SQL: Interrogazioni Avanzate

Prof. Viglietti Francesco

Www.in4matika.altervista.org

Introduzione

In questo modulo si tratteranno interrogazioni più avanzate, che coinvolgono più tabelle.
Questa possibilità impone di imparare il concetto di join .
Si parlerà dei seguenti aspetti:

- uso dei join per eseguire interrogazioni su più tabelle:
  - join naturale, join interno e cross join
  - join diretti e self join
  - join destro, join sinistro
- scrittura di interrogazioni annidate
- uso delle proprietà opzionali di SELECT.

Tutte le interrogazioni del modulo precedente restituivano dati estratti da una sola tabella. Dal momento che si parla di basi di dati normalizzate nelle quali l'informazione è memorizzata in molte tabelle, selezionare da una sola tabella è, come dire, limitativo.
Ciò che rende interessanti le basi di dati relazionali ben progettate è l'interrelazione, cioè le associazioni tra le tabelle (join).

Cross-join e Equi-join

Join per eseguire interrogazioni su due tabelle

Si consideri la seguente interrogazione:

SELECT impiegato.nome, dipartimento.nome
FROM impiegato, dipartimento
WHERE impiegato.IDdipartimento = dipartimento.IDdipartimento;

Come si può vedere, nel comando FROM sono state indicate due tabelle invece di una. In questo esempio, si vogliono trovare i nomi degli impiegati e i nomi dei dipartimenti per cui lavorano. Il risultato è qui mostrato:

nome	nome
Ben Smith	Finanza
Ajay Patel	Ricerca e Sviluppo
Nora Edwards	Ricerca e Sviluppo
Candy Burnett	Ricerca e Sviluppo

Join per eseguire interrogazioni su due tabelle

Come si ottiene questo risultato? Per prima cosa sono state selezionate colonne appartenenti a due diverse tabelle (è stata usata la notazione col punto per differenziare il nome dell'impiegato da quello del dipartimento).

Per ottenere questo si sono dovute includere entrambe le tabelle nel comando FROM.

La parte più interessante di questa query è costituita dalla condizione WHERE.

Se la si eseguisse senza la condizione WHERE, ossia:

SELECT impiegato.nome, dipartimento.nome FROM impiegato, dipartimento;

Si otterrebbe il seguente risultato ----=========>

nome	nome
Ajay Patel	Finanza
Nora Edwards	Finanza
Candy Burnett	Finanza
Ben Smith	Finanza
Ajay Patel	Ricerca e Sviluppo
Nora Edwards	Ricerca e Sviluppo
Candy Burnett	Ricerca e Sviluppo
Ben Smith	Ricerca e Sviluppo
Ajay Patel	Risorse Umane
Nora Edwards	Risorse Umane
Candy Burnett	Risorse Umane
Ben Smith	Risorse Umane
Ajay Patel	Vendite
Nora Edwards	Vendite
Candy Burnett	Vendite
Ben Smith	Vendite

Join per eseguire interrogazioni su due tabelle

L'interrogazione con la condizione WHERE, mostra l'elenco dei dipendenti associati correttamente al loro dipartimento, mentre l'altra mostra tutte le possibili combinazioni tra impiegati e dipartimenti, senza nessun nesso logico! (prodotto cartesiano delle due tabelle).
Chiaramente la condizione WHERE è importante ai fine di trovare le righe che si vogliono nel risultato. Quando si esegue una JOIN, ci si riferisce alla condizione, o insieme di condizioni, usata per congiungere le tabelle con i termini condizioni di JOIN. In questo esempio, la condizione usata è impiegato.IDdipartimento = dipartimento.IDdipartimento che rappresenta il legame tra le tabelle dello schema logico (PK-FK).
Quando si devono ritrovare informazioni suddivise su più tabelle, per avere ciò che si sta cercando occorre usare queste interrelazioni tra tabelle. A volte questo vuol dire dover cercare un percorso a partire dall'informazione che già si ha fino a quella che si desidera.

