e Stipendio) dove le prime due saranno di tipo alfanumerico con lunghezza fino a 80 byte mentre l'ultima colonna sarà di tipo numerico.

ALTER

Supponiamo ora di voler modificare la nostra tabella inserendo la colonna Ruolo. Per farlo scriveremo:

```
ALTER TABLE Anagrafica
ADD COLUMN Ruolo VARCHAR2(80 BYTE);
```

Otterremo così che la nostra tabella sarà costituita da quattro colonne (Nome, Cognome, Stipendio e Ruolo) dove le prime due e la quarta colonna saranno di tipo alfanumerico con lunghezza fino a 80 byte mentre la terza colonna sarà di tipo numerico.

DROP

Per finire vediamo come procedere alla cancellazione della nostra tabella di prova:

```
DROP TABLE Anagrafica;
```

Da notare che Oracle committa implicitamente la transazione corrente prima e dopo ogni comando DDL. Questo comporta che non è possibile effettuare il Rollback ma bisogna effettuare le operazioni "inverse". Questi concetti sono solo anticipati, verranno trattati approfonditamente in un successivo capitolo.

Oggetto di questa lezione sono, invece, i **comandi DML**; essi sono del tipo:

- INSERT
- SELECT
- DELETE
- UPDATE

Si tratta, cioè, dei comandi per la manipolazione dei dati.

INSERT

Il comando di **INSERT** serve per inserire nuovi record (righe di dati) in tabella.

Supponiamo di voler inserire nella nostra tabella di anagrafica due record; per farlo scriveremo:

```
INSERT
INTO Anagrafica
VALUES('Mario', 'Rossi', 20000, 'Impiegato');

INSERT
INTO Anagrafica
VALUES('Carlo', 'Verdi', 35000, 'Quadro');
```

Il comando INSERT, così come scritto sopra, può essere utilizzato solo quando nella parte VALUES vengono forniti valori per tutte le colonne della tabella.

Nel caso in cui, invece, si vogliono inserire solo i dati di alcune colonne (rispettando gli eventuali vincoli di obbligatorietà della tabella) bisognerà utilizzare la sintassi che possiamo definire completa. Essa, oltre alla parte vista precedentemente, prevede anche l'elenco dei campi della tabella per cui saranno forniti i valori nella rispettiva parte VALUES:

SELECT

L'istruzione di SELECT viene, invece, utilizzata per interrogare le tabelle:

```
SELECT *  
FROM Anagrafica;
```

Con la sintassi appena vista otterremo in risultato tutte le colonne per ogni record appartenente alla nostra tabella.

Per ottenere solo specifiche colonne, invece, scriveremo:

```
SELECT Nome, Cognome  
FROM Anagrafica;
```

Nello specifico abbiamo chiesto di ottenere in risposta solo le colonne "Nome" e "Cognome" per ogni record presente in tabella.

Qualora, invece, si volesse filtrare il risultato limitandolo a specifici record, dovremo far ricorso alla clausola WHERE, in questo modo:

```
SELECT Nome, Cognome  
FROM Anagrafica  
WHERE Cognome = 'Rossi';
```

DELETE e UPDATE

L'istruzione DELETE viene utilizzata per eliminare uno o più record dalla tabella. Vediamo un esempio:

```
DELETE  
FROM Anagrafica  
WHERE Nome = 'Carlo';
```

Mentre l'istruzione UPDATE viene utilizzata per aggiornare uno o più record presente in tabella:

```
UPDATE Anagrafica  
SET Ruolo = 'Impiegato'  
WHERE Cognome = 'Bianchi'  
AND Nome = 'Lucia';
```

Eccezion fatta per la SELECT, che non effettua alcuna modifica in tabella, dopo ognuna delle altre istruzioni elencate in questo capitolo, per far sì che le modifiche vengano effettivamente salvate sulla tabella, bisogna effettuare la COMMIT.

Tale istruzione, ed altre di questo tipo, saranno trattate in maniera approfondita nel capitolo seguente.

Il comando **INSERT**

L'inserimento successivo di dati può avvenire con un'apposita istruzione che opera direttamente prendendo i dati immessi:

INSERT INTO tabella
VALUES (dato1, dato2,)

Per aggiungere una nuova riga alla tabella docenti si può scrivere da esempio:

INSERT INTO Docenti
VALUES ("Mario", "Rossi", "Italiano")

Nome	Cognome	Materia
Lara	Bianco	Italiano
Lara	Bianco	Storia
Mario	Guidi	Diritto
Mario	Guidi	Economia
Lucia	Zucconi	Storia
Alfio	Righetti	Storia
Anna	Rossignuolo	Diritto
Giuseppina	Merlino	Italiano
Mario	Rossi	Italiano

Il comando **DELETE**

Si possono cancellare definitivamente intere righe di una tabella, con l'istruzione

DELETE
FROM tabella
WHERE condizione.

