ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI

PROGETTAZIONE DI DATABASE

Il linguaggio SQL

Lezione 11

Versione del: 17/12/2014 21.58.16
Revisione numero: 7

Prof. Andrea Zoccheddu
Dipartimento di Informatica

Immagine di copertina da: http://www.iamsterdam.com/en-GB/living/education/Dutch-Education-System

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

IL LINGUAGGIO SQL

INTRODUZIONE AL LINGUAGGIO SQL

Fonte (informazione tratta dal seguente sito) :

http://it.wikipedia.org/wiki/SQL

In informatica SQL (Structured Query Language) è un linguaggio standardizzato per database basati sul modello relazionale (RDBMS) progettato per:

- creare e modificare schemi di database (DDL - Data Definition Language);
- inserire, modificare e gestire dati memorizzati (DML - Data Manipulation Language);
- interrogare i dati memorizzati (DQL - Data Query Language);
- creare e gestire strumenti di controllo ed accesso ai dati (DCL - Data Control Language).

Nonostante il nome, non si tratta dunque solo di un semplice linguaggio di interrogazione, ma alcuni suoi sottoinsiemi si occupano della creazione, della gestione e dell'amministrazione del database.

STORIA

L'SQL nasce nel 1974 ad opera di Donald Chamberlin, nei laboratori dell'IBM. Nasce come strumento per lavorare con database che seguano il modello relazionale. A quel tempo però si chiamava SEQUEL (la corretta pronuncia IPA è ['ɛs'kju'ɛl], quella informale ['si:kwəl]). Nel 1975 viene sviluppato un prototipo chiamato SEQUEL-XRM; con esso si eseguirono sperimentazioni che portarono, nel 1977, a una nuova versione del linguaggio, che inizialmente avrebbe dovuto chiamarsi SEQUEL/2 ma che poi divenne, per motivi legali, SQL. Su di esso si sviluppò il prototipo System R, che venne utilizzato da IBM per usi interni e per alcuni suoi clienti. Ma, dato il suo successo, anche altre società iniziarono subito a sviluppare prodotti basati su SQL. Nel 1981 IBM iniziò a vendere alcuni prodotti relazionali e nel 1983 rilasciò DB2, il suo DBMS relazionale diffuso ancor oggi[quando?]. SQL divenne subito lo standard industriale per i software che utilizzano il modello relazionale.

L'ANSI lo adottò come standard fin dal 1986, senza apportare modifiche sostanziali alla versione inizialmente sviluppata da IBM. Nel 1987 la ISO fece lo stesso. Questa prima versione standard è denominata SQL/86. Negli anni successivi si realizzarono altre versioni, che furono SQL/89, SQL/92 e SQL/2003. Tale processo di standardizzazione mirava alla creazione di un linguaggio che funzionasse su tutti i DBMS (Data Base Management Systems) relazionali, ma questo obiettivo non fu raggiunto. Infatti, i vari produttori implementarono il linguaggio con numerose variazioni e, in pratica, adottarono gli standard ad un livello non superiore al minimo, definito dall'Ansi come Entry Level.

CARATTERISTICHE

SQL è un linguaggio per interrogare e gestire basi di dati mediante l'utilizzo di costrutti di programmazione denominati query. Con SQL si leggono, modificano, cancellano dati e si esercitano funzioni gestionali ed amministrative sul sistema dei database. La maggior parte delle implementazioni dispongono di interfaccia alla riga di comando per l'esecuzione diretta di comandi, in alternativa alla sola interfaccia grafica GUI.

Originariamente progettato come linguaggio di tipo dichiarativo, si è successivamente evoluto con l'introduzione di costrutti procedurali, istruzioni per il controllo di flusso, tipi di dati definiti dall'utente e varie altre estensioni del linguaggio. A partire dalla definizione dello standard SQL:1999 molte di queste estensioni sono state formalmente adottate come parte integrante di SQL nella sezione SQL/PSM dello standard.

Alcune delle critiche più frequenti rivolte ad SQL riguardano la mancanza di portabilità del codice fra vendor diversi, il modo inappropriato con cui vengono trattati i dati mancanti (NULL), e la semantica a volte inutilmente complicata.

PRIME INTERROGAZIONI

Una volta che abbiamo progettato ed implementato la Base di Dati, ed abbiamo inserito nelle tabelle i dati necessari (in modo anche da rappresentare totalità, parzialità e molteplicità) possiamo provare ad interrogare la base di dati.

Interrogare significa chiedere alla base di dati di mostrare non solo i dati che contiene, ma anche di filtrare solo quelli che riteniamo interessanti e di calcolare informazione aggiunta (come somme, medie, ecc…).

La classica interrogazione di una DB si dice **query**.

RIEPILOGO

SINTASSI GENERALE DELLA QUERY

La sintassi generale con cui scrivere un comando SELECT è la seguente:

```
SELECT [ ALL | DISTINCT ] lista_selezione
FROM lista_tabelle
[ WHERE condizione ]
[ GROUP BY lista_colonne ]
[ HAVING condizione]
[ ORDER BY lista_colonne ]
```

dove:

ALL e DISTINCT	Sono opzioni per specificare le si desiderano (o no) eventuali duplicati
lista_selezione	Per indicare tutti i campi usare * Per indicare eventuali campi scriverli separati da virgole È possibile usare operazioni su stringhe, su date ed algebriche È possibile usare funzioni di aggregazione
lista_tabelle	Indicare le tabelle necessarie per elaborare il risultato È possibile usare più tabelle separate da virgole (prodotto cartesiano) È possibile usare le clausole JOIN (INNER, OUTER, FULL)
WHERE	Clausola facoltativa (è possibile non usarla) Serve per filtrare i dati con un criterio
condizione	Espressione booleana che esprime il criterio di filtro La condizione ammette l'uso degli operatori logici NOT, AND OR
GROUP BY	Clausola facoltativa (è possibile non usarla) Serve per indicare i campi di raggruppamento con cui in seguito elaborare i dati aggregati
lista_colonne	Elenco separato da virgole dei campi di raggruppamento
HAVING	Clausola facoltativa (è possibile non usarla) Serve per indicare il criterio di filtro dei dati raggruppati
condizione	Espressione booleana che esprime il criterio di filtro È possibile filtrare in base al valore di una funzione di aggregazione (AVG, COUNT, MAX, MIN, SUM)
ORDER BY	Clausola facoltativa (è possibile non usarla) Serve per ordinare i record del risultato secondo un ordinamento crescente o decrescente
lista_colonne	Elenco separato da virgole dei campi di ordinamento Si osservi che è possibile ordinare anche per campi NON mostrati

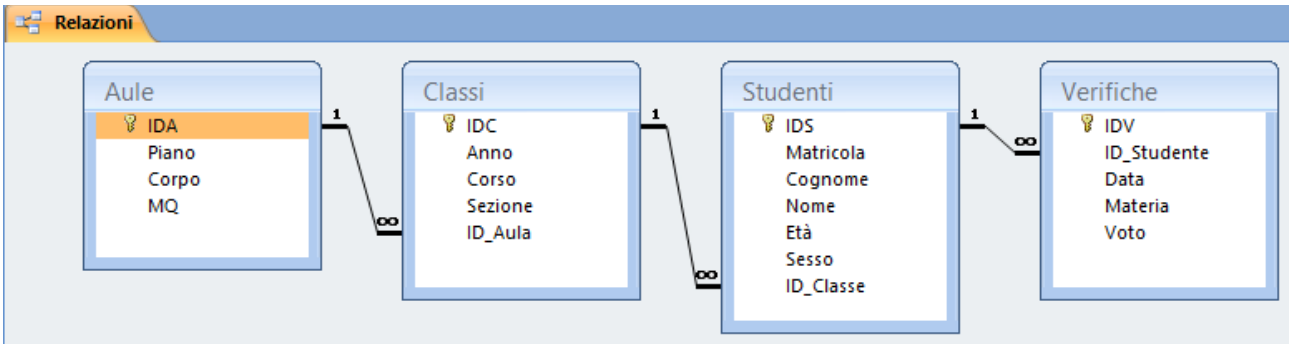
DB DI RIFERIMENTO

Per procedere con le query supponiamo di avere la base di dati implementata con uno strumento adatto (es. Access, mySQL, Oracle, ecc…).

