MANUALE SQL

Guida di livello base per gli utenti che vogliono approcciare con il linguaggio SQL per creare ricerche avanzate sull'archivio di Millewin

Ultima revisione Settembre 2017

Sommario

DATI PRODUTTORE

Via Di Collodi, 6/C – 50141 Firenze

COME CONTATTARE MILLENNIUM

Per posta elettronica:

- Assistenza Tecnica: assistenza.millennium@dedalus.eu
- Ufficio Commerciale: commerciale.millennium@dedalus.eu

Per telefono:

- Centralino: 055 45544.1
- Fax: 055 4554.420
- 800 949502 da rete fissa
- 199 110077 da rete mobile

Database e SQL

Database in informatica, è una raccolta di dati codificati, preparata per l'archiviazione dell'informazione nella memoria di massa di un computer e la loro successiva eventuale elaborazione, organizzata in modo da consentire facile accesso agli utenti autorizzati.

Considerando un database come un insieme di dati che si riferiscono ad una specifica esigenza, il database di un rivenditore di automobili, per esempio, conterrà i dati delle automobili, dei pezzi di ricambio, dei clienti e tutto quanto serve per condurre il suo lavoro in modo efficiente, facile e veloce.
Così il database del Medico di Medicina Generale contiene tutti i dati relativi ai pazienti, alle visite, ai farmaci, alle regole burocratiche ecc..

Nell'ambito del database, i dati sono raggruppati per tabelle.

Le tabelle sono sottoinsiemi ordinati di dati omogenei e possono essere correlate tra di loro se hanno una colonna con un tipo di informazione comune.
Avremo quindi la tabella anagrafica, quella delle esenzioni, quella degli esami di laboratorio ecc..

Si chiamano tabelle perché possono essere esaminate secondo una disposizione tabellare. Ecco per esempio un frammento della tabella Anagrafica di un ipotetico programma di gestione dell'ambulatorio del medico di MG:

Cognome	Nome	Indirizzo
Rossi	Alceste	Via Antrace, 5
Verdi	Duilio	Via Stafilo, 76
Gialli	Gaetano	Via Coliforme, 65
Bianchi	Giulio	Via Treponema, 6

Ogni riga della tabella si denomina record. Così i dati anagrafici relativi a Rossi Alceste sono un record e così via.

Ogni colonna di una tabella si denomina campo. In questo caso abbiamo il campo: Nome, Cognome e Indirizzo.

Il programma Millewin per la gestione dei dati dell'ambulatorio del Medico di Medicina Generale è composto di due blocchi funzionali distinti: un motore database (Postgres) che gestisce tutte le operazioni relative ai dati e un blocco di interfaccia tramite il quale diciamo al motore database le operazioni che desideriamo vengano effettuate.

Un po' come un'automobile: abbiamo il motore che spinge e riceve le istruzioni dall'acceleratore, dalla frizione e dal freno.

I moderni programmi di database usano motori database prodotti da ditte specializzate. Nessun programmatore svilupperebbe oggi un motore database nell'ambito di un suo programma. Occorrerebbe molto lavoro, molta perizia e ne verrebbero vendute poche copie legate al programma che ha intenzione di sviluppare. Meglio usare un motore database in commercio, testato, veloce e affidabile e creare una interfaccia funzionale.

Il colloquio tra l'interfaccia e il motore database avviene nei casi più nobili, tramite un linguaggio specializzato, SQL, il linguaggio dei database per eccellenza.

SQL significa Structured Query Language (Linguaggio Strutturato per Domande, traduzione infelice che indica un linguaggio strutturato per fare delle domande al database).

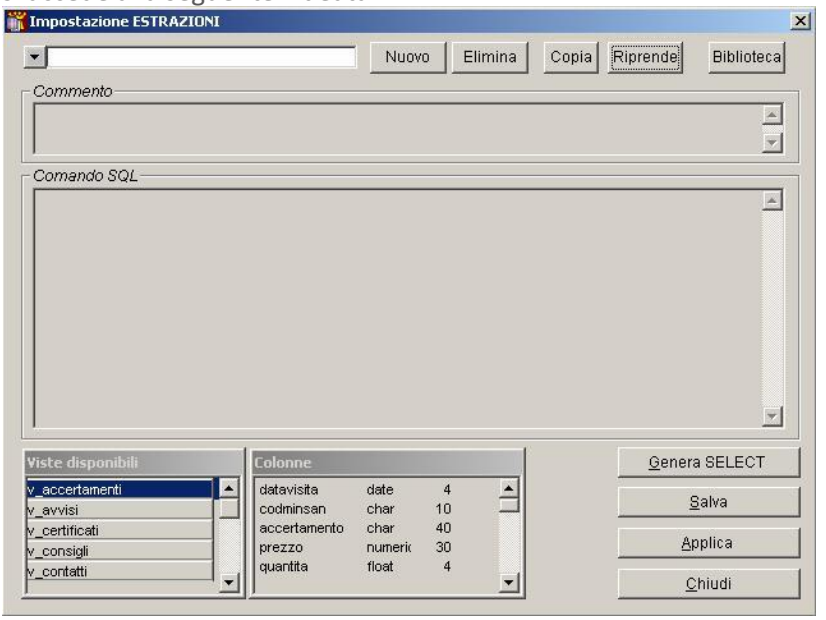
Il linguaggio SQL nacque negli anni 70 alla IBM, venendo incontro alla esigenza di fornire un linguaggio per l'interrogazione dei database che fosse indipendente dai linguaggi di programmazione.