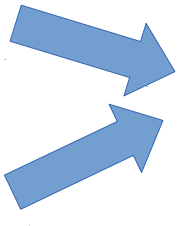
Join per eseguire interrogazioni su due tabelle

Per mostrare come funziona la selezione operata da WHERE, utilizziamo una interrogazione che esplicita i campi utilizzati da where per la selezione. Visualizziamo oltre al risultato le 2 tabelle su cui fare la ricerca. (passiamo per un attimo al database in inglese)

SELECT employee.name, employee.departmentID, department.name, department.departmentID FROM employee, department;

employeeID	name	job	departmentID
6651	Ajay Patel	Programmer	128
7513	Nora Edwards	Programmer	128
9006	Candy Burnett	Systems Adminis	128
9842	Ben Smith	DBA	42

departmentID	name
42	Finance
128	Research and Develop
129	Human Resources
130	Marketing



name	departmentID	name	departmentID
Ajay Patel	128	Finance	42
Nora Edwards	128	Finance	42
Candy Burnett	128	Finance	42
Ben Smith	42	Finance	42
Ajay Patel	128	Research and Develop	128
Nora Edwards	128	Research and Develop	128
Candy Burnett	128	Research and Develop	128
Ben Smith	42	Research and Develop	128
Ajay Patel	128	Human Resources	129
Nora Edwards	128	Human Resources	129
Candy Burnett	128	Human Resources	129
Ben Smith	42	Human Resources	129
Ajay Patel	128	Marketing	130
Nora Edwards	128	Marketing	130
Candy Burnett	128	Marketing	130
Ben Smith	42	Marketing	130

Join per eseguire interrogazioni su due tabelle

La condizione where utilizzata nella interrogazione

SELECT impiegato.nome, dipartimento.nome
FROM impiegato, dipartimento
WHERE impiegato.IDdipartimento = dipartimento.IDdipartimento;

va a cercare sulla tabella risultante dalla interrogazione precedente

SELECT employee.name, employee.departmentID,
department.name, department.departmentID
FROM employee, department;

tutte le righe per cui i campi departmentID coincidono In questo modo ciascuno dei campi employee.name viene associato al corretto campo department.name scartando tutte le associazioni in più create dal prodotto cartesiano.

Join per eseguire interrogazioni su due tabelle

Un’altra cosa da notare, guardando il risultato ottenuto precedentemente, è che entrambe le colonne si chiamano nome perché così sono indicate nel contesto della loro tabella di provenienza. È possibile migliorare la leggibilità usando gli alias:

SELECT impiegato.nome AS nomeImpiegato, dipartimento.nome AS nomeDipartimento
FROM impiegato, dipartimento
WHERE impiegato.IDdipartimento = dipartimento.IDdipartimento;

Si ottiene il seguente risultato:

La presentazione di questo risultato è molto più semplice da capire della precedente

nomeImpiegato	nomeDipartimento
Ben Smith	Finanza
Ajay Patel	Ricerca e Sviluppo
Nora Edwards	Ricerca e Sviluppo
Candy Burnett	Ricerca e Sviluppo

Join per eseguire interrogazioni su più tabelle

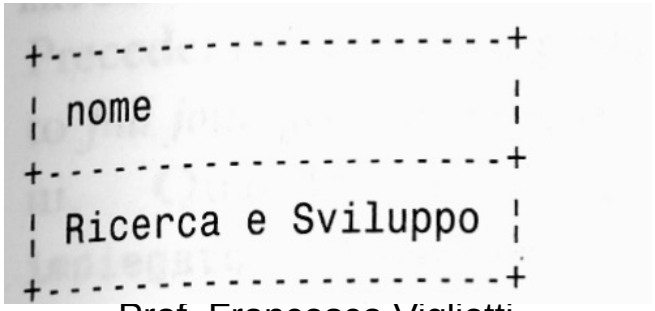
Il principio in base al quale si collegano più di due tabelle è lo stesso.
Si consideri la situazione in cui si vogliono trovare quali dipartimenti, hanno impiegati che hanno svolto un lavoro per il cliente Telco Inc. Come si trova questa informazione?
Il nome del cliente è noto, e guardando nella tabella cliente, si riesce a sapere il suo IDcliente. Con questa informazione si possono cercare i corrispondenti assegnamenti nella tabella assegnamento e sapere quali impiegati hanno lavorato per questo cliente.
A questo punto si ricava IDimpiegato dalla tabella assegnamento e lo si usa per cercare nella tabelle impiegato gli ID dei dipartimenti per i quali questi impiegati lavorano. Da quest'ultima informazione è possibile finalmente andare nella tabella dipartimento e trovare i nomi dei dipartimenti!