Per esempio è possibile ipotizzare uno schema relazionale come il seguente:

```
AULE (IDA, Piano, Corpo, mq);
      PK IDA;
CLASSI (IDC, Anno, Corso, Sezione, Aula);
      PK IDC;
      AK (Anno, Corso, Sezione);
      FK Aula REF Aule(IDA);
STUDENTI (Matricola, Cognome, Nome, Età, Sesso, Classe);
      PK Matricola;
      FK Classe REF Classi(IDC);
VERIFICHE (IDV, Studente, Data, Materia, Voto);
      PK IDV;
      FK Studente REF Studenti(Matricola);
      NOT NULL Studente, Data, Materia, Voto;
```

In Access la situazione potrebbe apparire come la seguente:



Supponiamo per esempio che la tabella Studenti contenga i seguenti dati:

Aule

IDA	Piano	Corpo	MQ
1	1	A	25
2	1	A	36
3	1	B	25
4	2	B	36
5	2	A	25
6	2	A	49

Classi

IDC	Anno	Sezione	Corso	ID_Aula
1	5	A	INF	4
2	4	A	INF	3
3	5	B	INF	
4	5	A	TEL	2

Studenti

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3
8	12352	Piano	Guida	18	F	1

QUERY ELEMENTARI

La query è una richiesta di elaborazione al sistema formulata con il linguaggio SQL. La query deve avere almeno due clausole: la clausola SELECT e la FROM. La clausola SELECT serve per indicare cosa si desidera visualizzare, mentre FROM serve per specificare dove si trovano i dati.

Query di base

QUERY 1 - QUERY ELEMENTARE

Una query elementare potrebbe essere la seguente:

```
SELECT Cognome, Nome
FROM Studenti;
```

In questa query prelevo i dati dalla tabella Studenti e visualizzo due colonne: Cognome e Nome. Altri dati dei ciascun record, pur esistendo, non sono mostrati.

In questo caso la query precedente renderebbe:

Query1

Cognome	Nome
Sale	Rosa
Carta	Bianca
Marino	Pino
Piu	Mino
Marino	Pino
Carta	Rosa
Piano	Remo
Piano	Guida

QUERY 2 - QUERY ELEMENTARE

Vediamo un altro esempio:

```
SELECT Età
FROM Studenti;
```

Query2

Età
17
18
19
17
18
17
16
18

In questa query prelevo i dati dalla tabella Studenti e visualizzo una sola colonna.

I dati potrebbero risultare duplicati, come mostra l'esempio e non è possibile fare previsioni sull'ordinamento dei dati.

QUERY 3 - QUERY SENZA DUPLICATI

Si noti come nella query precedente siano mostrati anche valori duplicati (per esempio il 17 compare più volte). Se si desidera evitarli è possibile usare:

```
SELECT DISTINCT Età
FROM Studenti;
```

Query3

Età
16
17
18
19

che rende in una tabella dinamica l'elenco dei valori. Si deve fare attenzione che non è possibile fare previsioni sull'ordinamento dei valori.

QUERY 4 - MOSTRARE TUTTI I CAMPI

Per visualizzare tutti i campi di una tabella è possibile usare l'asterisco, invece di elencare i campi:

```
SELECT *
FROM Studenti;
```

Query4

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3
8	12352	Piano	Guida	18	F	1

che rende tutti i campi e tutti i record della tabella Studenti.

Si deve fare attenzione che non è possibile fare previsioni sull'ordinamento dei valori.

QUERY 5 - ORDINAMENTI

In una query generica i valori non sono ordinati, ma sono mostrati nell'ordine in cui il sistema li trova in archivio. Se si desidera ordinarli è possibile usare una clausola facoltativa, ORDER BY:

```
SELECT Cognome, Nome, Età
FROM Studenti
ORDER BY Nome;
```

Query5

Cognome	Nome	Età
Carta	Bianca	18
Piano	Guida	18
Piu	Mino	17
Marino	Pino	18
Marino	Pino	19
Piano	Remo	16
Carta	Rosa	17
Sale	Rosa	17

che rende i campi Cognome, Nome, Età di tutti i record della tabella Studenti, ma sono ordinati per campo Nome.

La query non impedisce di visualizzare duplicati e risulta casuale sull'ordinamento di altri campi (Cognome e Età nel nostro esempio).

QUERY 6 - ORDINAMENTI

Se desiderassi ordinare per più campi posso elencarli separati da virgole, come per esempio:

```
SELECT Cognome, Nome, Età
FROM Studenti
ORDER BY Cognome, Nome, Età;
```

Query6

Cognome	Nome	Età
Carta	Bianca	18
Carta	Rosa	17
Marino	Pino	18
Marino	Pino	19
Piano	Guida	18
Piano	Remo	16
Piu	Mino	17
Sale	Rosa	17

che rende i campi Cognome, Nome, Età di tutti i record della tabella Studenti, e sono ordinati prioritariamente per Cognome, poi (a parità di Cognome) per Nome e infine (a parità di Cognome e Nome) per Età.

QUERY 7 - ORDINAMENTI

È possibile ordinare per campi non visualizzati come nel seguente esempio:

```
SELECT Cognome, Nome
FROM Studenti
ORDER BY Età;
```

Query7

Cognome	Nome
Piano	Remo
Carta	Rosa
Piu	Mino
Sale	Rosa
Piano	Guida
Marino	Pino
Carta	Bianca
Marino	Pino

QUERY 8 - ORDINAMENTI

È possibile ordinare in modo decrescente usando l'opzione DESC, come nel seguente esempio:

```
SELECT Cognome, Nome
FROM Studenti
ORDER BY Età DESC;
```

Query8

Cognome	Nome
Marino	Pino
Piano	Guida
Marino	Pino
Carta	Bianca
Carta	Rosa
Piu	Mino
Sale	Rosa
Piano	Remo

QUERY 9 - RINOMINARE I CAMPI CON LA CLAUSOLA AS

È possibile rinominare le intestazioni di colonna della query con l'opzione AS, come nel seguente esempio:

```
SELECT Cognome, Nome, Sesso AS Genere
FROM Studenti;
```

Query9

Cognome	Nome	Genere
Sale	Rosa	F
Carta	Bianca	F
Marino	Pino	F
Piu	Mino	M
Marino	Pino	M
Carta	Rosa	F
Piano	Remo	M
Piano	Guida	F

FILTRI

Clausola WHERE

È anche possibile mostrare solo i record che corrispondono a certi criteri. In questo caso è necessario utilizzare la clausola WHERE (sempre dopo SELECT, FROM ma prima di ORDER BY).

QUERY 10 - FILTRO ELEMENTARE

Per mostrare solo gli studenti maschi potremmo eseguire la seguente query:

```
SELECT Cognome, Nome, Età
FROM Studenti
WHERE Sesso = 'M';
```

Query10

Cognome	Nome	Età
Piu	Mino	17
Marino	Pino	18
Piano	Remo	16

Se si usa l'operatore di confronto di uguaglianza occorre fare attenzione al formato del valore da confrontare con quelli dei campi. Nell'esempio è necessario racchiudere tra singoli apici il valore del carattere M che rappresenta il sesso maschile.

Al posto dell'operatore di eguaglianza (=) è possibile usare anche altri operatori di confronto, come

<	>	<=	>=	<>
Minore	Maggiore	Minore uguale	Maggiore uguale	Diverso

QUERY 11 - FILTRO CON OPERATORI DI CONFRONTO

Per esempio per mostrare solo gli studenti con età superiore a 17 potremmo eseguire la seguente query:

```
SELECT Cognome, Nome, Età
FROM Studenti
WHERE Età > 17;
```

Query11

Cognome	Nome	Età
Carta	Bianca	18
Marino	Pino	19
Marino	Pino	18
Piano	Guida	18

QUERY 12 - FILTRO CON OPERATORI DI CONFRONTO

Per esempio per mostrare solo gli studenti con età diversa da 17 potremmo eseguire la seguente query:

```
SELECT Cognome, Nome, Età
FROM Studenti
WHERE Età <> 17;
```

Query12

Cognome	Nome	Età
Carta	Bianca	18
Marino	Pino	19
Marino	Pino	18
Piano	Remo	16
Piano	Guida	18

Rispetto alla query precedente compare anche un valore 16.

QUERY 13 - FILTRO CON OPERATORI DI CONFRONTO SU STRINGHE

Per esempio per mostrare solo gli studenti con nome precedente la lettera P alfabeticamente potremmo eseguire la seguente query:

```
SELECT Cognome, Nome
FROM Studenti
WHERE Nome < 'P';
```

Query13

Cognome	Nome
Carta	Bianca
Piu	Mino
Piano	Guida

Operatore Like

Se si desidera effettuare ricerche di una sottostringa in una stringa è possibile usare l'operatore like. Questo operatore è in grado di verificare se una stringa è presente in una altra stringa e permette di usare dei simboli speciali (metacaratteri) che hanno i seguenti significati:

- % significa qualsiasi stringa (in Access si usa *)
- _ significa un singolo carattere qualsiasi (in Access si usa ?)