La sigla SQL si pronuncia ess-que-ell (in inglese) o EsseQuElle e non se-qu-el. Questa confusione nasce dal fatto che il linguaggio SQL è il successore di un linguaggio dello stesso tipo sviluppato alla IBM che si chiamava per l'appunto Sequel.

I programmi più moderni usano un database che accetta istruzioni SQL. Però l'utente del programma non vede le istruzioni, ma ne vede solo i risultati.

Così, quando cerchiamo un paziente, nella schermata iniziale del programma, scrivendo le prime lettere del suo nome, parte una istruzione SQL che noi non vediamo, la quale arriva al motore database, che restituisce un insieme di record congruenti con la nostra richiesta.

Mille Utilità è dotato di una finestra in cui è possibile interrogare il database mediante istruzioni SQL per elaborare ricerche personali.
Da Mille Utilità ⇒ Statistiche ⇒ Impostazioni Estrazioni SQL Personali si accede alla seguente videata:



Nella finestra in basso a sinistra, Viste disponibili, sono riportati i nomi delle Tabelle disponibili. Nella finestra al centro, *Colonne*, sono riportati i campi che compongono ogni tabella; il tipo di campo, cioè se sono dati tipo carattere, numero o stringa e la loro lunghezza.

In Appendice riportiamo il titolo e la composizione di ogni singola tabella, ossia gli schemi di tutte le viste disponibili.

Vengono denominate Viste e non Tabelle, perché una tabella è un insieme di dati memorizzato stabilmente nel database, anche a computer spento. Le tabelle di Millewin sono incise nella memoria fissa del computer con microparticelle solidamente magnetizzate e solo una catastrofe informatica può distruggerle. Noi non accediamo a quelle tabelle ma ad una rielaborazione di esse che viene generata quando attiviamo il motore di ricerca SQL. In pratica le viste sono tabelle generate estemporaneamente e che vivono nella memoria volatile del computer.

Comandi SQL

Ad un qualsiasi database si possono chiedere informazioni che si desiderano selezionare (SELECT), inserire (INSERT), aggiornare (UPDATE) o cancellare (DELETE). Sono i quattro verbi fondamentali che si utilizzano. I termini o comandi SQL sono: SELECT – FROM – WHERE – ORDER BY – HAVING – GROUP BY. Tali comandi presentano modalità specifiche nella strutturazione delle parole che seguono. I gruppi di parole che comprendono queste parole chiave sono spesso definite clausole.

CLAUSOLA SELECT ⇒ SELECT cognome, nome

CLAUSOLA FROM ⇒ FROM v_pazienti

CLAUSOLA WHERE ⇒ WHERE cognome = ‘Rossi’

CLAUSOLA ORDER BY ⇒ ORDER BY nome

SELECT_FROM

La clausola principale del linguaggio SQL è SELECT_FROM, che significa SELEZIONA_DA. Con essa possiamo infatti selezionare i record che ci interessano da una o più tabelle.

Quindi SELECT è la parola chiave che indica le colonne che si desiderano visualizzare, mentre FROM indica i nomi delle tabelle in cui si trovano tali colonne.

La sintassi è la seguente:

SELECT campo1, campo2,… FROM tabella

Da Mille Utilità ⇒ Statistiche ⇒ Impostazioni Estrazioni SQL Personali procedere come segue:

- Fare clic sul tasto *Nuovo* e dare un nome query SQL
- Scegliete la tabella *v_pazienti* nella finestra in basso a sinistra
- Fare clic sul tasto *Genera SELECT*

A video si avrà, automaticamente, *SELECT cognome, nome, datanasc, …, stato_civ FROM v_pazienti*, con la quale otterrete la successione di tutti i record contenuti nella tabella, ciascuno con tutti i campi.

Fare clic su *Applica* e scegliete il periodo temporale che interessa.

È possibile, inoltre, filtrare e ordinare in vario modo l'elenco facendo clic su *Filtra* (corrisponde alla clausola WHERE descritta di seguito) o *Ordina* (corrisponde alla clausola ORDER BY descritta di seguito) e seguendo le istruzioni che compaiono.