Join per eseguire interrogazioni su più tabelle

Dopo aver seguito questo percorso logico attraverso quattro tabelle, occorre scrivere un'interrogazione che esprima questo ragionamento. L'interrogazione ha questa forma:

SELECT dipartimento.nome
FROM cliente, assegnamento, impiegato, dipartimento
WHERE cliente.nome = 'Telco Inc' AND cliente.IDcliente = assegnamento.Idcliente AND assegnamento.IDimpiegato = impiegato.Idimpiegato AND impiegato.IDdipartimento = dipartimento.IDdipartimento;

Questo è il risultato dell'esecuzione dell'interrogazione:



Join per eseguire interrogazioni su più tabelle

Guardando l’interrogazione si può vedere che si sono dovute elencare tutte e quattro le tabelle del ragionamento che si è seguito e le condizioni di join necessarie per passare da tabella in tabella.

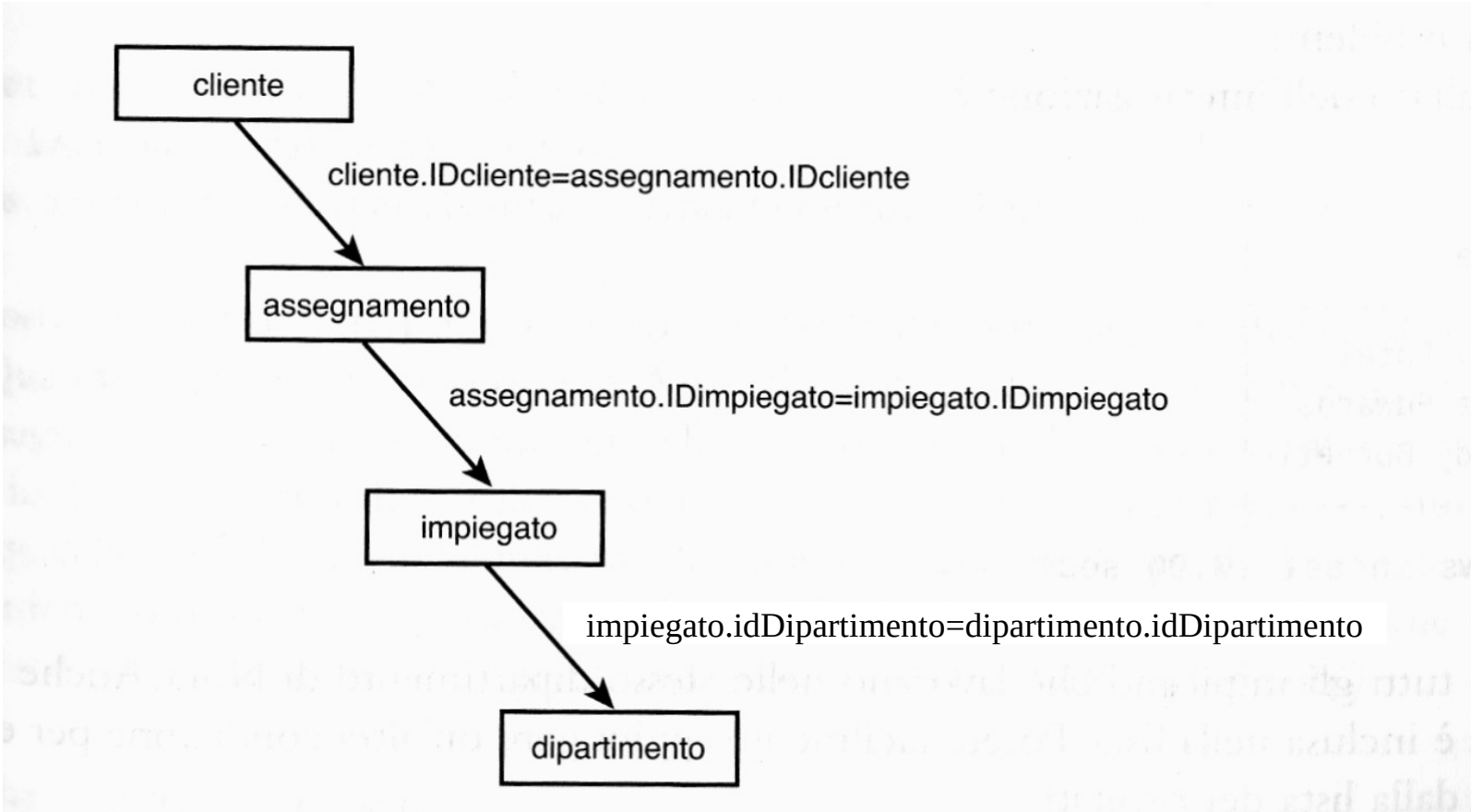
C’è una condizione regolare, cliente, nome = Telco Inc, e una serie di condizioni di join. Notate come ci siano tre condizioni di join per collegare quattro tabelle.