QUERY 14 - FILTRO CON OPERATORE LIKE

Per esempio per mostrare solo gli studenti il cui nome inizia con la lettera «**R**» potremmo avere la query:

versione SQL92

```
SELECT *
FROM Studenti
WHERE (Nome LIKE 'R%');
```

versione Access

```
SELECT *
FROM Studenti
WHERE (Nome LIKE 'R*');
```

Query14

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3

QUERY 15 - FILTRO CON OPERATORE LIKE

Per esempio per mostrare solo gli studenti il cui nome termina con la lettera «**a**» potremmo avere la query:

versione SQL92

```
SELECT *
FROM Studenti
WHERE (Nome LIKE '%a');
```

versione Access

```
SELECT *
FROM Studenti
WHERE (Nome LIKE '*a');
```

Query15

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
6	12351	Carta	Rosa	17	F	3
8	12352	Piano	Guida	18	F	1

QUERY 16 - FILTRO CON OPERATORE LIKE

Per esempio per mostrare solo gli studenti il cui nome **contiene** la lettera «**i**» potremmo avere la query:

versione SQL92

```
SELECT *
FROM Studenti
WHERE (Nome LIKE '%i%');
```

versione Access

```
SELECT *
FROM Studenti
WHERE (Nome LIKE '*i*');
```

Query16

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
8	12352	Piano	Guida	18	F	1

QUERY 17 - FILTRO CON OPERATORE LIKE

Per esempio per mostrare solo gli studenti il cui nome **è composto di 4 caratteri** potremmo avere la query:

Versione SQL92

```
SELECT *
FROM Studenti
WHERE (Nome LIKE '____');
```

Versione Access

```
SELECT *
FROM Studenti
WHERE (Nome LIKE '????');
```

Query17

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3

Operatore BETWEEN

Per mostrare i record il cui campo ha un valore compreso in un intervallo (estremi inclusi) si può usare l'operatore BETWEEN.

QUERY 18 - FILTRO CON OPERATORE BETWEEN

```
SELECT *
FROM Studenti
WHERE Età BETWEEN 17 AND 19;
```

Query18

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
6	12351	Carta	Rosa	17	F	3
8	12352	Piano	Guida	18	F	1

Mostra solo gli studenti la cui età è compresa tra 17 e 19

QUERY 19 - FILTRO CON OPERATORE BETWEEN

```
SELECT *
FROM Studenti
WHERE Nome BETWEEN 'B' AND 'P';
```

Query19

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
2	12346	Carta	Bianca	18	F	3
4	12347	Piu	Mino	17	M	
8	12352	Piano	Guida	18	F	1

Mostra solo gli studenti il cui nome è compresa tra le lettere B e P; ma attenzione che produce fino alla lettera P ma esclude le successive come Pino

Operatore IS NULL

Per mostrare i record il cui campo non ha un valore si deve usare l'operatore IS NULL. L'operatore rende vero se il campo è vuoto.

QUERY 20 - FILTRO CON OPERATORE IS NULL

```
SELECT *
FROM Studenti
WHERE ID_Classe IS NULL;
```

Query20

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
4	12347	Piu	Mino	17	M	

Mostra solo gli studenti senza la classe

QUERY 20B - FILTRO SBAGLIATO

Occorre fare attenzione che non è possibile usare l'operatore di uguaglianza con il valore NULL, poiché NULL è diverso da qualsiasi valore persino da altri valori NULL. Per esempio:

```
SELECT *
FROM Studenti
WHERE ID_Classe = NULL;
```

Query20

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
* (Nuovo)						

È sempre un recordset vuoto

QUERY 21 - FILTRO CON OPERATORI LOGICI BOOLEANI

```
SELECT *
FROM Studenti
WHERE Sesso = 'M' AND ID_Classe IS NOT NULL;
```

Query21

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
5	12349	Marino	Pino	18	M	1
7	12350	Piano	Remo	16	M	3

Mostra solo gli studenti maschi con una classe specificata

Operatori Logici

È anche possibile utilizzare gli operatori logici (NOT, AND, OR) per legare diverse condizioni tra loro. Per imporre precedenze logiche è possibile usare anche le parentesi tonde e costruire espressioni logiche nidificate. In assenza di parentesi tonde si usano le consuete precedenze (NOT precede tutti, poi AND precede lo OR).

QUERY 22 - FILTRO CON OPERATORI LOGICI BOOLEANI

```
SELECT *
FROM Studenti
WHERE Sesso = 'M' AND Età > 17;
```

Query22

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
5	12349	Marino	Pino	18	M	1

Mostra solo gli studenti maschi maggiorenni

QUERY 23 - FILTRO CON OPERATORI LOGICI BOOLEANI

```
SELECT *
FROM Studenti
WHERE NOT Sesso = 'M' AND Età > 17;
```

Query23

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
8	12352	Piano	Guida	18	F	1

Mostra solo le studentesse maggiorenni

QUERY 24 - FILTRO CON OPERATORI LOGICI BOOLEANI

```
SELECT *
FROM Studenti
WHERE NOT ( Sesso = 'M' AND Età > 17 ) ;
```

Query24

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3
8	12352	Piano	Guida	18	F	1

Mostra solo gli studenti che non sono contemporaneamente Maschi e maggiorenni (quindi tutte le femmine e tutti i minorenni)

ALTRI OPERATORI

Rinominare le tabelle con la clausola As

Talvolta può essere comodo rinominare una tabella (per esempio con una sola lettera) nella clausola FROM per poi usare questo nome (in sostituzione della tabella) in tutte le altre clausole. Per esempio:

QUERY 30 - RINOMINAZIONE DI TABELLA

```
SELECT S.*
FROM Studenti AS S
WHERE NOT S.Nome LIKE '*a*' OR NOT S.ID_Classe IS NULL ;
```

Query30

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3
8	12352	Piano	Guida	18	F	1

Mostra solo gli studenti che non contengono la lettera «a» oppure non hanno classe nulla (in pratica tutti)

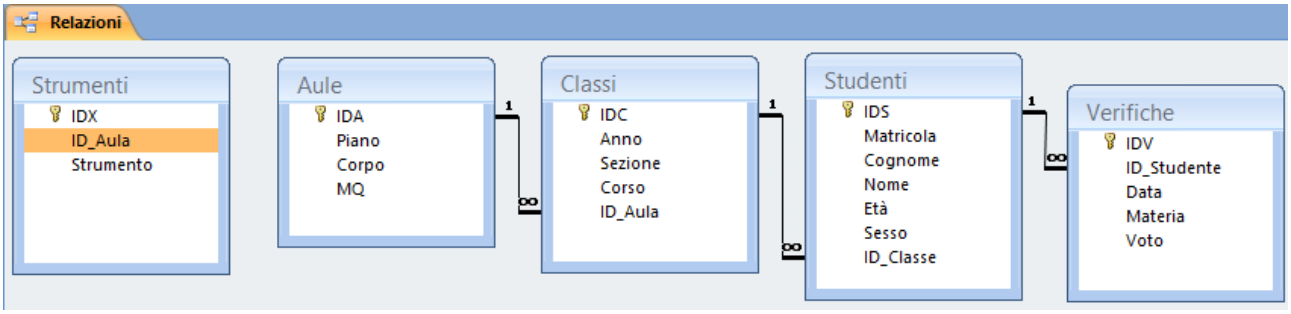
COLLEGARE PIÙ TABELLE

JOIN (GIUNZIONI)

Si è visto che nel database le informazioni sono sparse in più tabelle e che in alcuni casi occorre valutare insieme queste tabelle per ricostruire l'informazione complessiva. Per questo oltre a interrogare il database su una sola tabella è possibile anche farlo su più tabelle. In particolare questa esigenza appare necessaria quando due tabelle sono correlate attraverso un vincolo di chiave esterna, ovvero quando i dati di un campo devono essere collegati coi dati del campo di un'altra tabella.

Il linguaggio SQL permette di collegare le tabelle in modo da ottenere una struttura dei dati ampia e articolata.

Il database a cui faremo riferimento è il precedente con una modifica di comodo per alcune analisi particolari:



Dove i dati saranno i seguenti:

Strumenti

IDX	ID_Aula	Strumento
1	3	LIM
2		LIM
3	1	PC i7
4	1	LIM

L'assenza di valore è un valore NULL

Aule

IDA	Piano	Corpo	MQ
1	1	A	25
2	1	A	36
3	1	B	25
4	2	B	36
5	2	A	25
6	2	A	49

Classi

IDC	Anno	Sezione	Corso	ID_Aula
1	5	A	INF	4
2	4	A	INF	3
3	5	B	INF	
4	5	A	TEL	2

L'assenza di valore è un valore NULL

PRODOTTO CARTESIANO

Il prodotto cartesiano è il modo più semplice per combinare due (o più) diverse tabelle. Esso corrisponde ad una regola matematica che mette in relazione gli elementi di un insieme con quelli di un altro. In matematica il Piano Cartesiano rappresenta il prodotto cartesiano dell'insieme R con sé stesso ($RxR = R^2$).