Invece di digitare tutti i nomi dei campi potete semplicemente digitare il simbolo asterisco, in questo modo:

SELECT * FROM v_pazienti

ed otterrete lo stesso risultato.

Oppure, ottenuta la formula di prima potete cancellare i campi che non vi interessano per ottenere per esempio:

SELECT cognome, nome FROM v_pazienti

ed ottenere cognome e nome di tutti i vostri pazienti.

ORDER BY

Il comando ORDER BY (ORDINA SECONDO) si utilizza se si desidera che le informazioni selezionate siano visualizzate in un determinato ordine. Ad ORDER BY seguirà il campo che vogliamo regoli l'ordinamento.

Per esempio:

```
SELECT cognome, nome
FROM v_pazienti

ORDER BY cognome
```

si ottiene l'elenco alfabetico, per cognome, dei pazienti.

Possiamo usare anche due o più campi per ordinare i dati:

```
SELECT cognome, nome
FROM v_pazienti
ORDER BY cognome, nome
```

E potremmo anche usare campi non inclusi nella clausola SELECT, ad esempio:

```
SELECT cognome, nome
FROM v_pazienti
ORDER BY cognome, nome, comunasc
```

Quando non specificato il programma effettua l'ordinamento in senso ascendente (ASC), pertanto:

```
SELECT cognome, nome
FROM v_pazienti
ORDER BY cognome
```

equivale a:

```
SELECT cognome, nome
FROM v_pazienti
ORDER BY cognome ASC
```

Mentre se volete i dati ordinati in maniera discendente (DESC) dovrete scrivere:

```
SELECT cognome, nome
FROM v_pazienti
ORDER BY cognome DESC
```

DESC è la parola chiave che sta per descending (ordine decrescente).

È possibile utilizzare più campi diversi con ordinamenti diversi:

```
SELECT cognome, nome
FROM v_pazienti

ORDER BY cognome ASC, nome DESC
```

WHERE

La clausola WHERE serve per:

- selezionare i record secondo criteri specificati
- unire due o più tabelle in una sola tabella risultante

Selezione di record secondo criteri specificati

Tramite la clausola WHERE possiamo comparare i dati di ogni colonna con valori da noi definiti.
La sintassi è la seguente:

WHERE campo = valore

La clausola si pone dopo il FROM

Il segno di uguale (=) viene definito come operatore logico, perché lo si utilizza effettuando una verifica logica, ovvero confrontando valori posti alla sua sinistra e alla sua destra.

Vogliamo, per esempio, avere la lista dei pazienti il cui cognome è ROSSI:

```
SELECT cognome, nome
FROM v_pazienti
WHERE cognome = 'ROSSI'
```

Il parametro di paragone è scritto tra singoli apici e senza spazi tra il primo apice e la lettera successiva o tra la fine della parola e l'apice finale.

Operatori logici

La clausola con WHERE può essere composta da più parti, pertanto possiamo porre più di una clausola tramite gli *operatori logici* AND e OR:

```
SELECT cognome, nome
FROM v_pazienti
WHERE cognome = 'ROSSI' OR cognome = 'VERDI'
```

Ottenendo, così, l'elenco di tutti i pazienti il cui cognome è ROSSI o VERDI.

L' operatore AND è prevalente, perché collega le espressioni logiche alla sua sinistra e alla sua destra in maniera più forte di quanto accada con OR (tecnicamente si dice che ha precedenza superiore). Le parentesi che racchiudono le espressioni da interpretare insieme prevalgono sull'ordine di precedenza normale.

```
SELECT cognome, nome
FROM v_pazienti
WHERE cognome = 'ROSSI' AND nome <> 'GIOVANNI'
```

fornirà l'elenco dei pazienti il cui cognome è ROSSI ma con nome diverso da GIOVANNI (notare il simbolo <> per indicare diverso).

I simboli matematici detti operatori logici, in quanto è possibile effettuare dei test logici su un singolo valore, sono:

OPERATORE LOGICO	SIGNIFICATO
=	UGUALE a …
<>	DIVERSO da …
>	MAGGIORE di …
> =	MAGGIORE O UGUALE di…
<	MINORE di …
< =	MINORE O UGUALE di …

Altri operatori importanti sono BETWEEN_AND, IN, e LIKE.
Tali operatori consentono di effettuare test logici su un elenco di valori.

BETWEEN significa TRA e serve per definire un intervallo di valori in cui vogliamo siano selezionati i nostri record:

```
SELECT cognome, nome
FROM v_pazienti
WHERE datanasc BETWEEN '1954/01/01' AND '1964/01/01'
```

si ottiene l'elenco dei pazienti nati tra le due date, estremi compresi.

IN consente di effettuare test logici su diversi valori, cioè è necessario a fornire un elenco di termini di paragone:

```
SELECT cognome, nome
FROM v_pazienti
WHERE cognome IN ('ROSSI','BIANCHI','GIALLI','VERDI')
```

fornirà l'elenco dei pazienti con i nomi selezionati.