Potete usare quanto detto come guida per capire se sono state incluse tutte le condizioni di join necessarie.

Se si collegano n tabelle, nella maggior parte dei casi si ha un collegamento per ogni coppia di tabelle e di conseguenza si hanno n-1 condizioni di di join.

Join per eseguire interrogazioni su più tabelle

I join di questo esempio sono mostrati nella seguente. Vedete chiaramente che quattro tabelle hanno bisogno di tre (n-1) join.



Il join di base

Precedentemte si è parlato del concetto di prodotto cartesiano, a volte anche chiamato **cross-join**, che restituisce l'intero insieme delle combinazioni (senza clausola WHERE per intenderci). Quando si aggiunge una condizione al join (ad esempio impiegato.IDdipartimento = dipartimento.IDdipartimento) lo si trasforma in qual cosa chiamato **equi-join** o **inner-join** che limita il numero di righe nel risultato.
MySQL ha diverse forme sintattiche che possono essere usate per questo tipo di join. Si consideri l'interrogazione originale:

SELECT impiegato.nome, dipartimento.nome
FROM impiegato, dipartimento
WHERE impiegato.IDdipartimento = dipartimento.IDdipartimento;

Il join di base

Si sarebbe potuto, in modo opzionale, usare la parola chiave JOIN al posto della virgola

SELECT impiegato.nome, dipartimento.nome
FROM impiegato JOIN dipartimento
WHERE impiegato.IDdipartimento = dipartimento.IDdipartimento;

Al posto di JOIN si sarebbe potuto scrivere anche INNER JOIN.

Quando si usa questo tipo di join, MySQL controlla le tabelle che si vogliono collegare ed esegue l'operazione nel modo più efficiente piuttosto che necessariamente nell'ordine in cui le tabelle sono scritte.

A volte l'ottimizzazione dell'interrogazione può causare qualche piccolo errore. Vedremo meglio questo aspetto nel seguito. Se si desidera eludere l'ottimizzatore e forzare MySQL perché colleghi le tabelle nell'ordine in cui sono state elencate è sufficiente sostituite la parola JOIN con STRAIGHT JOIN (join diretto).

ESERCIZI

1. Scrivete un’interrogazione che elenchi il nome di un impiegato e tutte le sue competenze.
2. Inventare il testo di un esercizio simile al precedente che fissi i concetti legati alle interrogazioni che coinvolgono più tabelle. Risolvere quindi l’esercizio

Self-join

Join di una tabella con se stessa (self-join)

Esattamente come si collegano tabelle con altre tabelle, è possibile collegare una tabella a se stessa. Perché si dovrebbe farlo? A volte si cercano delle relazioni tra righe una stessa tabella. Si immagini di voler sapere il nome degli impiegati che lavorano nello stesso dipartimento di Nora Edwards. Si deve trovare nella tabella impiegato l'IDdipartimento per il quale Nora lavora e, quindi, cercare ancora nella tabella impiegato quali impiegati lavorano in quel dipartimento.