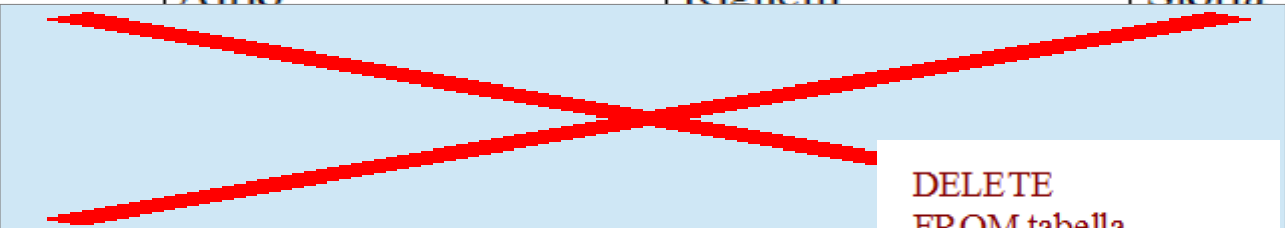
Con la seguente istruzione

DELETE
FROM DOCENTI
WHERE Materia="Italiano"

Si cancellano dalla tabella DOCENTI gli insegnanti d'Italiano

Tabella derivata

Nome	Cognome	Materia
Lara	Bianco	Storia
Mario	Guidi	Diritto
Mario	Guidi	Economia
Lucia	Zucconi	Storia
Alfio	Righetti	Storia



DELETE
FROM tabella

Il comando UPDATE

E' possibile operare sui dati sia in modo complessivo sia in modo selettivo. In ogni caso gli operatori di modifica, come tutti gli altri, operano sempre su gruppi di dati. La sintassi generale è:

UPDATE tabella
SET colonna=nuovo_valore
[WHERE condizione]

La clausola WHERE consente di selezionare le righe da trattare.

Supponiamo che cambi la descrizione di una materia, ad esempio da STORIA si passi a STORIA E EDUCAZIONE CIVICA. A partire dalla tabella:

Tabella DOCENTI

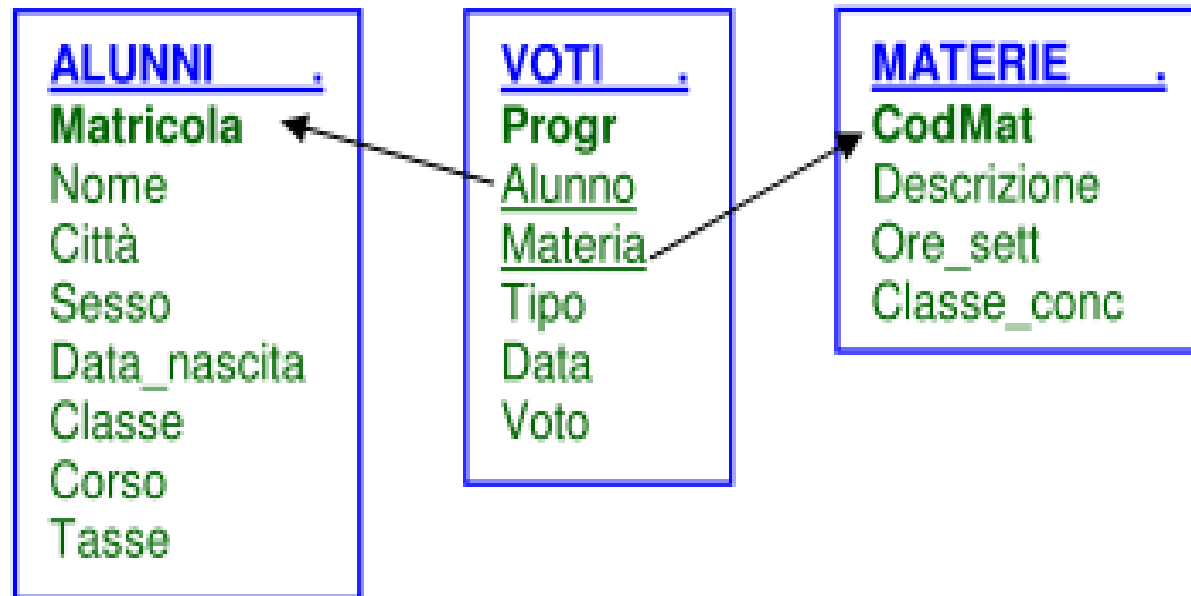
Nome	Cognome	Materia
Lara	Bianco	Italiano
Lara	Bianco	Storia
Mario	Guidi	Diritto
Mario	Guidi	Economia
Lucia	Zucconi	Storia
Alfio	Righetti	Storia
Anna	Rossignuolo	Diritto
Giuseppina	Merlino	Italiano

si può procedere con:

```
UPDATE DOCENTI  
SET MATERIA=" STORIA E EDUCAZIONE CIVICA "  
WHERE MATERIA="STORIA"
```

Supponiamo di disporre di una tabella LISTINO con i prezzi dei testi in adozione e supponiamo che ci sia stato, per tutti, un aumento del 2%. Si può operare come segue:

```
UPDATE LISTINO  
SET PREZZO=PREZZO * 1.02
```



Istruzioni di tipo DDL (*Data Definition Language*)

Creazione tabella:

CREATE TABLE <i>tabella</i> (<i>attributo</i> ₁ <i>tipo</i> ₁ [NOT NUL], <i>attributo</i> _n <i>tipo</i> _n [NOT NUL])	CREATE TABLE Voti (Progr Integre, Alunno Integer, Materia Text(3), Tipo Text(1), Data Date)
---	--

crea la tabella Voti con i campi Progr, Alunno, Materia, Tipo e data

Creazione indice:

CREATE [UNIQUE] INDEX <i>indice</i> ON <i>tabella</i> (<i>attributo</i> _{j1} , <i>attributo</i> _{j2} , ..., <i>attributo</i> _{jk})	CREATE UNIQUE INDEX Chiave ON Voti (Progr)
---	---

Crea l'indice Chiave della tabella Voti sul campo Progr

Rimozione tabella e/o indice:

DROP <i>tabella</i>	DROP Voti	Cancella la tabella Voti
DROP <i>indice</i> [DB2 Oracle]	DROP <i>indice</i> ON <i>tabella</i>	[Access]
DROP <i>tabella.indice</i> [SQL Server]	ALTER TABLE <i>tabella</i> DROP INDEX <i>indice</i> [My SQL]	

Modifica tabella (aggiunge, toglie campi):

ALTER TABLE <i>tabella</i> ADD <i>attributo</i> <i>tipo</i> ALTER TABLE <i>tabella</i> DROP <i>attributo</i>	ALTER TABLE Voti ADD Voto Single
---	--

Aggiunge il campo Voto alla tabella Voti

Creazione TABELLA, con INDICI ed ASSOCIAZIONI:

CREATE TABLE <i>tabella</i> (<i>attributo</i> ₁ <i>tipo</i> ₁ [NOT NUL], <i>attributo</i> _n <i>tipo</i> _n [NOT NUL], PRIMARY KEY (<i>chiave_primaria</i>) FOREIGN KEY (<i>chiave_esterna</i>) REFERENCES <i>tabella</i> (<i>chiave</i>) ON DELETE <i>set null/no action</i> ON UPDATE <i>cascade/no action</i>)	CREATE TABLE Voti (Progr Integre, Alunno Integer, Materia Text(3), Tipo Text(1), Data Date, PRIMARY KEY Progr, FOREIGN KEY (Alunno) REFERENCES Alunni(Matricola) ON DELETE set null ON UPDATE cascade)
---	---

crea la tabella Voti, la sua chiave primaria e la chiave esterna "Alunno" (la definizione di "Materia" è analoga)

Istruzioni di tipo DML (*Data Manipulation Language*)

Inserimento dati:

INSERT INTO <i>tabella</i> [(<i>attr</i> ₁ , <i>attr</i> ₂ , ..., <i>attr</i> _n)] VALUES (<i>valore</i> ₁ , <i>valore</i> ₂ , ..., <i>valore</i> _n)	INSERT INTO <i>Voti</i> VALUES (55, 358, 'INF', 'S', #01/03/05#, 7)
--	--

Registra il 7 di informatica scritta all'alunno 358

Modifica dati:

UPDATE <i>tabella</i> SET <i>attr</i> ₁ = <i>exp</i> ₁, <i>attr</i> _n = <i>exp</i> _n [WHERE <i>condizione</i>]	UPDATE <i>Voti</i> SET <i>Voto</i> = <i>Voto</i> - 1 WHERE <i>Alunno</i> = 358
---	---

Diminuisce di 1 tutti i voti dell'alunno 358

Cancellazione dati:

DELETE FROM <i>tabella</i> [WHERE <i>condizione</i>]	DELETE <i>Voti</i> WHERE <i>Alunno</i> = 358 And <i>Tipo</i> = 'O'
--	--

Cancella tutti i voti orali dell'alunno 358

Istruzioni di tipo QL (*Query Language*)

Per il reperimento dati SQL prevede un unico potentissimo comando, quello che ne ha decretato il successo e che, nella sua forma più semplice si presenta così:

```
SELECT    lista attributi (risultato)  
FROM      lista tabelle  
WHERE     condizione
```

che restituisce la relazione formata da "lista attributi" del prodotto delle relazioni "lista relazioni" ristretto alla "condizione" impostata. Si possono avere SELECT "nidificate" specificando nella clausola WHERE un'altra SELECT il cui risultato sarà utilizzato per impostare le condizioni della SELECT principale.

Si può creare una query a "campi incrociati" (che calcola una somma, una media, un conteggio o altri tipi di totale sui record e quindi raggruppa i risultati in base a due tipi di informazioni: uno in basso a sinistra della griglia - intestazioni di righe - e l'altro nella parte superiore - intestazioni di colonne):

Istruzione SELECT

L'istruzione che ha decretato il successo di SQL è senz'altro la SELECT. Con tale istruzione si possono fare in modo relativamente semplice interrogazioni anche molto complesse.