Infine la clausola LIKE (SIMILE), molto potente e versatile, può essere utilizzata per trovare i record il cui campo indicato è uguale a quello dato:

```
SELECT cognome, nome
FROM v_pazienti
WHERE cognome LIKE 'ROSSI'
```

selezionerà tutti i pazienti il cui cognome è ROSSI.

Questo è un utilizzo un po' banale di LIKE, che dà il meglio di sé in associazione al simbolo % che indica una qualsiasi stringa. Pertanto con l'operatore LIKE è possibile effettuare una ricerca nelle righe di

una colonna di un database per valori che assomigliano ad un dato modello descritto.

Così, per esempio:

```
SELECT cognome, nome
FROM v_pazienti
WHERE cognome LIKE 'R%'
ORDER BY cognome, nome
```

Selezionerà tutti i record il cui cognome comincia con la lettera R seguita da qualsiasi successione di caratteri (Rossi, Rinaldi, Rimondi etc.) ordinando i cognomi in ordine alfabetico.

Ed inoltre

```
SELECT cognome, nome
FROM v_pazienti
WHERE nome LIKE '%io%'
ORDER BY cognome, nome
```

Selezionerà tutti i record nel cui campo nome vi sia la stringa “io” preceduta e seguita da qualsiasi insieme di caratteri (Giovanni, Niobe, etc.)

```
SELECT cognome, nome
FROM v_pazienti
WHERE nome LIKE '%o%o%'
ORDER BY cognome, nome
```

I segni di percentuale multipli servono a selezionare tutti i record nel cui campo nome vi siano presenti due “o” in qualsiasi posizione.

```
SELECT cognome, nome
FROM v_pazienti
WHERE nome LIKE '_io%'
ORDER BY cognome, nome
```

Selezionerà tutti i record nel cui campo nome vi sia la stringa “io” preceduta da un solo (e qualsiasi) carattere e seguita da qualsiasi insieme di caratteri (Giovanni, Niobe, etc.)

```
SELECT cognome, nome
FROM v_pazienti
WHERE nome LIKE '_ _io%'
ORDER BY cognome, nome
```

Selezionerà tutti i record nel cui campo nome vi sia la stringa “io” preceduta da due qualsiasi caratteri (i due trattini indicano che qualsiasi carattere precedente è accettabile) e seguita da qualsiasi insieme di caratteri.

Riprenderemo più avanti la funzione LIKE quando parleremo delle funzioni di aggregazione.

Possiamo anche introdurre clausole comprendenti date:

La funzione days ('aaaa/mm/gg','aaaa/mm/gg') ci restituisce la differenza in giorni tra la seconda data e la prima

Possiamo sfruttarla per selezionare le nostre pazienti con età maggiore di 50 anni che devono effettuare la mammografia:

```
SELECT cognome, nome, (days(datanasc, today())/365) AS Eta FROM
v_pazienti

WHERE (days(datanasc, today())/365) >49 AND sesso ='F' ORDER BY
Eta Desc
```

Il report è ordinato in senso discendente

TODAY() è la funzione che riporta la data di oggi.

Si notino due importanti cose in questa query:

- posso usare come contenuto di una colonna un campo calcolato, in questo caso l'età;

- la parola chiave AS che segue il nome o la formula per riempire il campo in questo modo:
- SELECT codfisc AS Codice Fiscale,….

Permette di ridenominare l'intestazione della colonna in modo più leggibile. Per default il computer mette il nome del campo che molte volte è criptico per esigenze di programmazione.
Potremmo anche non mettere la clausola AS e far seguire direttamente il nome che intendiamo attribuire alla colonna:

SELECT cognome, nome, (days(datanasc, today())/365) Eta

Selezioniamo i pazienti con glicemie superiori a 110 mg/dl:

SELECT cognome, nome, risults
FROM v_accertamenti
WHERE accertamento LIKE 'glice%' AND risults > 110 ORDER BY cognome, nome

Impiego di WHERE per collegare due o più tabelle in una tabella finale

Possiamo creare una tabella unendo record di due tabelle distinte, le quali però abbiano un campo in comune.

Per esempio, vogliamo fare uno stampato riassuntivo di tutte le prescrizioni eseguite in un certo periodo riportando il nome del farmaco ed il nominativo del paziente a cui è stato prescritto.

Occorrono dati dalla tabella v_pazienti e dalla tabella v_terapie. Le tabelle sono collegate tra loro dal campo codice che è unico per ogni paziente:

```
SELECT v_pazienti.cognome, v_pazienti.nome, v_terapie.terapia,
v_terapie.quantita
FROM v_pazienti, v_terapie
WHERE v_pazienti.codice = v_terapie.codice
ORDER BY v_pazienti.cognome, v_pazienti.nome
```

Notate come bisogna specificare la tabella di appartenenza di un campo facendo precedere il nome del campo dal nome della tabella uniti da un punto.