Un esempio è mostrato di seguito:

SELECT e2.nome FROM impiegato AS e1, impiegato AS e2

WHERE e1.nome = 'Nora Edwards' AND e1.IDdipartimento = e2.IDdipartimento

Join di una tabella con se stessa (self-join)

In realtà potete vedere che in questa interrogazione sono stati dichiarati due diversi Alias per la tabella impiegato. Ciò equivale a dire a MySQL che si suppone di avere due diverse tabelle, e1 ed e2, che solo per caso contengono gli stessi dati. Di conseguenza è possibile collegarle esattamente come se si avessero due tabelle qualsiasi.

Iniziate trovando la riga di Nora in e1 (WHERE e1.nome = ‘Nora Edwards’). Poi cercate in e2 le righe che hanno lo stesso IDdipartimento di 'Nora Edwards' (el.IDdipartimento = e2.IDdipartiniento).

Questo procedimento può richiedere un po’ di tempo per diventare familiare, ma fintanto che si suppone di lavorare con due tabelle differenti non dovreste avere grossi problemi.

Il risultato dell’interrogazione è:

```
+---------------+
| nome          |
+---------------+
| Ajay Patel    |
| Nora Edwards  |
| Candy Burnett |
+---------------+
```

Join di una tabella con se stessa (self-join)

Il risultato mostra tutti gli impiegati che lavorano nello stesso dipartimento di Nora. Anche Nora stessa è inclusa nella lista.

Si può facilmente aggiungere un'altra condizione per escluderla dalla lista dei risultati:

```
SELECT e2.nome
FROM impiegato as e1, impiegato as e2
WHERE el.nome = 'Nora Edwards'
AND e1.IDdipartimento = e2.IDdipartimento
AND e2.nome != 'Nora Edwards';
```

Il risultato diventa:

```
+---------------+
| nome          |
+---------------+
| Ajay Patel    |
| Candy Burnett |
+---------------+
```

Join di una tabella con se stessa (self-join)

Approfondiamo meglio questo meccanismo. Vediamo il risultato della query senza la condizione WHERE, visualizzando anche le colonne sulle quali fare la selezione:

SELECT e2.name, e2.departmentID, e1.name, e1.departmentID
FROM employee AS e1, employee AS e2

Ricordiamoci che noi vogliamo scoprire il nome degli impiegati che lavorano nello stesso dipartimento di Nora Edwards.
Abbiamo per ora accoppiato a ciascuna riga name, departmentID della tabella di destra, tutte le altre righe.
A noi interessa vedere gli accoppiamenti della sola Nora Edwards.

name	departmentID	name	departmentID
Ajay Patel	128	Ajay Patel	128
Ajay Patel	128	Nora Edwards	128
Ajay Patel	128	Candy Burnett	128
Ajay Patel	128	Ben Smith	42
Nora Edwards	128	Ajay Patel	128
Nora Edwards	128	Nora Edwards	128
Nora Edwards	128	Candy Burnett	128
Nora Edwards	128	Ben Smith	42
Candy Burnett	128	Ajay Patel	128
Candy Burnett	128	Nora Edwards	128
Candy Burnett	128	Candy Burnett	128
Candy Burnett	128	Ben Smith	42
Ben Smith	42	Ajay Patel	128
Ben Smith	42	Nora Edwards	128
Ben Smith	42	Candy Burnett	128
Ben Smith	42	Ben Smith	42

Join di una tabella con se stessa (self-join)

Aggiungiamo quindi la condizione per cui nella tabella di destra il campo name deve essere uguale a “Nora Edwards”

SELECT e2.name, e2.departmentID, e1.name, e1.departmentID FROM employee AS e1, employee AS e2 WHERE e1.name = "Nora Edwards"

Ricordiamoci che vogliamo scoprire il nome degli impiegati che lavorano nello stesso dipartimento di Nora Edwards.

