Linguaggio SQL

DBMS e database

Un database è un archivio elettronico di dati, ossia un sistema per gestire una grande mole di dati tramite un elaboratore elettronico.

I motivi che portano a preferire l'utilizzo del computer a quello del supporto cartaceo sono molteplici:

- Gestione di grandi quantità di dati (es.: anagrafe)
- Basi di dati distribuite (diverse sedi di un'azienda che usano un unico archivio centralizzato)
- Possibilità di effettuare ricerche complesse in breve tempo
- Automatismi (controlli automatici sugli errori di immissione dati, aggiornamenti dei dati)

Un DBMS (DataBase Management System) è un software che consente di gestire database, in particolare fornisce il supporto per la creazione, la modifica della struttura e dei contenuti, l'interrogazione dei dati, la gestione della sicurezza e dell'integrità dei dati.

I dati all'interno di un database relazionale sono organizzati in tabelle, le tabelle possono essere messe in relazione fra loro inserendo nel database determinati vincoli (ad esempio è possibile archiviare i clienti di un hotel nella tabella Clienti e le prenotazioni nella tabella Prenotazioni, specificando il vincolo che non può esistere una prenotazione senza un corrispondente cliente nella tabella correlata).

Per gestire in modo omogeneo gli archivi e favorire la portabilità dei programmi è stato realizzato un linguaggio di programmazione che consente di creare, aggiornare ed interrogare database, questo linguaggio si chiama SQL ed è alla base della quasi totalità del software gestionale oggi in commercio.

Un database è composto da diversi elementi, i principali sono *tabelle* e *query*.

Tabelle

Le tabelle sono gli oggetti che contengono i dati, ogni tabella è costituita da un nome e da una serie di colonne a loro volta dotate di un nome; le righe che compongono la tabella vengono dette *record*.

Esempio

Clienti:=<Cognome, Nome, Telefono, Citta>

Clienti			
Cognome	Nome	Telefono	Citta[1]
Bianchi	Mario	0541254825	Rimini

[1] in SQL i nomi delle tabelle e delle colonne devono essere composti da sequenze di caratteri alfabetici e numerici (sono esclusi spazi e lettere accentate), il cui primo carattere sia alfabetico.

Bianchi	Ettore	0514875978	Bologna
Casadei	Mario	0554578659	Firenze

La prima rappresentazione mostra soltanto la struttura della tabella, la seconda mostra anche i valori contenuti nei record.

Esistono apposite operazioni per la creazione e la distruzione di tabelle, l'inserimento, la modifica e la cancellazione dei dati.

Query

Per operare sui dati contenuti all'interno delle tabelle si usano le query, tramite una query SQL è possibile ricercare dei dati in una tabella o in una combinazione di più tabelle.

Esempio

```
SELECT CLIENTI.COGNOME, CLIENTI.NOME, CLIENTI.TELEFONO, CLIENTI.CITTA
FROM CLIENTI
WHERE CLIENTI.COGNOME="BIANCHI"[1]
ORDER BY CLIENTI.NOME;
```

Questa semplice query restituisce tutti i dati dei clienti il cui cognome è Bianchi.

Il prefisso "CLIENTI." può essere omesso se ciò non crea ambiguità di riferimento, ossia se si sta operando su una sola tabella (come nell'esempio) o se non esistono più campi con lo stesso nome nelle tabelle aperte.

La query precedente si può scrivere più concisamente nella seguente maniera:

```
SELECT *
FROM CLIENTI
WHERE COGNOME="BIANCHI"
ORDER BY NOME;
```

L'asterisco (*) indica che la query deve ritornare tutti i campi.

[1] Le virgolette sono necessarie per evitare che il DBMS consideri Bianchi come nome di una colonna e non come una stringa.

Tramite le query è anche possibile modificare i dati nella tabella o eliminarli.

Le query sono di diverso tipo:

Query di creazione

Servono a creare tabelle definendone la struttura.

Esempio

SELECT CODICE, COGNOME, TELEFONO INTO NUOVA

FROM CLIENTI;

Crea una nuova tabella di nome *nuova* contente solamente i campi *codice, cognome, telefono* della tabella *clienti*.

In questo modo è possibile creare una nuova tabella contenente solo parte dei dati presi da una tabella già esistente o contente dati provenienti da più tabelle.

Query di inserimento

Permettono di inserire dati all'interno di una tabella già esistente.

Database

Esempio

```
INSERT INTO NUOVA ( CODICE, COGNOME, TELEFONO )
SELECT CODICE, COGNOME, TELEFONO
FROM CLIENTI
WHERE CLIENTI.COGNOME=”BIANCHI”;
```

Aggiunge alla tabella *nuova* i campi *codice, cognome, telefono* dei record della tabella *clienti* in cui il cognome è bianchi.