In questo caso il programma cercherà: per ogni Cognome e Nome nel database, a cui corrisponde un unico codice, le prescrizioni relative nella tabella terapie a cui corrisponde il medesimo codice. Il tutto ordinato per cognome e nome.

Selezioniamo tutte le glicemie anormali delle pazienti donne che abitano in via del Campo e hanno più di 40 anni:

```
SELECT v_pazienti.cognome, v_pazienti.nome,
v_accertamenti.risultn
FROM v_accertamenti, v_pazienti
WHERE v_pazienti.codice = v_accertamenti.codice AND
v_pazienti.indirizzo LIKE '%del Campo%' AND
(days(v_pazienti.datanasc, today())/365) >39 AND v_pazienti.sesso
='F' AND v_accertamenti.accertamento = 'glicemia' AND
v_accertamenti.risultn > 110
ORDER BY v_pazienti. cognome, v_pazienti.nome
```

In questo caso troviamo i dati relativi all'indirizzo nella tabella v_pazienti e i dati relativi agli esami richiesti nella tabella accertamenti, correlate tra loro con la clausola WHERE.

DISTINCT E DISTINCTROW

Supponiamo di voler ottenere un elenco degli assistiti che ha nel database almeno una misurazione della pressione.

Con la normale clausola SELECT FROM otterremo quanto richiesto ma nel caso un paziente abbia più di una misurazione della pressione avremo diversi record con lo stesso cognome e nome.

La clausola DISTINCT dice al motore database di prendere solo il primo record nel caso vi siano più record con gli stessi dati:

```
SELECT DISTINCT cognome, nome
FROM V_PRESSIONE
```

Non si può aggiungere i campi massima e minima perché in questo caso record con lo stesso nome e cognome potrebbero differire per i valori pressori e quindi vi sarebbero ancora dei duplicati.

Vi sono casi in cui si vogliono selezionare record che abbiano almeno un campo differente su tutti i campi.

Potremmo usare SELECT DISTINCT e la lista di tutti i campi.

FUNZIONI DI AGGREGAZIONE

Possiamo tramite le funzioni di aggregazione COUNT (CONTA) AVG (MEDIA), SUM (SOMMA), MIN (MINIMO), MAX (MASSIMO) compiere operazioni su vasti gruppi di record numerici.

La clausola COUNT conta tutti i record che ci interessano. Le altre clausole eseguono le operazioni sui campi di questi record selezionati.

Vogliamo sapere il numero di glicemie che abbiamo ordinato, la media dei valori, il valore minimo e il valore massimo:

```
SELECT COUNT( risultn) AS Misurazioni, AVG(risultn) AS Media,
MIN(risultn) As Minimo, MAX(risultn) AS Massimo FROM
```

V_ACCERTAMENTI WHERE accertamento = 'glicemia' AND NOT risultn ='0'

L'ultima clausola NOT risultn … che specifica che il risultato delle glicemie deve essere diverso da 0 serva ad impedire che vengano contate anche le glicemie di cui attendiamo risposta o che abbiamo già siglato come normali.

Ed ora vogliamo sapere il totale delle scatole di Tareg 80 che abbiamo ordinato:

```
SELECT COUNT( terapia), SUM(quantita)
FROM V_TERAPIE
WHERE terapia LIKE 'tareg%'
```

Si utilizza la clausola LIKE perché non conosciamo esattamente la dizione da introdurre. Nel caso in cui si voglia effettuare la ricerca su un farmaco con lo stesso nome, ma due o più dosaggi differenti, dobbiamo inserire il nome esatto nella clausola.

GROUP BY

Questa è una clausola molto potente che consente di raggruppare gruppi di record secondo criteri a nostra scelta.

Per esempio vogliamo sapere la spesa mensile per i farmaci di fascia A:

```
SELECT terapia, SUM(quantita) quanti, SUM (prezzo*quantita) spesa
FROM v_terapie
WHERE fascia = 'A'
GROUP BY terapia
ORDER BY terapia
```

La query sopra riportata indica:

- SELECT terapia GROUP BY terapia: raggruppa i record che hanno lo stesso campo terapia (es. tutte le prescrizioni di “QUINAZIDE”)
- SUM(quantita) quanti: somma i campi quantità per i record con lo stesso campo terapia ed assegna il termine “Quanti” come intestazione di colonna
- SUM (prezzo*quantita) spesa: moltiplica la quantità per il prezzo di ogni record ed esegue il totale. Assegna il termine “Spesa” come intestazione di colonna

Se nel report sopra riportato si aggiunge alla clausola SELECT un altro campo (es. Fascia) allora occorre ricordarsi di includerlo nella clausola GROUP BY. In caso contrario il programma darà un messaggio di errore, dato che il campo aggiunto non è un campo che viene sommato.