SELECT [DISTINCT]	<i>elenco campi / espressioni (risultato)</i> [INTO nuova_tabella [IN database_esterno]]
FROM	<i>elenco tabelle</i>
[WHERE	<i>condizione</i>]
[GROUP BY	<i>espressione gruppo</i>]
[HAVING	<i>condizione per il gruppo/risultato</i>]
[ORDERED BY	<i>campi di ordinamento</i> [DESC]]

Se i nomi dei campi contengono degli spazi debbono essere delimitati da parentesi quadre []. Con la clausola **AS** (nell'elenco campi e/o nell'elenco tabelle) si possono rinominare le colonne del risultato e/o le tabelle. Nell'elenco campi si possono inserire espressioni il cui nome viene dichiarato da AS.

```
SELECT Matricola, Nome AS Alunno, (Year(DATE())-Year(Data_nascita)+1) AS Età
```

rinomina il campo Nome come Alunno e calcola l'Età in base al Data_nascita e alla data di sistema

L'asterisco (*) indica "tutti i campi". Se ci sono più tabelle, i campi si indicano con *tabella.campo*.

```
SELECT Alunni.Nome,Voti.* FROM Alunni, Voti
```

seleziona il Nome della tabella Alunni e tutti i campi della tabella Voti

Con la clausola **DISTINCT** le righe uguali del risultato vengono ridotte ad una (cancella i duplicati).

```
SELECT DISTINCT Città FROM Alunni
```

elenca, senza duplicazioni, le diverse città dalle quali provengono gli alunni

Nelle condizioni – WHERE e HAVING - si usano gli operatori di relazione (=, >, >=, <, <=, <>), logici (AND, OR, NOT) e le parentesi tonde. Si può testare se un campo del risultato è nullo con la clausola **IS NULL** (e, analogamente, se non è nullo con NOT IS NULL).

```
SELECT * FROM Alunni WHERE Città = 'Perugia' AND Data_nascita NOT IS NULL
```

elenca tutti gli alunni di Perugia con data di nascita impostata

Nelle condizioni si possono utilizzare gli operatori: **BETWEEN** per impostare un intervallo; **IN** per impostare un insieme di ricerca; **LIKE** per ricercare sottostringhe con i caratteri jolly **%** (tutto) e **_** (singolo carattere).

```
SELECT * FROM Alunni WHERE (Tasse BETWEEN 10 AND 30) AND Classe IN ('3Atp', '4Atp') AND Nome LIKE 'M%'
```

elenca tutti gli alunni della 3Atp o 4Atp il cui nome inizia per 'M' che hanno tasse da 10 a 30 euro

Il risultato può essere ordinato su uno o più campi in ordine crescente – per default - o in ordine decrescente, specificando la clausola **DESC**.

```
SELECT * FROM Alunni WHERE Città = 'Perugia' ORDER BY Classe DESC, Nome
```

elenca tutti gli alunni di Perugia in ordine decrescente di Classe e, per ogni Classe, in ordine alfabetico sul Nome

Con la clausola **INTO** si può creare una nuova tabella nella quale memorizzare permanentemente il risultato dell'interrogazione e tale tabella può anche essere creata in un database esterno specificato dalla clausola **ON**.

```
SELECT Matricola, Nome FROM Alunni INTO AluPg WHERE Città = 'Perugia'
```

crea la nuova tabella AluPg con tutti gli alunni di Perugia (e con i soli campi Matricola e Nome)

Con l'istruzione SELECT si possono usare diverse **funzioni di aggregazione**:

- **COUNT(*)** numero totale di record; **COUNT(campo)** – numero di record con *campo* non nullo;
- **SUM(campo)** somma tutti i valori di *campo* (*campo* numerico);
- **AVG(campo)** calcola la media aritmetica di *campo* (*campo* numerico);
- **MAX(campo)** trova il massimo di *campo* (*campo* numerico);
- **MIN(campo)** trova il minimo di *campo* (*campo* numerico);

SELECT COUNT(*) AS Tot, AVG(Tasse) as Media, MAX(Tasse) AS Max FROM Alunni

restituisce il numero di record (Tot), le Tasse medie (Media) e le Tasse massime (Max) della tabella Alunni

Con l'istruzione SELECT si possono fare le classiche **operazioni relazionali** dei DBMS (*Data Base Management System*) Relazionali: proiezione, selezione, congiunzione.

Intera Tabella - l'intera tabella,
tutti i record e tutti i campi

SELECT * FROM <i>tabella</i>	SELECT * FROM Materie
---	--

Elenco di tutti i record della tabella Materie

Proiezione - alcuni campi di tutti
i record

SELECT <i>campo_1, ..., campo_n</i> FROM <i>tabella</i>	SELECT CodMat, Descrizione FROM Materie
--	--

Elenco di CodMat e Descrizione di tutti i record della tabella Materie

Selezione - alcuni record filtrati
in base ad una condizione di
selezione

SELECT * FROM <i>tabella</i> WHERE <i>condizione</i>	SELECT * FROM Materie WHERE Ore_sett > 15
---	--

Elenco delle Materie – tutti i campi - con più 15 ore settimanali

Proiezione e Selezione - alcuni campi di alcuni record filtrati in base ad una condizione di selezione

```
SELECT campo_1, ..., campo_n
FROM tabella
WHERE condizione
```

```
SELECT CodMat, Descrizione
FROM Materie
WHERE Ore_sett > 15
```

Elenco di CodMat e Descrizione delle Materie con più 15 ore settimanali

Congiunzione - concatenazione dei record di due tabelle in associazione 1:N (legate da una chiave esterna) che hanno record correlati in entrambe le tabelle

```
SELECT tabella1.*, tabella2.*
FROM tabella1, tabella2
WHERE legame
      [AND condizione]
```

```
SELECT Alunni.*, Voti.*
FROM Alunni, Voti
WHERE
      Alunni.Matricola = Voti.Alunno
      AND Voti.Tipo = 'S'
```

Elenco degli Alunni e dei loro Voti relativi alle prove scritte (se un alunno non ha alcun voto scritto non sarà nel risultato)

La congiunzione di cui sopra - **equi join** - realizzata con la sintassi precedente produce un risultato "non aggiornabile". La equi join "aggiornabile" si realizza con la **inner join**, di cui esistono varianti come la **left join**, la **right join** e la **self join** qui di seguito esaminate.