Il prodotto cartesiano di due tabelle restituisce una tabella con un numero di colonne pari alle colonne delle tabelle moltiplicate e ciascuna riga è ottenuta giustapponendo (mettendo affianco) ciascuna riga della prima tabella con ciascuna riga della seconda. Quindi la tabella risultato ha un numero di righe pari al prodotto del numero di righe delle due tabelle moltiplicate.

Prodotto Cartesiano

Con il primo esempio vogliamo costruire una query che restituisce il prodotto cartesiano tra le tabelle Aule e Classi; poiché abbiamo 6 righe nella tabella Aule e 4 righe nella tabella Classi il prodotto consisterà in 24 righe. La query che rende il prodotto cartesiano è il seguente:

QUERY 40. - PRODOTTO CARTESIANO

```
SELECT *
FROM Aule, Classi ;
```

Query40

IDA	Piano	Campo	MQ	IDC	Anno	Sezione	Corso	ID_Aula
1	1	A	25	1	5	A	INF	4
1	1	A	25	2	4	A	INF	3
1	1	A	25	3	5	B	INF	
1	1	A	25	4	5	A	TEL	2
2	1	A	36	1	5	A	INF	4
2	1	A	36	2	4	A	INF	3
2	1	A	36	3	5	B	INF	
2	1	A	36	4	5	A	TEL	2
3	1	B	25	1	5	A	INF	4
3	1	B	25	2	4	A	INF	3
3	1	B	25	3	5	B	INF	
3	1	B	25	4	5	A	TEL	2
4	2	B	36	1	5	A	INF	4
4	2	B	36	2	4	A	INF	3
4	2	B	36	3	5	B	INF	
4	2	B	36	4	5	A	TEL	2
5	2	A	25	1	5	A	INF	4
5	2	A	25	2	4	A	INF	3
5	2	A	25	3	5	B	INF	
5	2	A	25	4	5	A	TEL	2
6	2	A	49	1	5	A	INF	4
6	2	A	49	2	4	A	INF	3
6	2	A	49	3	5	B	INF	
6	2	A	49	4	5	A	TEL	2

Combina ogni riga della tabella Aule (le prime 4 colonne del record set qui a lato) con ogni riga della tabella Classi (le ultime 5 colonne).

Commento

Il prodotto cartesiano è un risultato utile ed interessante ma ha il difetto di correlare anche quei dati che non hanno connessioni logiche tra loro. Quando si desidera mostrare solo quei dati che hanno un collegamento tra loro occorre impostare un filtro rispetto a questo prodotto e ottenere solo i dati ritenuti interessanti.

Vediamo come …

GIUNZIONE NELLA CLAUSOLA WHERE

Un modo per filtrare i dati del prodotto cartesiano è di impostare un vincolo nella clausola WHERE.

QUERY 41. - GIUNZIONE NELLA CLAUSOLA WHERE

```
SELECT *
FROM Classi, Aule
WHERE Classi.ID_Aula = Aule.IDA ;
```

Query41

IDC	Anno	Sezione	Corso	ID_Aula	IDA	Piano	Campo	MQ
4	5	A	TEL	2	2	1	A	36
2	4	A	INF	3	3	1	B	25
1	5	A	INF	4	4	2	B	36

Il vincolo del filtro lega i valori delle due colonne centrali, che come si vede sono identici.

La giunzione è un filtro che si applica sul prodotto cartesiano e che convalida solo le righe in cui i due campi eguagliati contengono valori identici.

La giunzione è solitamente applicata tra i campi chiave primaria e chiave esterna di due tabelle correlate, ma è possibile giungere campi che non sono necessariamente chiavi, come nel seguente esempio:

QUERY 42. - GIUNZIONE NELLA CLAUSOLA WHERE II

Classi

IDC	Anno	Sezione	Corso	ID_Aula
1	5	A	INF	4
2	4	A	INF	3
3	5	B	INF	
4	5	A	TEL	2

Strumenti

IDX	ID_Aula	Strumento
1	3	LIM
2		LIM
3	1	PC i7
4	1	LIM

```
SELECT *
FROM Classi, Strumenti
WHERE Classi.ID_Aula = Strumenti.ID_Aula ;
```

Query42

IDC	Anno	Sezione	Corso	Classi.ID_Aula	IDX	Strumenti.ID_Aula	Strumento
2	4	A	INF	3	1	3	LIM

L'unico record che soddisfa la giunzione tra Classi e Strumenti

Altre query tra più tabelle

Oltre a eguagliare i campi è ovviamente possibile utilizzare altri tipi di operazione come ad esempio i seguenti:

<	>	<=	>=	<>
Minore	**Maggiore**	**Minore uguale**	**Maggiore uguale**	**Diverso**

Per esempio:

QUERY 43. - DUE TABELLE DISGIUNTE

```
SELECT *
FROM Classi, Strumenti
WHERE Classi.ID_Aula <> Strumenti.ID_Aula ;
```

Query43

IDC	Anno	Sezione	Corso	Classi.ID_Aula	IDX	Strumenti.ID_Aula	Strumento
1	5	A	INF	4	1	3	LIM
4	5	A	TEL	2	1	3	LIM
1	5	A	INF	4	3	1	PC i7
2	4	A	INF	3	3	1	PC i7
4	5	A	TEL	2	3	1	PC i7
1	5	A	INF	4	4	1	LIM
2	4	A	INF	3	4	1	LIM
4	5	A	TEL	2	4	1	LIM

È il complementare della query precedente.

Queste modalità possono essere molto utili per filtrare dei dati e rispondere a determinate richieste.

GIUNZIONE NELLA CLAUSOLA FROM

La giunzione tra due tabelle comunque (ovvero quando si eguagliano i valori di due campi) è particolarmente importante specie in presenza di una chiave esterna da correlare alla chiave primaria. Questa rilevanza ha promosso lo sviluppo di specifiche clausole di giunzione, dette in inglese JOIN, che correlano dati uguali o nulli.

INNER JOIN

La giunzione interna (INNER JOIN) è quella più stretta e coincide con la query già vista:

QUERY 44. - INNER JOIN TRA DUE TABELLE

```
SELECT *
FROM Classi INNER JOIN Aule ON Classi.ID_Aula = Aule.IDA;
```

Query44

IDC	Anno	Sezione	Corso	ID_Aula	IDA	Piano	Campo	MQ
4	5	A	TEL	2	2	1	A	36
2	4	A	INF	3	3	1	B	25
1	5	A	INF	4	4	2	B	36

Giunzione tra Classi ed Aule

Si noti che solo i record con valori identici tra le due chiavi vincolate in eguaglianza

QUERY 45. - INNER JOIN TRA DUE TABELLE

```
SELECT *
FROM Classi INNER JOIN Aule ON Classi.ID_Aula = Aule.IDA;
```

Query44

IDC	Anno	Sezione	Corso	ID_Aula	IDA	Piano	Campo	MQ
4	5	A	TEL	2	2	1	A	36
2	4	A	INF	3	3	1	B	25
1	5	A	INF	4	4	2	B	36

Giunzione tra Classi ed Aule

Si noti che solo i record con valori identici tra le due chiavi vincolate in eguaglianza

Si ribadisce che le due query (con giunzione in clausola WHERE e con INNER JOIN) sono equivalenti. La seconda però è più esplicita e utilizza uno standard dei linguaggi SQL che evidenzia il modo di collegamento tra tabelle.

In caso di presenza di valori nulli in uno dei due campi NON si mostrano le righe del prodotto cartesiano. Quindi sono validi sono i valori esattamente coincidenti. Vediamo un caso con valori nulli:

QUERY 46. - GIUNZIONE NELLA CLAUSOLA FROM

Classi

IDC	Anno	Sezione	Corso	ID_Aula
1	5	A	INF	4
2	4	A	INF	3
3	5	B	INF	
4	5	A	TEL	2

Strumenti

IDX	ID_Aula	Strumento
1	3	LIM
2		LIM
3	1	PC i7
4	1	LIM

```
SELECT *
FROM Classi INNER JOIN Strumenti ON Classi.ID_Aula = Strumenti.ID_Aula ;
```

Query46

IDC	Anno	Sezione	Corso	Classi.ID_Aula	IDX	Strumenti.ID_Aula	Strumento
2	4	A	INF	3	1	3	LIM

L'unico record che soddisfa la giunzione tra Classi e Strumenti

LEFT OUTER JOIN

In alcuni casi può essere utile mostrare tutte le righe di una delle due tabelle anche se non correlati. Consideriamo ad esempio il seguente caso:

QUERY 47. - GIUNZIONE ESTERNA SINISTRA

```
SELECT *
FROM Classi LEFT OUTER JOIN Strumenti ON Classi.ID_Aula = Strumenti.ID_Aula ;
```

Query47

IDC	Anno	Sezione	Corso	Classi.ID_Aula	IDX	Strumenti.ID_Aula	Strumento
3	5	B	INF				
4	5	A	TEL	2			
2	4	A	INF	3	1	3	LIM
1	5	A	INF	4			