JOIN

Vi sono 3 tipi di clausola JOIN:

INNER JOIN
LEFT OUTHER JOIN
RIGHT OUTHER JOIN

INNER JOIN

Consente di estrarre da una tabella tutti i record che hanno una esatta corrispondenza per uno o più campi in un’altra tabella.

Ammettiamo di voler estrarre dalla tabella anagrafica Nome, Cognome e Indirizzo di tutti i pazienti che hanno una misurazione della pressione:

SELECT DISTINCT v_pazienti.cognome, v_pazienti.nome,
v_pazienti.indirizzo

FROM v_pazienti INNER JOIN v_pressione ON v_pazienti.codice = v_pressione.codice

Selezioniamo tutti i pazienti a cui è stato prescritto il Cibadrex:

SELECT DISTINCT v_pazienti.cognome, v_pazienti.nome, v_pazienti.indirizzo FROM v_pazienti INNER JOIN v_terapie ON v_pazienti.codice = v_terapie.codice WHERE v_terapie.terapia like 'cibadrex%' ORDER BY v_pazienti.cognome, v_pazienti.nome

LEFT OUTER JOIN

LEFT OUTER JOIN si utilizza quando occorrono TUTTI i record della prima tabella (quella a sinistra/left) PIU' i record della seconda (quella di destra) che abbiano una corrispondenza in un campo con la prima tabella.

Supponiamo di aver bisogno dell'elenco di tutti i pazienti (cognome, nome ed indirizzo) con i nomi e i valori di tutti gli accertamenti eseguiti, per quei pazienti che li hanno eseguiti.

Avremo una tabella con TUTTI i pazienti e per ognuno di essi saranno riportati gli esami con i valori corrispondenti e nel caso dei pazienti che non hanno eseguito esami sarà riportato nome, cognome indirizzo e i rimanenti campi saranno vuoti:

SELECT v_pazienti.cognome, v_pazienti.nome, v_pazienti.indirizzo, v_accertamenti.accertamento, v_accertamenti.risultn
FROM v_pazienti LEFT OUTER JOIN v_accertamenti ON v_pazienti.codice = v_accertamenti.codice
ORDER BY v_pazienti.cognome, v_pazienti.nome

RIGHT OUTER JOIN

Si comporta al contrario della clausola precedente: si utilizza quando occorrono TUTTI i record della seconda tabella (quella a destra) e quelli della prima tabella che abbiano una corrispondenza in un campo con la seconda tabella.

Supponiamo di voler compilare una tabella in cui appare a destra il nome del consiglio che abbiamo eventualmente dato al paziente e a sinistra: nome, cognome e indirizzo del paziente e in più nome, cognome e indirizzo anche dei pazienti a cui non abbiamo mai dato alcun consiglio, il tutto ordinato per cognome e nome dei pazienti:

```
SELECT v_consigli.consiglio, v_pazienti.cognome, v_pazienti.nome,
v_pazienti.indirizzo
FROM v_consigli RIGHT OUTER JOIN v_pazienti ON v_consigli.codice
= v_pazienti.codice
ORDER BY v_pazienti.nome, v_pazienti.cognome
```

Infine parliamo di una altra possibilità di usare la clausola JOIN.

Se abbiamo una tabella “accertamenti” composta dai seguenti records:

Nome	Esame	Valore
Gigi	Colesterolo	200
Carlo	Trigliceridi	180
Gigi	Trigliceridi	175
Carlo	Colesterolo	187

e facciamo un JOIN della tabella su se stessa:

```
SELECT a.nome, a.risultn as Colesterolo, b.risultn as Trigliceridi
FROM v_accertamenti a LEFT OUTER JOIN v_accertamenti b ON
a.nome = b.nome
```

WHERE a.accertamento = 'Colesterolo' and b.accertamento = 'Trigliceridi'

otteniamo una tabella le cui righe sono le seguenti:

Nome	Colesterolo	Trigliceridi
Gigi	200	175
Carlo	187	180

Se poi facciamo un doppio JOIN su questa tabella:

Nome	Esame	Valore
Gigi	Colesterolo	200
Carlo	Trigliceridi	180
Gigi	Trigliceridi	175
Carlo	Colesterolo	187
Carlo	HDL	40
Gigi	HDL	70

SELECT a.nome, a.risultn as Colesterolo, b.risultn as Trigliceridi, c.risultn as HDL
FROM (v_accertamenti a LEFT OUTER JOIN v_accertamenti b ON a.nome = b.nome) LEFT OUTER JOIN v_accertamenti c ON a.nome = c.nome
WHERE a.accertamento = 'Colesterolo' and b.accertamento = 'Trigliceridi' and c.accertamento = 'HDL'

Otteniamo:

Nome	Colesterolo	Trigliceridi	HDL
Gigi	200	175	70
Carlo	187	180	40

Aggiungendo poi un campo calcolato secondo la formula LDL = Colesterolo – HDL – Trigliceridi/5 otteniamo:

```
SELECT a.nome, a.risultn as Colesterolo, b.risultn as Trigliceridi,
c.risultn as HDL,
a.risultn - c.risultn - b.risultn/5 as LDL
FROM (v_accertamenti a LEFT OUTER JOIN v_accertamenti b ON
a.nome = b.nome) LEFT OUTER JOIN v_accertamenti c ON a.nome =
c.nome
WHERE a.accertamento = 'Colesterolo' and b.accertamento =
'Trigliceridi' and c.accertamento = 'HDL'
```

otteniamo la seguente tabella:

Nome	Colesterolo	Trigliceridi	HDL	LDL
Gigi	200	175	70	95
Carlo	187	180	40	111

In pratica non é così semplice poiché i valori devono essere considerati nella stessa data e bisogna porre delle condizioni per verificare che vi siano i tre valori contemporaneamente e che non siano nulli.

UNION

Si usa la clausola UNION quando si vuole ottenere una tabella unendo le richieste di due query diverse.

Supponiamo di volere estrarre dagli accertamenti tutti i valori delle glicemie e delle azotemie:

```
SELECT risultn, cognome, nome
FROM v_accertamenti
WHERE accertamento ='glicemia' AND risultn <> 0
UNION
SELECT risultn, cognome, nome
FROM v_accertamenti
WHERE accertamento ='azotemia' AND risultn <> 0
```

ORDER BY 2,3

Due osservazioni:

- la clausola ORDER BY agisce sulla tabella finale. Non è possibile mettere una ORDER BY alla fine di ogni sottotabella.
- ORDER BY seguito da numero intero indica che l'ordinamento deve essere effettuato per la o le colonne corrispondenti al numero indicato, in questo caso la seconda e la terza e cioè il cognome e il nome.

  ORDER BY in questo caso non funziona indicando i nomi delle colonne (ORDER BY cognome, nome).

Query all'interno di altre query

Tramite Sybase SQL possiamo innestare query all'interno di altre query:

Immaginiamo di avere necessità di invitare per lettera tutti i pazienti che non abbiano almeno una misurazione della pressione:

La lista si ottiene facilmente nel modo seguente:

```
SELECT cognome, nome, indirizzo
FROM v_pazienti
WHERE v_pazienti.codice NOT IN
        (SELECT codice
                FROM v_pressione)
ORDER BY v_pazienti. cognome, v_pazienti.nome
```

Vengono selezionati cognome, nome e indirizzo di quei pazienti in cui il codice non compare in un'altra tabella generata estemporaneamente e che comprende tutti i pazienti a cui è stata misurata la pressione

Tutti gli operatori logici che si utilizzano per verificare valori singoli possono funzionare nelle sottoquery, a patto che il risultato di queste sia una riga singola.

Come in una sottoquery possono essere utilizzati gli operatori logici a valore singolo, possono essere utilizzati gli operatori a valori multipli. Se una sottoquery ha come risultato una o più righe, il valore nella colonna per ciascuna riga viene visualizzato in un elenco.

Alcuni aspetti delle sottoquery

Le sottoquery devono essere racchiuse tra parentesi.

Le sottoquery che hanno come risultato soltanto una riga possono essere utilizzate sia con operatori a valori singoli, sia con operatori a valori multipli.

Le sottoquery che hanno come risultato più di una riga possono essere utilizzate soltanto con operatori a valori multipli.

L'operatore BETWEEN _ AND non può essere utilizzato con una sottoquery.

APPENDICE: viste disponibili

Vista v_accertamenti

Nome	Tipo	Dimensione
accertamento	Testo	40
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codminsan	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
datavisita	Data/ora	8
evidenza	Testo	1
icd9	Testo	6
indotto	Testo	1
medico	Testo	40
nome	Testo	20
num_ese	Testo	14
prezzo	Numerico (Precisione doppia)	8
problema	Testo	60
quantita	Numerico (Precisione doppia)	8
risultn	Numerico (Precisione doppia)	8
risuts	Testo	6
sesso	Testo	1
tipo	Testo	3
tipo_ese	Testo	1
usl	Testo	3

Vista v_avvisi

Nome	Tipo	Dimensione
avviso	Testo	40
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
medico	Testo	40
nome	Testo	20
sesso	Testo	1
tipo	Testo	3
usl	Testo	3

Vista v_certificati

Nome	Tipo	Dimensione
certific	Testo	24
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8

datavisita	Data/ora	8
icd9	Testo	6
inizio	Data/ora	8
medico	Testo	40
nome	Testo	20
problema	Testo	60
prognosi	Testo	3
sesso	Testo	1
usl	Testo	3

Vista v_consigli

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
consiglio	Testo	24
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
datavisita	Data/ora	8
icd9	Testo	9
medico	Testo	40
nome	Testo	20
problema	Testo	80
sesso	Testo	1
usl	Testo	3