Inner Join - concatenazione dei record di due tabelle in associazione 1:N (legate da una chiave esterna) che hanno record correlati in entrambe le tabelle

```
SELECT tabella1.*, tabella2.*
FROM tabella1
      INNER JOIN tabella2
      ON legame
[WHERE condizione]
```

```
SELECT Alunni.*, Voti.*
FROM Alunni
      INNER JOIN Voti
      ON Alunni.Matricola = Voti.Alunno
WHERE Voti.Tipo = 'S'
```

Elenco degli Alunni e dei loro Voti relativi alle prove scritte (se un alunno non ha alcun voto scritto non sarà nel risultato)

Left Join - concatenazione di tutti i record della prima tabella con quelli della seconda tabella in associazione 1:N, anche di quelli che non hanno record correlati nella seconda tabella

Elenco di tutti gli Alunni con i relativi Voti, quando presenti (se un alunno non ha alcun voto sarà comunque nel risultato)

```
SELECT tabella1.*, tabella2.*
FROM tabella1
      LEFT JOIN tabella2
            ON legame
[WHERE condizione]
```

```
SELECT Alunni.*, Voti.*
FROM Alunni
      LEFT JOIN Voti
            ON Alunni.Matricola = Voti.Alunno
```

Right Join - concatenazione di tutti i record della seconda tabella con quelli della prima tabella in associazione 1:N, anche di quelli che non hanno record correlati nella prima tabella

Elenco delle Materie con i relativi Voti, quando presenti (se una materia non ha alcun voto sarà comunque nel risultato)

```
SELECT tabella1.*, tabella2.*
FROM tabella1
      RIGHT JOIN tabella2
            ON legame
[WHERE condizione]
```

```
SELECT Voti.*, Materie.*
FROM Voti
      RIGHT JOIN Materie
            ON Voti.Materia = Materie.CodMat
```

Per poter illustrare la self join occorre modificare la tabella Alunni inserendo un nuovo campo – **Tutor** – che fa riferimento alla matricola del "tutor" dell'alunno, immaginando che ci siano degli alunni che ricoprono questa funzione. Si ha quindi un'associazione 1:N interna alla tabella Alunni.

Self Join - concatenazione dei record della tabella con tutti i record della tabella stessa ai quali sono legati da una relazione 1:N (occorre rinominare la tabella)

```
SELECT
alias1.campi, alias2.campi
FROM tabella AS alias1,
      tabella AS alias2
WHERE legame
```

```
SELECT Alu1.Nome, Alu2.*
FROM Alunni AS Alu1,
      Alunni AS Alu2
WHERE Alu1.Matricola = Alu2.Tutor
```

Elenco di tutti gli Alunni con indicato, per ciascuno di essi, il nome del rispettivo Tutor

La clausola **GROUP BY** serve per effettuare il raggruppamento delle righe che hanno uno stesso valore nelle colonne indicate per il raggruppamento stesso. Ogni diverso gruppo produce una sola riga di risultato.

I campi di raggruppamento che compaiono nella SELECT (unitamente alle funzioni di aggregazione) debbono comparire nella clausola GROUP BY. Nella SELECT non possono comparire campi non di raggruppamento.

Con la condizione della clausola **WHERE** si selezionano i record "prima" del loro raggruppamento, mentre con la clausola **HAVING** si selezionano i risultati del raggruppamento, "a posteriori", si controllano cioè i risultati (generalmente i valori restituiti dalle funzioni Count, Sum, Min, Avg, Max).

SELECT <i>campi_raggruppamento, funzioni_aggregazione</i> FROM <i>tabella</i> GROUP BY <i>campi_raggruppamento</i> [WHERE <i>condizione_a_priori</i>] [HAVING <i>condizione_a_posteriori</i>]	SELECT Corso, Count(*) AS Num, Sum(Tasse) AS Tot FROM Alunni GROUP BY Corso WHERE Città = 'Perugia' HAVING Count(*) > 10
---	---

Elenco dei Corsi - con numero alunni e totale tasse – frequentati dagli alunni di Perugia
che risultano avere più di 10 alunni perugini

Con la clausola **ANY** la condizione è verificata se la subquery restituisce almeno un valore che la soddisfa (viene da pensare all'operatore **OR**).

```
SELECT Matricola, Nome, Classe FROM Alunni  
WHERE Classe <> '3Atp' AND Tasse > ANY ( SELECT Tasse FROM Alunni WHERE Classe = '3Atp' )
```

oppure:

```
SELECT Matricola, Nome, Classe FROM Alunni  
WHERE Classe <> '3Atp' AND Tasse > ( SELECT Min(Tasse) FROM Alunni WHERE Classe = '3Atp' )
```