NB: la tabella di destinazione deve essere esistente, se la tabella non è stata creata precedentemente l’esecuzione della query genera un errore.

Query di modifica

Consentono di modificare i dati presenti all’interno di una tabella.

Esempio

```
UPDATE CLIENTI SET NOME = "ETTORE"
WHERE COGNOME="BIANCHI";
```

Cambia il nome di tutti i *bianchi* presenti nella tabella *clienti* in *Ettore.*

Query di eliminazione record

Servono ad eliminare i dati da una tabella.

Esempio

```
DELETE *
FROM CLIENTI
WHERE COGNOME="BIANCHI";
```

Cancella il *nome* di tutti i *Bianchi* presenti nella tabella.

Query di eliminazione tabelle

Permettono di distruggere interamente (dati e struttura) una tabella.

Esempio

```
DROP TABLE CLIENTI;
```

Cancella la tabella Clienti.

Linguaggio SQL

Il linguaggio SQL permette di programmare in più ambienti (Access, SQL server, Oracle, ecc.) allo stesso modo.

Alcuni ambienti (per esempio Access) usano delle estensioni del linguaggio che aggiungono potenzialità alle istruzioni di base, queste istruzioni aggiuntive non sono argomento del corso dato che funzionano solamente all'interno dell'ambiente che le possiede e non permettono una semplice conversione dei programmi.

Il linguaggio SQL non è case sensitive, ossia i caratteri maiuscoli e minuscoli vengono considerati uguali per quanto riguarda i nomi delle tabelle, dei campi e i comandi, il contenuto dei campi può invece essere case sensitive.

L'argomento di maggior interesse all'interno del linguaggio SQL è la creazione di query di selezione, ossia di interrogazioni del database che consentano di estrarre i dati interessanti dalla mole di dati contenuti nell'archivio.

Query di Selezione

Una query è composta da più elementi, vediamo i più importanti

Select

Specifica i campi da visualizzare come risultato dell'interrogazione, con * si visualizzano tutti i campi.

Distinct

Specifica che se il risultato contiene più record uguali deve essere preso in considerazione solo il primo.

Esempio

Clienti			
Cognome	Nome	Telefono	Citta[1]
Bianchi	Mario	0541254825	Rimini
Bianchi	Ettore	0514875978	Bologna
Casadei	Mario	0554578659	Firenze

[1] in SQL non si possono usare lettere accentate né nomi composti da più di 8 caratteri o contenenti spazi bianchi per identificare tabelle o campi.

```
SELECT COGNOME
FROM CLIENTI
```

La query appena proposta restituisce:

Cognome
Bianchi
Bianchi
Casadei

```
SELECT DISTINCT COGNOME
FROM CLIENTI
```

Questa versione della query produce il seguente risultato:

Cognome
Bianchi
Casadei

From

Specifica le tabelle o le query origine dei dati.

Where

Specifica eventuali condizioni per filtrare i dati (l'operazione si chiama *Proiezione*) e permette di unire le tabelle di origine.

Esempio

```
SELECT COGNOME, NOME, SALARIO
FROM CLIENTI
WHERE SALARIO >= 4000;
```

Il risultato contiene solamente in record in cui il salario è maggiore o uguale a 4000.

Order by

Consente di ordinare a piacere i dati che compaiono nel risultato, prevede due tipi di ordinamento:

- asc ordinamento ascendente
- desc ordinamento discendente

È possibile specificare più valori per ottenere ordinamenti a più livelli.

Esempio

```
SELECT COGNOME, NOME, TELEFONO, CITTA
FROM CLIENTI
WHERE COGNOME=“BIANCHI”
ORDER BY COGNOME, NOME;
```

ordina il risultato in base al cognome, i cognomi uguali vengono ordinati in base al nome.

Funzioni di aggregazione

È possibile estrarre dalle tabelle anche valori di aggregazione, cioè valori calcolati su più righe.

Esempio

Clienti			
Codice	Cognome	Nome	Salario
A01	Bianchi	Mario	1.000
A02	Bianchi	Ettore	5.000
B01	Casadei	Mario	3.000

```
SELECT MAX(SALARIO)
FROM CLIENTI;
```

restituisce “5000” ossia il valore massimo fra quelli presenti nel campo salario della tabella.

Allo stesso modo si possono usare MIN() per ottenere il minimo, AVG() per la media aritmetica e SUM() per la somma.

Esiste anche la funzione COUNT() che permette di contare le righe di una tabella.

Esempio

```
SELECT COUNT(*)
FROM CLIENTI;
```

restituisce il numero di righe (nell'esempio 3) contenute nella tabella.

NB: * è obbligatorio.