Mostra tutti i record della tabella **Classi** anche se non sono congiunte con quelle della tabella **Strumenti**

QUERY 48. - GIUNZIONE ESTERNA SINISTRA

```
SELECT *
FROM Classi LEFT OUTER JOIN Aule ON Classi.ID_Aula = Aule.IDA;
```

Query48

IDC	Anno	Sezione	Corso	ID_Aula	IDA	Piano	Campo	MQ
3	5	B	INF					
4	5	A	TEL	2	2	1	A	36
2	4	A	INF	3	3	1	B	25
1	5	A	INF	4	4	2	B	36

Mostra tutti i record della tabella **Classi** anche se non sono congiunte con quelle della tabella **Aule**

RIGHT OUTER JOIN

Lo speculare della precedente è la giunzione esterna destra:

QUERY 49. - GIUNZIONE ESTERNA DESTRA

```
SELECT *
FROM Classi RIGHT OUTER JOIN Strumenti ON Classi.ID_Aula = Strumenti.ID_Aula ;
```

Query49

IDC	Anno	Sezione	Corso	Classi.ID_	IDX	Strumenti.II	Strumento
					2		LIM
					3	1	PC i7
					4	1	LIM
2	4	A	INF	3	1	3	LIM

Mostra tutti i record della tabella **Strumenti** anche se non sono congiunte con quelle della tabella **Classi**

QUERY 50. - GIUNZIONE ESTERNA COMPLETA

```
SELECT *
FROM  Classi , Strumenti
WHERE Classi.ID_Aula = Strumenti.ID_Aula
OR    Classi.ID_Aula IS NULL
OR    Strumenti.ID_Aula IS NULL ;
```

Query50

IDC	Anno	Sezione	Corso	Classi.ID_	IDX	Strumenti.II	Strumento
2	4	A	INF	3	1	3	LIM
3	5	B	INF		1	3	LIM
1	5	A	INF	4	2		LIM
2	4	A	INF	3	2		LIM
3	5	B	INF		2		LIM
4	5	A	TEL	2	2		LIM
3	5	B	INF		3	1	PC i7
3	5	B	INF		4	1	LIM

Mostra tutti i record della tabella **Strumenti** congiunti con quelli della tabella **Classi** ma anche quei record coi valori nulli

AGGREGAZIONE DEI DATI

FUNZIONI DI AGGREGAZIONE

Una delle opportunità offerte dal linguaggio SQL è quella di ottenere dati ad informazione aggiunta come elaborazione di dati elementari. Si è già visto in precedenza che SQL permette di eseguire operazioni algebriche sui campi (somma, sottrazione, divisione, moltiplicazione). Queste operazioni sono eseguite su ciascun record della tabella che si sta esaminando.

In questa lezione vedremo invece come sia possibile ottenere dati su più record della tabella ed in particolare vediamo quelle operazioni che SQL permette di effettuare tramite operazioni predefinite.

FUNZIONI DI AGGREGAZIONE

Le operazioni predefinite che SQL offre sono le seguenti cinque: minimo, massimo, media, conteggio e somma. Queste operazioni sono chiamate FUNZIONI DI AGGREGAZIONE.

Ciascuna di questo operazioni si riferisce ad una colonna di una tabella.

AVG (MEDIA)

La funzione AVG (AVERAGE) restituisce la media dei valori di una colonna.

QUERY 51. - FUNZIONE DI AGGREGAZIONE AVG

```
SELECT AVG(Età) AS EtaMedia
FROM Studenti;
```

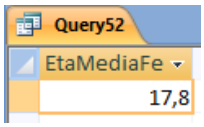
Query51

EtaMedia
17,5

Mostra la media delle età di tutti gli studenti

QUERY 52. - FUNZIONE DI AGGREGAZIONE AVG

```
SELECT AVG(Età) AS EtaMediaFemmine
FROM Studenti
WHERE Sesso = 'F';
```



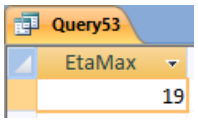
Mostra l'età media delle studentesse

MAX (MASSIMO)

La funzione MAX (MAXIMUM) restituisce il massimo dei valori di una colonna.

QUERY 53. - FUNZIONE DI AGGREGAZIONE MAX

```
SELECT MAX(Età) AS EtaMax
FROM Studenti;
```

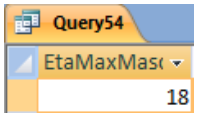


Mostra l'età dello studente più vecchio

QUERY 54. - FUNZIONE DI AGGREGAZIONE MAX

È anche possibile eseguire l'operazione solo su alcuni dati impostando un filtro (che viene eseguito prima di compiere il calcolo).

```
SELECT MAX(Età) AS EtaMaxMaschi
FROM Studenti
WHERE Sesso = 'M';
```

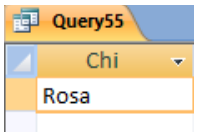


Mostra l'età dello studente maschio più vecchio

QUERY 55. - FUNZIONE DI AGGREGAZIONE MAX

È possibile trovare il massimo di una data (data più recente) oppure di una stringa (quella più avanti nel vocabolario). Per esempio:

```
SELECT MAX(Nome) AS Chi
FROM Studenti
WHERE Sesso = 'F';
```



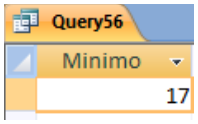
Mostra l'ultimo nome in ordine alfabetico tra le donne

MIN (MINIMO)

La funzione MIN (minimum) restituisce il minimo dei valori di una colonna. Come per il massimo è possibile filtrare i dati prima di calcolarlo ed è possibile eseguire l'elaborazione sia su date che su stringhe. Per esempio:

QUERY 56. - FUNZIONE DI AGGREGAZIONE MIN

```
SELECT MIN(Età) AS Minimo
FROM Studenti
WHERE Sesso = 'F';
```



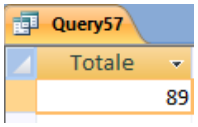
Mostra l'età della studentessa più giovane

SUM (SOMMA)

La funzione SUM restituisce la somma dei valori numerici di una colonna. Come per il massimo è possibile filtrare i dati prima di calcolarlo. Per esempio:

QUERY 57. - FUNZIONE DI AGGREGAZIONE SUM

```
SELECT SUM(Età) AS Totale
FROM Studenti
WHERE Sesso = 'F';
```



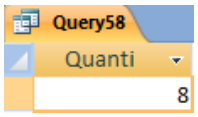
Mostra la somma delle età di tutte le studentesse

COUNT (CONTEGGIO)

La funzione COUNT restituisce il numero dei valori non nulli di una colonna. L'argomento può essere un campo oppure l'asterisco per indicare il numero delle righe. Come per il massimo è possibile filtrare i dati prima di calcolarlo.

QUERY 58. - FUNZIONE DI AGGREGAZIONE COUNT

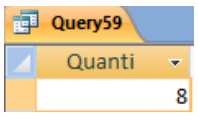
```
SELECT COUNT(*) AS Quanti
FROM Studenti ;
```



Mostra quanti sono gli studenti

QUERY 59. - FUNZIONE DI AGGREGAZIONE COUNT

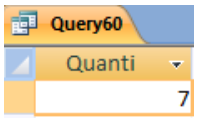
```
SELECT COUNT(Età) AS Quanti
FROM Studenti ;
```



Mostra quante sono le età non nulle degli studenti

QUERY 60. - FUNZIONE DI AGGREGAZIONE COUNT

```
SELECT COUNT(ID_Classe) AS Quanti
FROM Studenti ;
```



Mostra quanti sono gli studenti che hanno ID_Classe non nulla

COUNT DISTINCT (CONTEGGIO DEI DIVERSI)

La funzione COUNT calcola anche i valori duplicati presenti nella colonna. Se invece si desidera evitare di conteggiare più volte i valori ripetuti occorre specificarlo con la clausola DISTINCT. Per esempio:

```
SELECT COUNT (DISTINCT Eta) AS QuanteEtà
FROM Studenti ;
```

ma non è implementata in Access.

RAGGRUPPAMENTO DEI DATI

Finora i dati sono stati elaborati complessivamente. In alcuni casi però è interessante calcolare le funzioni di aggregazione su **sottoinsiemi dei dati**. Per esempio immaginiamo la domanda: quanti sono gli studenti maschi e quelli femmine?

Potremmo calcolare due distinte query ma è molto più comodo poterli riassumere con un'unica query. Per questo esiste la possibilità di raggruppare i dati secondo i valori di una o più colonne specificate. Questa clausola è la **GROUP BY**.