Elenco degli alunni non della 3Atp che pagano tasse maggiori di uno qualsiasi degli alunni della 3Atp

Con la clausola **ALL** la condizione è verificata se la subquery restituisce valori ciascuno dei quali la soddisfa (viene da pensare all'operatore **AND**).

```
SELECT Matricola, Nome, Classe FROM Alunni  
WHERE Classe <> '3Atp' AND Tasse > ALL ( SELECT Tasse FROM Alunni WHERE Classe = '3Atp' )
```

oppure:

```
SELECT Matricola, Nome, Classe FROM Alunni  
WHERE Classe <> '3Atp' AND Tasse > ( SELECT Max(Tasse) FROM Alunni WHERE Classe = '3Atp' )
```

Elenco degli alunni non della 3Atp che pagano tasse maggiori di qualsiasi alunno della 3Atp

Con la clausola **IN** la condizione è verificata se la subquery restituisce almeno un valore che la soddisfa (l'operatore **IN** equivale all'operatore **=ANY** e l'operatore **NOT IN** equivale a **<>ALL**).

```
SELECT * FROM Alunni WHERE Città IN ( SELECT Città FROM Alunni GROUP BY Città HAVING Count(*) < 30 )
```

oppure:

```
SELECT * FROM Alunni WHERE Città = ANY ( SELECT Città FROM Alunni GROUP BY Città HAVING Count(*) < 30 )
```

oppure:

```
SELECT * FROM Alunni WHERE Città NOT IN (SELECT Città FROM Alunni GROUP BY Città HAVING Count(*) ≥ 30)
```

oppure:

```
SELECT * FROM Alunni WHERE Città <>ANY (SELECT Città FROM Alunni GROUP BY Città HAVING Count(*) ≥ 30)
```

Elenco degli alunni che provengono da città dalle quali provengono meno di 30 alunni

Utilizzando le clausole **IN** e **Not IN** si possono fare anche operazioni relazionali di **intersezione** e **differenza**.

Supponiamo di avere la tabella **Nuoviscritti** - stessa struttura della tabella Alunni - nella quale transitano gli alunni prima di essere registrati definitivamente nella tabella Alunni.

```
SELECT * FROM Alunni WHERE Matricola IN ( SELECT Matricola FROM Nuoviscritti )
```

Intersezione - Elenco dei nuovi iscritti presenti nella tabella Alunni

```
SELECT * FROM Alunni WHERE Matricola NOT IN ( SELECT Matricola FROM Nuoviscritti )
```

Differenza - Elenco dei vecchi iscritti presenti nella tabella Alunni

Con la clausola **EXISTS** la condizione è verificata se la subquery restituisce un risultato non vuoto

```
SELECT * FROM Alunni AS A1 WHERE EXISTS  
    ( SELECT * FROM Alunni AS A2 WHERE A1.Nome = A2.Nome AND A1.Matricola <> A2.Matricola )
```

Elenco degli alunni che hanno degli omonimi

```
SELECT * FROM Alunni WHERE NOT EXISTS ( SELECT * FROM Alunni WHERE Città = 'Magione' )
```

Elenco di tutti gli alunni solo se non esistono alunni di Magione

Per terminare questa breve panoramica, vediamo una query (funzionante in Access) che usa la funzione Visual Basic IIF - **IIF(*condizione*, *vero*, *falso*)** -, analoga alla funzione SE di Excel

```
SELECT Materia,  
    SUM( IIF(Tipo='S',1,0) ) AS NumVotiScritto,  
    SUM( IIF(Tipo='O',1,0) ) AS NumVotiOrale,  
    SUM( IIF(Tipo='P',1,0) ) AS NumVotiPratico  
FROM Voti  
GROUP BY Materia;
```