name	departmentID	name	departmentID
Ajay Patel	128	Nora Edwards	128
Nora Edwards	128	Nora Edwards	128
Candy Burnett	128	Nora Edwards	128
Ben Smith	42	Nora Edwards	128

Abbiamo per ora accoppiato alla ri corrispondente a Nora Edwards della tabella di destra, tutte le altre righe della tabella di sinistra, compresa la riga di Nora Edwards stessa (sono 2 tabelle distinte). Risulta ora più chiaro che per individuare i nomi degli impiegati che lavorano nello stesso dipartimento di Nora è sufficiente estendere la condizione WHERE in modo da verificare l’uguaglianza dei campi departmentID

Join di una tabella con se stessa (self-join)

Aggiungiamo quindi la condizione la condizione che verifica l'uguaglianza dei campi departmentID

SELECT e2.name, e2.departmentID, e1.name, e1.departmentID
FROM employee AS e1, employee AS e2
WHERE e1.name = "Nora Edwards"
AND e1.departmentID = e2.departmentID

name	departmentID	name	departmentID
Ajay Patel	128	Nora Edwards	128
Nora Edwards	128	Nora Edwards	128
Candy Burnett	128	Nora Edwards	128

Se vogliamo quindi escludere Nora, basta aggiungere un'altra condizione in and a where che escluda Nora stessa.

SELECT e2.name, e2.departmentID, e1.name, e1.departmentID
FROM employee AS e1, employee AS e2
WHERE e1.name = "Nora Edwards"
AND e1.departmentID=e2.departmentID AND e2.nome!= 'Nora Edwards';

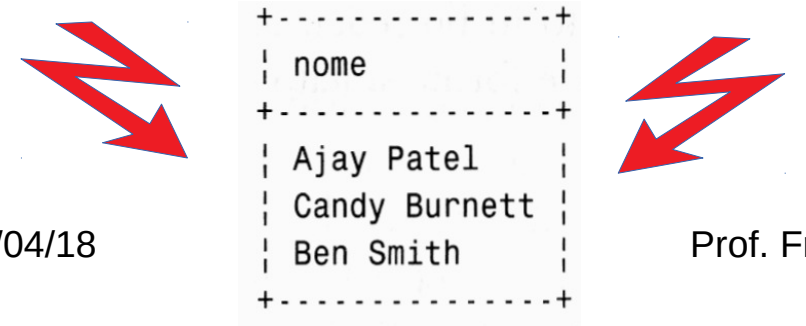
Ajay Patel	128	Nora Edwards	128
Candy Burnett	128	Nora Edwards	128

Per visualizzare in modo corretto è sufficiente restringere le colonne da visualizzare elencate dopo select, tornando quindi alla query originale

Left join e Right join

Join destro e Join sinistro

Quando sono stati usati gli equi-join sono stati utilizzati JOIN, CROSS JOIN, INNER JOIN oppure anche STRAIGHT JOIN e si sono cercate righe collegate tra loro in due o più tabelle.
Cosa succede quando si vogliono trovare le righe di una tabella che non sono collegate a nessuna riga di un'altra tabella?
Consideriamo, ad esempio, la situazione in cui si vogliono trovare quegli impiegati che non hanno mai lavorato per un progetto esterno, vale a dire che non sono elencati attraverso IDimpiegato nella tabella assegnamento. E' possibile trovarli usando il LEFT JOIN (join sinistro):
SELECT impiegato.nome FROM impiegato LEFT JOIN assegnamento ON impiegato.IDimpiegato = assegnamento.Idimpiegato WHERE IDcliente IS NULL;



Join destro e Join sinistro

Controllando la schermata è possibile accertarsi facilmente che il risultato sia proprio quello cercato. Ma come funziona e perché?

Il funzionamento del left join si basa sul considerare la tabella a sinistra del join (in questo caso, impiegato) e nel cercare di trovare tutte le corrispondenze riga per riga con la tabella a destra.

Le righe della tabella di destra che hanno la corrispondente riga nella tabella di sinistra vengono aggiunte di lato alla tabella di sinistra.