La GROUP BY richiede di indicare una o più colonne di raggruppamento. La sintassi generale è:

```
SELECT Lista Elementi
FROM Tabelle
WHERE Filtro prima del raggruppamento
GROUP BY campi di raggruppamento
HAVING Filtro dopo il raggruppamento;
```

GROUP BY

Il modo più semplice di usare la GROUP BY è il seguente:

```
SELECT Lista Elementi
FROM Tabelle
GROUP BY campi di raggruppamento;
```

per esempio la query seguente:

QUERY 61. - RAGGRUPPAMENTI

```
SELECT Sesso , COUNT(*) AS Quanti
FROM Studenti
GROUP BY Sesso;
```

Query61

Sesso	Quanti
F	5
M	3

Numero di studenti per genere

Per prima cosa raggruppa tutti gli studenti in categorie a seconda del valore del campo Sesso. In questo caso ci sono due sole categorie: M e F . Dopo aver raggruppato i dati la query procede a calcolare quante righe costituiscono ciascuna categoria.

QUERY 62. - RAGGRUPPAMENTI

```
SELECT Età , COUNT(*) AS Quanti
FROM Studenti
GROUP BY Età;
```

Query62

Età	Quanti
16	1
17	3
18	3
19	1

Numero di studenti per fascia d'età

Per prima cosa raggruppa tutti gli studenti in categorie a seconda del valore del campo Età. In questo caso esamina i campi e verifica quante diverse età ci sono. Poi per ogni valore trovato, si raggruppano i dati e si procede a calcolare quante righe costituiscono ciascuna categoria.

QUERY 63. - RAGGRUPPAMENTI

```
SELECT Sesso , MAX(Età) AS Vecchio, MIN(Età) AS Giovane, AVG(Età) AS Media
FROM Studenti
GROUP BY Sesso;
```

Query63

Sesso	Vecchio	Giovane	Media
F	19	17	17,8
M	18	16	17

Età massima minima e media per sesso

Ci sono due categorie di sesso: M e F . Dopo aver raggruppato i dati la query procede a calcolare le funzioni di aggregazione per ciascuna categoria.

GROUP BY E WHERE

Se si usa la clausola WHERE essa viene elaborata **prima di raggruppare** i dati.

Per esempio:

QUERY 64. - RAGGRUPPAMENTI E FILTRI WHERE

```
SELECT Età , COUNT(*) AS Quanti
FROM  Studenti
WHERE Nome LIKE 'R*'
GROUP BY Età;
```

Query64

Età	Quanti
16	1
17	2

Quanti studenti che iniziano per R ci sono per fasce d'età

Per iniziare si filtrano i dati secondo la clausola WHERE e restano solo 3 righe; poi raggruppa per categorie di sesso: M e F . Infine dopo aver raggruppato i dati la query procede a calcolare le funzioni di aggregazione per ciascuna categoria.

QUERY 65. - RAGGRUPPAMENTI E FILTRI WHERE

```
SELECT ID_Classe, COUNT(*) AS Quanti
FROM  Classi INNER JOIN Studenti ON Classi.IDC = Studenti.ID_Classe
WHERE NOT Nome LIKE 'R*'
GROUP BY ID_Classe;
```

Query65

ID_Classe	Quanti
1	3
3	1

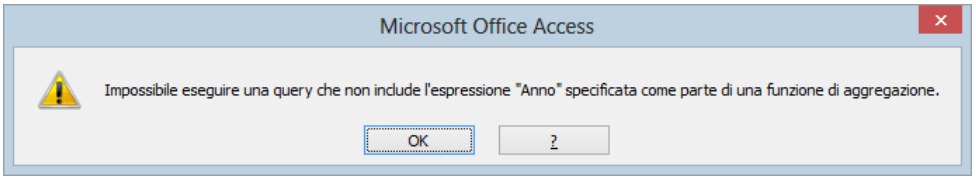
Quanti studenti che non iniziano per R ci sono in ogni classe?

Per prima cosa esegue la giunzione interna, poi il filtro, poi raggruppa per ID_Classe e infine calcola la funzione di aggregazione

Un limite della clausola di raggruppamento è che non è possibile visualizzare nella clausola SELECT campi che non siano di raggruppamento; per esempio se volessi mostrare oltre alla chiave primaria di ciascuna classe anche la sua dicitura (Anno, Sezione e Corso) avrei un errore se cercassi di farlo nel seguenti modo:

QUERY 65B - ERRORE DI RAGGRUPPAMENTO

```
SELECT Anno, Sezione, Corso, COUNT(*) AS Quanti
FROM  Classi INNER JOIN Studenti ON Classi.IDC = Studenti.ID_Classe
GROUP BY ID_Classe;
```



La soluzione non è sempre banale, ma in questo caso è possibile provare nel seguente modo:

QUERY 66. - RAGGRUPPAMENTO SU PIÙ CAMPI

```
SELECT Anno, Sezione, Corso, COUNT(*) AS Quanti
FROM  Classi INNER JOIN Studenti ON Classi.IDC = Studenti.ID_Classe
GROUP BY ID_Classe, Anno, Sezione, Indirizzo;
```

Query66

Anno	Sezione	Corso	Quanti
5	A	INF	4
5	B	INF	3

Quanti studenti ci sono in ogni classe?

FILTRO DEL RAGGRUPPAMENTO

Dopo che il raggruppamento è stato effettuato è possibile anche filtrare i dati aggregati. Il filtro agisce in maniera analoga alla clausola WHERE con la differenza che viene eseguito dopo aver raggruppato i dati e che può filtrare anche in base al risultato di una funzione di aggregazione. Il filtro è specificato con la clausola HAVING (in inglese significa: *che ha…*)

QUERY 67. - CLAUSOLA HAVING

Quanti studenti ci sono in ogni classe dove ce ne sono almeno 2?

```
SELECT Età, COUNT(*) AS QuantiSono
FROM Studenti
GROUP BY Età
HAVING COUNT(*) > 1;
```

Query67

Età	QuantiSono
17	3
18	3

Quanti studenti ci sono in ogni classe dove ce ne sono almeno 2?

QUERY 68. - CLAUSOLA HAVING

Quale è l'età media degli studenti delle classi dove ci sono almeno 2 studenti?

```
SELECT Anno, Sezione, Corso, AVG(Età) AS EtaMedia
FROM Classi, Studenti
WHERE Classi.IDC = Studenti.ID_Classe
GROUP BY Anno, Sezione, Corso
HAVING COUNT(*) > 1;
```

Query68

Anno	Sezione	Corso	EtaMedia
5	A	INF	18
5	B	INF	17

Quale è l'età media degli studenti delle classi dove ci sono almeno 2 studenti?

QUERY 69. - CLAUSOLA HAVING

```
SELECT Anno, Sezione, Corso, AVG(Età) AS EtaMedia
FROM Classi, Studenti
WHERE Classi.IDC = Studenti.ID_Classe
GROUP BY Anno, Sezione, Corso
HAVING EtaMedia > 17 ;
```

Osservazioni:

1) non è possibile usare la clausola HAVING senza una clausola GROUP BY; quindi se si vuole usare la HAVING è necessario usare anche la GROUP BY; in viceversa non è vero;
2) non è possibile usare funzioni di aggregazione nella clausola WHERE mentre si possono usare nella sola clausola HAVING; quindi se occorre filtrare su un valore aggregato è indispensabile usare sia la clausola GROUP BY che quella HAVING
3) se si usa un filtro su un campo nella clausola HAVING si ottiene un errore, salvo che sia un campo di raggruppamento;

QUERY ANNIDATE

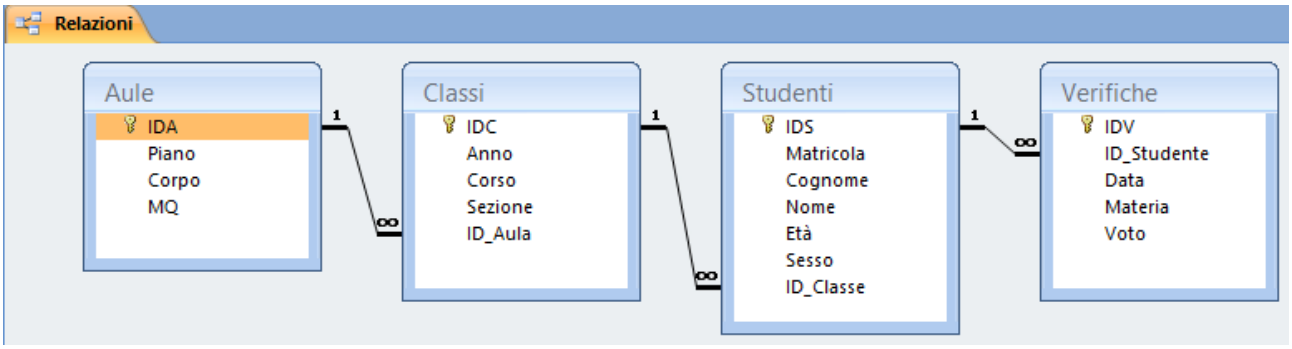
CONCETTO DI ANNIDAMENTO

Per annidamento si intende la possibilità che, all'interno di un comando, vi sia posto un ulteriore comando analogo. Per chi avesse studiato la programmazione imperativa, un esempio di annidamento è costituito da un ciclo WHILE dentro il quale è posto un ulteriore ciclo WHILE …