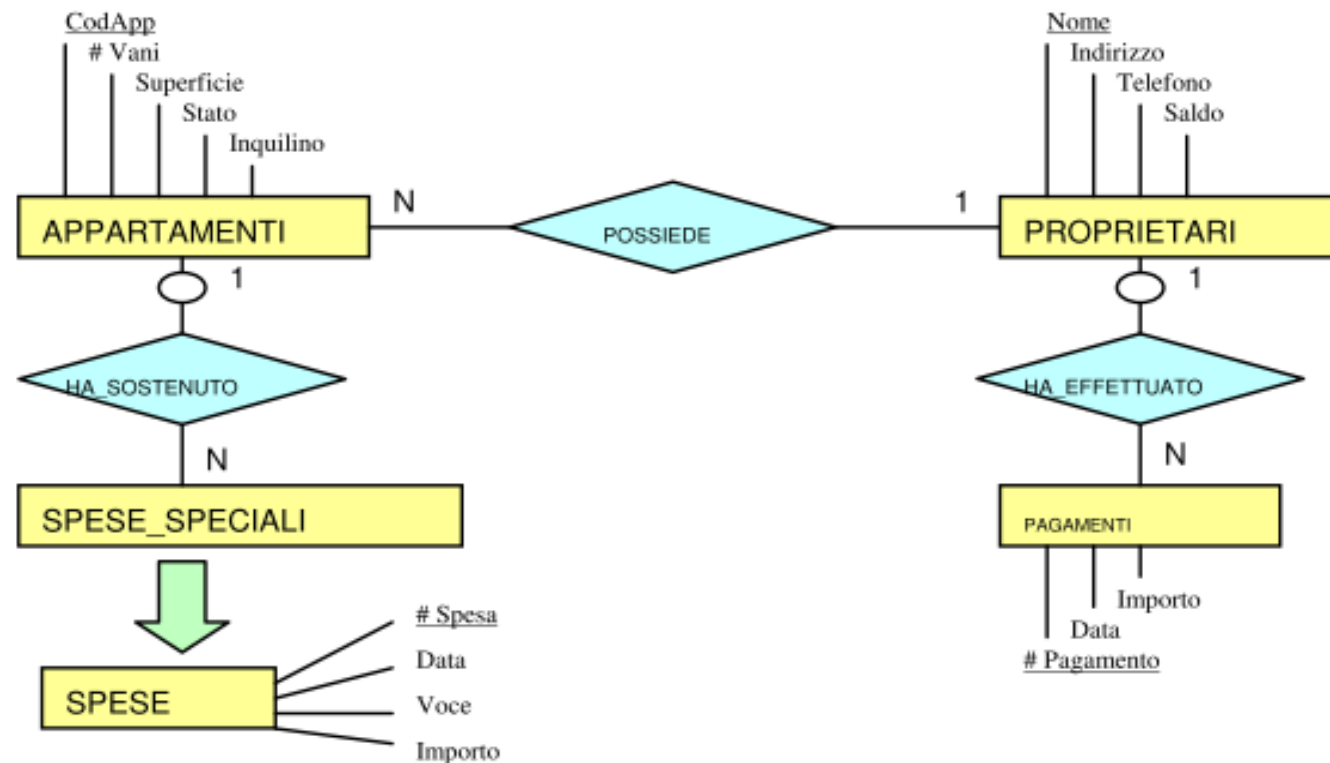
Totalizza, per ciascuna materia, il numero di valutazioni scritte, orali e pratiche

ESEMPIO: Gestione CONDOMINIO

La realtà che si prende in esame è quella relativa all'amministrazione di un condominio.

Il condominio ha un certo numero di appartamenti ciascuno dei quali può essere o meno occupato da un inquilino e ha un unico proprietario (che però può possedere più appartamenti). Il saldo di ogni proprietario nei confronti dell'amministrazione può essere positivo (se è a credito) o negativo (se è a debito). Tra le spese ci sono le spese speciali, quelle cioè che sono attribuibili solamente ad un determinato appartamento (e quindi ad un solo proprietario). I proprietari effettuano dei pagamenti periodici all'amministratore.

Lo SCHEMA CONCETTUALE di tale realtà è il seguente:



L'associazione ISA tra SPESE e SPESE_SPECIALI si risolve inserendo in SPESE un attributo opzionale (*CodApp*) che stabilisce a quale categoria appartiene la spesa (se indefinito è *spesa normale*, altrimenti è *spesa speciale*). Occorre inoltre prevedere una ulteriore relazione (ULTIMA) che contiene gli ultimi numeri di spese e pagamenti.

Uno schema relazionale che rappresenta il diagramma entità-associazioni precedente è:

APPARTAMENTI				
<u>CodApp</u>	Superficie	#Vani	Inquilino	Proprietario

PROPRIETARI			
<u>Proprietario</u>	Indirizzo	Telefono	Saldo

SPESE				
<u>#Spesa</u>	Data	Voce	Importo	CodApp

PAGAMENTI			
<u>#Pagamento</u>	Data	Importo	Proprietario

ULTIMA	
Ult#Pagamento	Ult#Spesa

In SQL i comandi per creare questo schema relazionale sono:

CREATE TABLE APPARTAMENTI

```
(   CodApp      char(4)      NOT NULL,  
    Superficie  decimal(4,3) NOT NULL,  
    #Vani        smallint     NOT NULL,  
    Inquilino   char(15),  
    Proprietario char(15)     NOT NULL )
```

CREATE UNIQUE INDEX INDCODAPP ON APPARTAMENTI (*CodApp*)

CREATE TABLE PROPRIETARI

```
(   Proprietario char(15)      NOT NULL,  
    Indirizzo    char(35)      NOT NULL,  
    Telefono     char(12),  
    Saldo         decimal(10)  NOT NULL )
```

CREATE UNIQUE INDEX INDPROPR ON PROPRIETARI (*Proprietario*)

CREATE TABLE PAGAMENTI

```
(   #Pagamento  integer       NOT NULL,  
    Data         char(6)       NOT NULL,  
    Importo      decimal(9)    NOT NULL,  
    Proprietario char(15)     NOT NULL )
```

CREATE UNIQUE INDEX INDPAGAM ON PAGAMENTI (*#Pagamento*)

CREATE TABLE **SPESE**

(*#Spesa* *integer* NOT NULL,
 Data *char(6)* NOT NULL,
 Importo *decimal(10)* NOT NULL,
 CodApp *char(4)*)

CREATE UNIQUE INDEX **INDSPESE** ON **SPESE** (*#Spesa*)