Vista v_contatti

Nome	Tipo	Dimensione

codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
data_contatto	Data/ora	8
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
medico	Testo	40
nome	Testo	20
ora_contatto	Ora	5
sesso	Testo	1
tipo	Testo	23
usl	Testo	3

Vista v_descrizioni

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
datavisita	Data/ora	8
descriz	Testo	128
icd9	Testo	9
medico	Testo	40
nome	Testo	20

numcar	Numerico	4
problema	Testo	80
segue	Testo	1
sesso	Testo	1
Tipo_descr	Testo	19
usl	Testo	3

Vista v_esenzioni

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascad	Data/ora	8
datascelta	Data/ora	8
datavisita	Data/ora	8
esenzione	Testo	80
icd9	Testo	9
medico	Testo	40
nome	Testo	20
numero	Testo	14
problema	Testo	80
sesso	Testo	1
usl	Testo	3

Vista v_pazienti

Nome	Tipo	Dimensione

autor675	Testo	1
cap	Testo	5
capofam	Testo	1
causadeces	Testo	60
codcomnasc	Testo	6
codcomune	Testo	6
coddecesso	Testo	6
codecapof	Testo	22
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
comunasc	Testo	35
comune	Testo	35
convenzion	Testo	1
datadecess	Data/ora	8
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
indirizzo	Testo	46
inval_av	Testo	1
inval_civ	Testo	1
inval_gue	Testo	1
istruz	Testo	10
medico	Testo	40
nome	Testo	20
nota675	Testo	10
provincia	Testo	2
prvnascita	Testo	2
residenza_cap	Testo	5
residenza_codco	Testo	6
residenza_comun	Testo	35
residenza_indiriz	Testo	46
residenza_provin	Testo	2
serv_milo	Testo	1
sesso	Testo	1

stato_civ	Testo	13
telefono	Testo	13
usl	Testo	3

Vista v_pressione

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
datavisita	Data/ora	8
frequenza	Numerico	8
ICD9	Testo	6
massima	Testo	3
medico	Testo	40
minima	Testo	3
nome	Testo	20
nota	Testo	55
problema	Testo	60
sesso	Testo	1
usl	Testo	3

Vista v_prestazioni

Nome	Tipo	Dimensione
codfiscale	Testo	16

codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
datavbisita	Data/ora	8
descr	Testo	24
diagnosi	Testo	25
ICD9	Testo	6
medico	Testo	40
nome	Testo	20
problema	Testo	60
sesso	Testo	1
tipo	Testo	1
usl	Testo	3

Vista v_problemi

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
dataopen	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
ICD9	Testo	6
medico	Testo	40
nome	Testo	20

problema	Testo	60
sesso	Testo	1
stato_pb	Testo	8
usl	Testo	3

Vista v_richieste

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codminsan	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
datavisita	Data/ora	8
ICD9	Testo	6
indotto	Testo	1
medico	Testo	40
nome	Testo	20
num_ese	Testo	14
prezzo	Numerico	8
problema	Testo	60
quantità	Numerico	8
richiesta	Testo	40
sede	Testo	30
sesso	Testo	1
tipo	Testo	3
tipo_ese	Testo	1
usl	Testo	3

Vista v_scadenze

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascad	Data/ora	8
datascelta	Data/ora	8
ICD9	Testo	6
medico	Testo	40
nome	Testo	20
problema	Testo	60
scadenza	Testo	50
sesso	Testo	1
tipo	Testo	12
usl	Testo	3

Vista v_terapia

Nome	Tipo	Dimensione
atc	Testo	7
atc_descr	Testo	86
attivo	Testo	197
codattivo	Testo	6
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codminsan	Testo	9
codusl	Testo	16

cognome	Testo	30
continuatiivo	Testo	1
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascelta	Data/ora	8
datavisita	Data/ora	8
fascia	Testo	1
ICD9	Testo	6
indotto	Testo	1
medico	Testo	40
nome	Testo	20
num_ese	Testo	14
posologia	Testo	35
prezzo	Numerico	8
problema	Testo	60
quantita	Numerico	8
sesso	Testo	1
terapia	Testo	30
tipo_ese	Testo	1
usl	Testo	3

Vista v_vaccini

Nome	Tipo	Dimensione
codfiscale	Testo	16
codice	Testo	22
codmedico	Testo	10
codusl	Testo	16
cognome	Testo	30
convenzion	Testo	1
datanasc	Data/ora	8
datarevoca	Data/ora	8
datascad	Data/ora	8
datascelta	Data/ora	8

datavacc	Data/ora	8
ICD9	Testo	6
medico	Testo	40
nome	Testo	20
note	Testo	25
numero	Numerico	8
problema	Testo	60
sesso	Testo	1
tipo	Testo	15
usl	Testo	3