In SQL è possibile scrivere una SELECT dentro un'altra SELECT, purché nei punti previsti dalla sintassi. In teoria è possibile porre delle QUERY nelle tre clausole SELECT, FROM e WHERE. In effetti non tutti gli ambienti SQL ammettono di inerirle nelle due prime clausole e quindi eviteremo di trattarle …

DB DI RIFERIMENTO

Per procedere con le Query supponiamo uno schema relazionale che in Access potrebbe apparire come segue:



Supponiamo per esempio che la tabella Studenti contenga i seguenti dati:

Aule

IDA	Piano	Corpo	MQ
1	1	A	25
2	1	A	36
3	1	B	25
4	2	B	36
5	2	A	25
6	2	A	49

Classi

IDC	Anno	Sezione	Corso	ID_Aula
1	5	A	INF	4
2	4	A	INF	3
3	5	B	INF	
4	5	A	TEL	2

Studenti

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3
8	12352	Piano	Guida	18	F	1

Verifiche

IDV	ID_Studente	Data	Materia	Voto
1	3	01/04/2004	Storia	7
2	3	04/04/2004	Matematica	6
3	3	14/04/2004	Inglese	
4	6	01/04/2004		8
5	6	04/04/2004	Storia	3
6	2	01/04/2004	Matematica	7

ANNIDAMENTO NELLA CLAUSOLA SELECT

L'annidamento prevede che oltre a mostrare campi o funzioni nella clausola SELECT si possa anche inserire una query che calcola dati. La query interna deve restituire una sola colonna ed una sola riga (un solo valore) ogni volta che viene eseguita (per esempio con una funzione di aggregazione).

QUERY 70. - ANNIDAMENTO NELLA CLAUSOLA SELECT

```
SELECT * , (      SELECT COUNT(*) AS Inutile
                  FROM Verifiche AS V
                  WHERE S.IDS = V.ID_Studente) AS NumeroVerifiche
FROM Studenti AS S;
```

COMMENTO

Per ogni Studente (ogni riga della tabella Studenti) si esegue la query interna che calcola quante sono le verifiche affrontate per lo specifico studente). In pratica questa query mostra quante verifiche siano state svolte da ciascun studente.

Query70

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe	NumeroVerifiche
1	12345	Sale	Rosa	17	F	1	0
2	12346	Carta	Bianca	18	F	3	1
3	12348	Marino	Pino	19	F	1	3
4	12347	Piu	Mino	17	M		0
5	12349	Marino	Pino	18	M	1	0
6	12351	Carta	Rosa	17	F	3	2
7	12350	Piano	Remo	16	M	3	0
8	12352	Piano	Guida	18	F	1	0

ANNIDAMENTO NELLA CLAUSOLA FROM

L'annidamento suppone di poter eseguire una query a partire da una tabella dinamica invece che da una struttura.

La query interna deve essere calcolata prima di eseguire quella esterna.

QUERY 71. - ANNIDAMENTO NELLA CLAUSOLA FROM

```
SELECT MAX (V.NumeroVerifiche) AS MaxNumeroVerifiche
FROM (      SELECT      ID_Studente , COUNT(*) AS NumeroVerifiche
            FROM        Verifiche
            GROUP BY    ID_Studente) AS V;
```

Query71

MaxNumeroVerifiche
3

COMMENTO

Prima è eseguita la query interna che restituisce il numero di verifiche svolte per ogni studente. Questa tabella ha due colonne (ID_Studente e NumeroVerifiche). Su questa query si esegue quella esterna che calcola quale sia il valore massimo del campo NumeroVerifiche. In sintesi restituisce il numero delle verifiche dello studente più verificato.

ANNIDAMENTO NELLA CLAUSOLA WHERE

I precedenti tipi di annidamento non sono sempre ammessi dai database esistenti. Invece in generale sono ammesse le query annidate nella clausola WHERE. Nella clausola WHERE è possibile usare operatori di confronto, l'operatore **IN** e l'operatore **EXISTS**.

Prima di analizzare questo tipo di query è opportuno fare una considerazione sulle query nidificate. Quando una query è interna ad un'altra sono possibili **due diverse eventualità**:

- che la query interna sia **indipendente** da quella esterna: in questo caso la query interna può essere eseguita per prima mentre quella esterna si elabora in seguito sui risultati della prima. In questo caso si parla di **binding interno**.
- che la query interna sia **legata** a quella esterna e quindi non sia possibile eseguirla per prima. In questo caso per ogni elemento (riga) della query esterna si analizza il comportamento di quella interna e si valuta il risultato. In questo caso si parla di **binding esterno**.

ANNIDAMENTO E OPERATORI DI CONFRONTO

Un primo modo di usare le query nidificate è di confrontare il risultato della query interna con un operatore di confronto (= , <> , > , < , >= , <=) rispetto ad un altro valore.

Per queste query si deve fare attenzione se quelle interne sono indipendenti da quelle esterne. Vediamo alcuni esempi.

= , <> , > , < , >= , <=

QUERY 72. - ANNIDAMENTO E OPERATORI DI CONFRONTO

```
SELECT *
FROM   Studenti AS S1
WHERE  Età = (SELECT MAX(Età)
              FROM Studenti AS S2 );
```

Query25

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
5	12349	Marino	Pino	18	M	1

COMMENTO

La query interna è indipendente da quella esterna e viene eseguita per prima. La query interna restituisce la massima età tra gli studenti. A questo punto è possibile eseguire la query esterna che confronta l'età di ciascuno studente con il valore ottenuto in precedenza. Se coincidono la query esterna lo restituisce.

In sintesi la query mostra tutti i dati dello studente più anziano.

QUERY 73. - ANNIDAMENTO E OPERATORI DI CONFRONTO

```
SELECT *
FROM   Studenti AS S1
WHERE (Età = (SELECT MAX(Età)
                  FROM Studenti AS S2
                  WHERE S2.Sesso = 'M'))
AND   S1.Sesso = 'M';
```

Query25

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
5	12349	Marino	Pino	18	M	1

COMMENTO

La query interna è indipendente da quella esterna e viene eseguita per prima. La query interna restituisce la massima età tra gli studenti maschi. A questo punto è possibile eseguire la query esterna che confronta l'età di ciascun studente maschio con il valore ottenuto in precedenza. Se coincidono la query esterna lo restituisce.

In sintesi la query mostra tutti i dati dello studente maschio più anziano.

QUERY 74. - ANNIDAMENTO E OPERATORI DI CONFRONTO

Vediamo adesso un caso in cui la query interna sia legata a quella esterna e quindi non sia possibile eseguirla per prima. In questo caso si parla di binding esterno e per ogni elemento (riga) della query esterna si analizza il comportamento di quella interna e si valuta il risultato.

```
SELECT *
FROM   Studenti AS S
WHERE 2 < (SELECT COUNT(*)
                  FROM Verifiche AS V
                  WHERE V.ID_Studente = S.IDS));
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
3	12348	Marino	Pino	19	F	1

COMMENTO

La query interna e quella esterna sono legate! Quindi si analizza ogni singolo record della query esterna; preso un singolo record (uno studente) si elabora la query interna e si conteggia il numero di verifiche sostenute dallo studente; se il numero di verifiche è maggiore di 2 la query esterna rende lo studente.

In sintesi la query mostra tutti i dati degli studenti che hanno più di 2 verifiche (almeno 3).

IN (E NOT IN)

L'operatore **IN** verifica se un dato valore è presente in un elenco.

Se è presente rende **Vero** altrimenti rende **Falso**.

La forma sintattica generale somiglia alla seguente:

```
SELECT  elenco_campi
FROM    Tabelle
WHERE   Campo IN (Query_interna);
```

Vediamo alcuni esempi:

QUERY 75. - ANNIDAMENTO E OPERATORE IN

```
SELECT *
FROM Studenti
WHERE IDS IN (    SELECT ID_Studente
                  FROM Verifiche );
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
6	12351	Carta	Rosa	17	F	3

COMMENTO

La query interna e quella esterna sono indipendenti. La query interna rende l'elenco delle chiavi di coloro che hanno sostenuto verifiche (ovvero sono elencati nelle chiavi esterne). La query esterna esamina tutti gli studenti e filtra solo quelli la cui chiave primaria compare in elenco.

In sintesi la query mostra tutti i dati degli studenti che hanno almeno una verifica.