Per ogni riga della tabella di sinistra senza corrispondente in quella di destra, il LEFT JOIN inserisce dei valori NULL dove dovrebbero esserci i valori della riga di destra. Potete cercare le righe della tabella di sinistra che non hanno corrispondente in quella di destra trovando quali righe hanno un valore nullo nella colonna (o insieme di colonne) che per la tabella di destra era la chiave prima del join.

Join destro e Join sinistro

Si Torni all'esempio. In questo join, per ogni impiegato che è stato assegnato a un progetto, si ottiene una riga che mostra l'impiegato e i dettagli del lavoro.
Quando un impiegato non ha nessuna riga collegata a lui nella tabella assegnamento, il left join inserisce una "riga vuota" di valori NULL. Queste righe vuote si trovano cercando gli impiegati che abbiano un assegnamento con un valore NULL nella colonna IDcliente (il campo IDcliente è una chiave per la tabella assegnamento, quindi nella tabella originale non può contenere valori nulli).
In questo esempio è stato usato un LEFT JOIN ma si sarebbe potuto utilizzare facilmente anche un RIGHT JOIN (join destro), che funziona esattamente nello stesso modo ma usa la tabella di destra come tabella di base e riempie le righe mancanti dalla tabella di sinistra con valori NULL.

Join destro e Join sinistro

Come prima, ripetiamo il procedimento visualizzando tutti i campi in maniera da evidenziare meglio come ragiona il DBMS.

Eseguiamo la seguente query:

clientID	employeeID	workdate	hours
1	7513	2003-01-20	8.5

employeeID	name	job	departmentID
6651	Ajay Patel	Programmer	128
7513	Nora Edwards	Programmer	128
9006	Candy Burnett	Systems Adminis	128
9842	Ben Smith	DBA	42

```
SELECT *
FROM employee LEFT JOIN assignment
ON employee.employeeID = assignment.employeeID;
```

employeeID	name	job	departmentID	clientID	employeeID	workdate	hours
6651	Ajay Patel	Programmer	128	*NULL*	*NULL*	*NULL*	*NULL*
7513	Nora Edwards	Programmer	128	1	7513	2003-01-20	8.5
9006	Candy Burnett	Systems Adminis	128	*NULL*	*NULL*	*NULL*	*NULL*
9842	Ben Smith	DBA	42	*NULL*	*NULL*	*NULL*	*NULL*

Risulta più chiaro il funzionamento del LEFT JOIN. Ad ogni riga della tabella di destra associo una tutte le righe della tabella di sinistra per cui la condizione ON risulta soddisfatta. Se la condizione non è mai soddisfatta associo NULL a tutti i campi. E' quindi evidente come su una tabella risultante di questo tipo si possano fare selezioni su condizioni di assenza di correlazione che utilizzando il join di base sarebbero meno immediate.

Join destro e Join sinistro

Per chiarire ulteriormente le idee sui join destro e join sinistro vediamo il risultato della seguente query:

```
SELECT *
FROM employee RIGHT JOIN department
ON employee.departmentID = department.departmentID;
```

departmentID	name
42	Finance
128	Research and Develop
129	Human Resources
130	Marketing

employeeID	name	job	departmentID	departmentID	name
9842	Ben Smith	DBA	42	42	Finance
6651	Ajay Patel	Programmer	128	128	Research and Develop
7513	Nora Edwards	Programmer	128	128	Research and Develop
9006	Candy Burnett	Systems Adminis	128	128	Research and Develop
NULL	*NULL*	*NULL*	*NULL*	129	Human Resources
NULL	*NULL*	*NULL*	*NULL*	130	Marketing

Ad ogni riga della tabella di destra vengono associati tutte le righe della tabella di sinistra (ora è un RIGHT JOIN) per cui la condizione specificata da ON è soddisfatta. Ci sono 3 impiegati nel dipartimento 128, quindi 3 occorrenze; c'è un solo impiegato nel dipartimento 42, quindi 1 occorrenza. Gli altri 2 dipartimenti non hanno impiegati, il RIGHT JOIN quindi riempie le righe con NULL.