CREATE TABLE **ULTIMA**

(*Ult#Pagamento* *integer* NOT NULL,
 Ult#Spesa *integer* NOT NULL)

Vediamo ora alcune possibili **elaborazioni** che evidenziano la potenza di SQL:

- a) Elenco dei proprietari:

```
SELECT      *  
FROM        PROPRIETARI
```

- b) Nome ed indirizzo dei proprietari con saldo negativo:

```
SELECT      Proprietario, Indirizzo  
FROM        PROPRIETARI  
WHERE       Saldo < 0
```

- c) Nome e saldo dei proprietari di appartamenti con superficie maggiore di 120 mq:

```
SELECT DISTINCT Proprietario, Saldo  
FROM           APPARTAMENTI, PROPRIETARI  
WHERE          Superficie > 120      AND  
              APPARTAMENTI.Proprietario = PROPRIETARI.Proprietario
```

- d) Elenco di tutti gli appartamenti occupati (specificando il nome dell'inquilino):

```
SELECT      CodApp, Inquilino  
FROM        APPARTAMENTI  
WHERE       Inquilino IS NOT NULL
```

- e) Elenco delle spese normali (non speciali) sostenute per "lavori idraulici":

```
SELECT      *  
FROM        SPESE  
WHERE       CodApp IS NULL      AND   Voce = "lavori idraulici"
```


- l) Elenco appartamenti con un numero vani maggiore di ogni appartamento di "Rossi":

```
SELECT      *
FROM        APPARTAMENTI
WHERE       #Vani > ALL(      SELECT      #Vani
                                FROM        APPARTAMENTI
                                WHERE       Proprietario = "Rossi" )
```

- m) Elenco proprietari che non hanno effettuato alcun pagamento:

```
SELECT      Proprietario
FROM        PROPRIETARI
WHERE       NOT EXIST ( SELECT      *
                        FROM        PAGAMENTI
                        WHERE Proprietario = PROPRIETARI.Proprietario )
```

- n) Elenco degli appartamenti di "Rossi":

```
SELECT      COUNT(*)
FROM        APPARTAMENTI
WHERE       Proprietario = "Rossi"
```

- o) Calcolo delle spese sostenute a gennaio '94 per "pulizie":

```
SELECT      SUM(Importo)
FROM        SPESE
WHERE       Voce = "pulizie" AND Data ≥ "940101" AND Data ≤ "940131"
```

- p) Individuazione dell'appartamento che ha superficie maggiore:

```
SELECT      CodApp, Superficie
FROM        APPARTAMENTI
WHERE       Superficie = ( SELECT      MAX(Superficie)
                        FROM        APPARTAMENTI )
```

- q) Calcolo della superficie media degli appartamenti con più di 4 vani:

```
SELECT    AVG(Superficie)
FROM      APPARTAMENTI
WHERE     #Vani>4
```

- r) Calcolo della spesa sostenuta per ogni voce di spesa nel '94:

```
SELECT    Voce, SUM(Importo)
FROM      SPESE
WHERE     DataLIKE"94%"
GROUP BY  Voce
```

- s) Elenco dei proprietari che hanno effettuato più di 10 pagamenti dopo il 10 marzo 1993:

```
SELECT    Proprietario, SUM(Importo)
FROM      PAGAMENTI
WHERE     Data>"930310"
GROUP BY  Proprietario
HAVING    COUNT(*)>10
```

- t) Elenco dei proprietari (con relativo indirizzo) in ordine alfabetico:

```
SELECT      Proprietario, Indirizzo
FROM        PROPRIETARI
ORDERED BY  Proprietario
```

- u) Elenco appartamenti secondo il numero di vani e, a parità di vani, secondo la superficie:

```
SELECT      *
FROM        APPARTAMENTI
ORDERED BY  #Vani, Superficie   DESC
```

- v) Elenco in ordine alfabetico dei proprietari che nel '94 hanno sostenuto spese speciali per un importo superiore a 100.000 L. (con la specifica dell'importo della spesa):

```
SELECT Proprietario, SUM(Importo)
FROM SPESE, APPARTAMENTI
WHERE CodApp IN ( SELECT CodApp
                   FROM SPESE
                   WHERE CodApp IS NOT NULL AND Data LIKE "94%" )

GROUP BY Proprietario
HAVING   SUM(Importo) > 100.000
ORDERED BY Proprietario
```


- z) Registrazione di una spesa speciale (di 500.000 L.) a carico di un appartamento (A101) e da addebitare al relativo proprietario in data 05/08/93 [INTERATTIVO]:

```
SELECT    Ult#Spesa
FROM      ULTIMA                                (restituisce 757)
```

```
INSERT INTO    SPESE
               ( #Spesa, Data, Voce, Importo, CodApp )
               VALUE ( 757, "930805", "lavori edilizi", 500.000, "A101" )
```

```
UPDATE    ULTIMA
          SET      Ult#Spesa      =      Ult#Spesa + 1
```

```
SELECT    Proprietario
FROM      APPARTAMENTI
WHERE     CodApp = "A101"                        (restituisce Bianchi)
```

```
UPDATE    PROPRIETARI
          SET      Saldo =      Saldo - 500.000
          WHERE    Prprietario = "Bianchi"
```