QUERY 76. - ANNIDAMENTO E OPERATORE IN

```
SELECT *
FROM Studenti
WHERE IDS NOT IN (      SELECT ID_Studente
                        FROM Verifiche );
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
7	12350	Piano	Remo	16	M	3
8	12352	Piano	Guida	18	F	1

COMMENTO

La query interna e quella esterna sono indipendenti. La query interna rende l'elenco delle chiavi di coloro che hanno sostenuto verifiche (ovvero sono elencati nelle chiavi esterne). La query esterna esamina tutti gli studenti e esclude quelli la cui chiave primaria compare in elenco.

In sintesi la query mostra tutti i dati degli studenti che **non** hanno alcuna verifica. La query è complementare alla precedente.

OPERATORE EXISTS

L'operatore EXISTS verifica se una query interna è vuota oppure restituisca dei dati. Se ci sono dei dati allora rende Vero altrimenti (è vuota) rende Falso.

La forma sintattica generale somiglia alla seguente:

```
SELECT  elenco_campi
FROM    Tabelle
WHERE   EXISTS (Query_interna);
```

La forma complementare somiglia alla seguente:

```
SELECT  elenco_campi
FROM    Tabelle
WHERE   NOT EXISTS (Query_interna);
```

Vediamo alcuni esempi:

QUERY 77. - ANNIDAMENTO E OPERATORE EXISTS (QUERY INGENUA)

```
SELECT *
FROM Studenti
WHERE EXISTS (    SELECT ID_Studente
                  FROM Verifiche );
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
1	12345	Sale	Rosa	17	F	1
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
4	12347	Piu	Mino	17	M	
5	12349	Marino	Pino	18	M	1
6	12351	Carta	Rosa	17	F	3
7	12350	Piano	Remo	16	M	3
8	12352	Piano	Guida	18	F	1

COMMENTO

La query interna e quella esterna sono indipendenti. La query interna rende qualcosa. La query esterna rende true per ciascuno degli studenti.

In sintesi la query mostra tutti i dati di tutti gli studenti; il filtro è inutile.

QUERY 78. - ANNIDAMENTO E OPERATORE EXISTS

```
SELECT *
FROM Studenti AS S
WHERE EXISTS (    SELECT *
                  FROM Verifiche AS V
                  WHERE S.IDS = V.ID_Studente );
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
6	12351	Carta	Rosa	17	F	3

COMMENTO

La query interna e quella esterna sono legate. Per ogni studenti si verifica se la query interna rende qualcosa. Per ogni studente si verifica se esiste almeno una verifica di quello studente.

In sintesi la query mostra tutti i dati di tutti gli studenti con almeno una verifica.

QUERY 79. - ANNIDAMENTO E OPERATORE EXISTS

```
SELECT *
FROM Studenti AS S
WHERE NOT EXISTS (SELECT *
                  FROM Verifiche AS V
                  WHERE S.IDS = V.ID_Studente );
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
6	12351	Carta	Rosa	17	F	3

COMMENTO

La query interna e quella esterna sono legate. Per ogni studenti si verifica se la query interna rende qualcosa. Per ogni studente si verifica se esiste almeno una verifica di quello studente. Se esiste lo si trascura.

In sintesi la query mostra tutti i dati di tutti gli studenti senza verifiche.

QUERY 80. - ANNIDAMENTO E OPERATORE EXISTS

```
SELECT *
FROM Verifiche AS VX
WHERE NOT EXISTS (      SELECT *
                        FROM   Verifiche AS VY
                        WHERE  VX.Materia = VY.Materia
                        AND    VX.Voto > VY.Voto );
```

Query

IDV	ID_Studente	Data	Materia	Voto
2	3	04/04/2004	Matematica	6
3	3	14/04/2004	Inglese	
4	6	01/04/2004		8
5	6	04/04/2004	Storia	3

Mostra quelle verifiche col voto migliore in ogni materia.

COMMENTO

La query interna e quella esterna sono legate. Per ogni verifica si controlla se non esiste una altra verifica con stessa materia ma voto superiore. Se non esiste allora rende tutti i dati della verifica.

In sintesi la query mostra quelle verifiche col voto migliore in ogni materia.

ANNIDARE IN PIÙ LIVELLI

Se necessario è possibile nidificare le query in più livelli, ovvero inserire in una query interna un'altra query ancora più interna.

QUERY 81. - ANNIDAMENTO E OPERATORE IN

```
SELECT *
FROM Studenti AS S1
WHERE IDS IN (SELECT ID_Studente
              FROM Verifiche
              WHERE Materia IN (    SELECT Materia
                                    FROM Verifiche
                                    WHERE ID_Studente <> S1.IDS )       );
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
2	12346	Carta	Bianca	18	F	3
3	12348	Marino	Pino	19	F	1
6	12351	Carta	Rosa	17	F	3

COMMENTO

La query più interna e quella più esterna sono legate. Per ogni studente si controlla che sia tra quelli con una verifica la cui materia sia presente tra le materie delle verifiche di uno studente diverso. Quindi cerca gli studenti con verifiche in materie per le quali anche altri abbiano verifiche.

In sintesi la query mostra tutti gli studenti che hanno verifiche in materie per le quali anche altri hanno verifiche.

QUERY 82. - ANNIDAMENTO E OPERATORE EXISTS

```
SELECT *
FROM Studenti AS SX
WHERE NOT EXISTS( SELECT *
                  FROM Verifiche AS V1
                  WHERE NOT EXISTS( SELECT *
                                    FROM  Verifiche AS V2
                                    WHERE V2.Materia = V1.Materia
                                    AND   V2.ID_Studente <> SX.IDS )  );
```

Query

IDS	Matricola	Cognome	Nome	Età	Sesso	ID_Classe
* (Nuovo)						

COMMENTO

Non ho idea di cosa faccia questa Query … Chi mi può aiutare?

ESERCIZI

Si consideri il seguente schema relazionale:

```
IMPIEGATI (IDI, Cognome, Nome, Età, Sesso);
      PK IDI;
PROGETTI (IDP, Nome, Budget, Direttore);
      PK IDP;
      FK Direttore REF Impiegati (IDI);
SISTEMI (IDS, Nome, Descrizione);
      PK IDS;
ATTIVITÀ (IDA, Data, Progetto, Sistema);
      PK IDA;
      FK Progetto REF Progetti (IDP);
      FK Sistema REF Sistemi (IDI);
```

ESERCIZI BASE

Esercizio 1.

Elencare in ordine alfabetico tutti gli impiegati di età superiore a 50.

Esercizio 2.

Elencare in ordine alfabetico tutti gli impiegati di età compresa tra 25 e 50.

Esercizio 3.

Elencare in ordine alfabetico tutti gli impiegati col nome identico al proprio cognome.

Esercizio 4.

Elencare in ordine alfabetico tutti gli impiegati che sono direttori di progetto.

Esercizio 5.

Elencare tutti i sistemi in ordine alfabetico.

ESERCIZI INTERMEDI

Esercizio 6.

Mostrare l'età media degli impiegati.

Esercizio 7.

Mostrare l'età media dei direttori.

Esercizio 8.

Mostrare l'età dell'impiegato più anziano e quella del più giovane

Esercizio 9.

Contare quanti impiegati hanno meno di 30 anni.

Esercizio 10.

Contare quanti progetti hanno un budget inferiore a 50000 euro.

Esercizio 11.

Contare quanti progetti non hanno un direttore.

ESERCIZI SU GIUNZIONI ESTERNE

Esercizio 12.

Elencare tutti i progetti e, dove possibile, i rispettivi direttori.

Esercizio 13.

Elencare tutti i sistemi e, dove possibile, le attività in cui sono stati utilizzati.

Esercizio 14.

Elencare tutti i sistemi e, dove possibile, i progetti in cui sono stati utilizzati.

ESERCIZI SU ANNIDAMENTI

Esercizio 15.

Elencare quanti impiegati hanno un'età inferiore alla media.

Esercizio 16.

Elencare quanti progetti hanno un budget superiore alla media.

Esercizio 17.

Elencare quante impiegate hanno un'età inferiore alla media (delle sole donne).

Esercizio 18.

Elencare quanti direttori hanno un'età inferiore alla media.

ESERCIZI SU RAGGRUPPAMENTI

Esercizio 19.

Per ogni impiegato calcolare quanti progetti dirige.

Esercizio 20.

Per ogni sistema calcolare quante volte è stato usato in progetti.

Esercizio 21.

Per ogni sistema calcolare in quanti progetti è stato usato.

Esercizio 22.

Elencare ogni sistema usato più di 2 volte in qualche progetto.

ESERCIZI VARI

Esercizio 23.

Contare quanti impiegati non dirigono progetti.

Esercizio 24.

Contare quanti sistemi non mai stati impiegati in progetti.

Esercizio 25.

Contare quanti sistemi sono stati impiegati esattamente una sola volta in progetti.

Esercizio 26.

Elencare quali sistemi non sono mai stati usati in progetti con budget di almeno 50000 euro.