Non è possibile specificare nella Select altre colonne oltre a quelle che costituiscono gli argomenti delle funzioni di aggregazione.

```
SELECT COGNOME, MAX(SALARIO)        questa query è sbagliata!!!
FROM CLIENTI;
```

Espressioni

All'interno delle query è anche possibile far ricorso ad espressioni algebriche.

Esempio

```
SELECT COGNOME, NOME, SALARIO*1000
FROM CLIENTI;
```

Il risultato è il seguente:

Cognome	Nome	Espr1[1]
Bianchi	Mario	1.000.000
Bianchi	Ettore	5.000.000
Casadei	Mario	3.000.000

[1] Il nome del campo del risultato, se non diversamente specificato dal programmatore, viene assegnato dal sistema e può generare confusione dato che non è significativo.

As

Permette di assegnare dei nomi alle colonne del risultato.

Esempio

SELECT COGNOME AS COGN, NOME, SALARIO*1000 AS REALE

FROM CLIENTI;

Cogn	Nome	Reale
Bianchi	Mario	1.000.000
Bianchi	Ettore	5.000.000
Casadei	Mario	3.000.000

Group by

La clausola group by consente di raggruppare le righe del risultato, il risultato conterrà una sola riga per ogni valore diverso del campo (o dei campi) specificati con Group by.

La Select può contenere solamente:

- i nomi specificati nel Group by
- funzioni d'insieme
- espressioni che uniscano gli oggetti dei due punti precedenti

Esempio

SELECT COGNOME, SUM(SALARIO) AS TOT

FROM CLIENTI

GROUP BY COGNOME;

Risultato

Cognome	Tot
Bianchi	6.000
Casadei	3.000

La query proposta calcola il totale del salario per ogni cognome (non ha senso se non a scopo esemplificativo).

L'uso del Group by costituisce l'unica eccezione alla regola precedentemente citata dell'impossibilità di citare nella Select nomi di colonne che non siano argomenti delle funzioni di aggregazione, in questo caso il vincolo è che compaiano nella Select solamente colonne che compaiono anche nel Group by.

Having

La clausola Having consente limitare il numero di righe che compaiono nel risultato, si può usare solo in presenza di Group by.

Esempio

```
SELECT COGNOME, SUM(SALARIO) AS TOT
FROM CLIENTI
GROUP BY COGNOME
HAVING SUM(SALARIO)[1] >= 4000;
```

Risultato

Cognome	Tot
Bianchi	6.000

Nel risultato compaiono solamente le righe in cui Tot è maggiore o uguale a 4000.

In

Permette di annidare le query per creare interrogazioni complesse.

```
SELECT COGNOME, NOME
FROM CLIENTI
WHERE CODICE IN (SELECT CODICE
        FROM PRENOT
        WHERE DATA="21/07/98");
```

[1] È necessario usare "SUM(SALARIO)", non è possibile usare "Tot"

L'esecuzione della query proposta restituisce i nomi dei clienti che hanno una prenotazione per il 21/07/98.

Predicati

È possibile specificare più di una condizione di selezione all'interno di una clausola Where, per unire le condizioni si usano i predicati *And, Or, Not*.

Esempio

```
SELECT COGNOME, NOME
FROM CLIENTI
WHERE SALARIO > 1000
AND COGNOME = "BIANCHI";
```

L'esecuzione della query restituisce cognomi e nomi dei clienti che hanno un salario maggiore di 1000 e si chiamano Bianchi.

Esempio

```
SELECT COGNOME, NOME
FROM CLIENTI
WHERE SALARIO > 1000
AND NOT (COGNOME = "BIANCHI");
```

Like

È possibile specificare solo una parte di una stringa all'interno di una query SQL utilizzando la clausola *Like* e i caratteri jolly '_' e '%', oppure '[ ]'.

Esempio

```
SELECT COGNOME, NOME
FROM CLIENTI
WHERE COGNOME LIKE "B%";
```

Restituisce i nomi e cognomi dei clienti il cui cognome inizia con B.

Il carattere _ indica un generico carattere, % indica una sequenza di caratteri di qualsiasi lunghezza (anche nulla).

Le parentesi quadre consentono di indicare un range di caratteri, ad esempio [a-g] significa ogni carattere compreso fra a e g (a, b, c, d, e, f, g).

Top

La clausola *Top* fa sì che venga visualizzata solo una parte dei record restituiti dalla query.

Esempio

```
SELECT TOP 2 COGNOME, NOME
FROM CLIENTI;
```

Restituisce nome e cognome dei primi due record della tabella Clienti.

Il parametro di Top può essere una valore numerico intero (come nell'esempio) oppure un valore percentuale espresso nella forma "<Valore> PERCENT".

Esempio

```
SELECT TOP 50 PERCENT COGNOME, NOME
FROM CLIENTI;
```

Restituisce nome e cognome contenuti nella prima metà dei record della tabella Clienti.

Join

Le query viste negli esempi precedenti operavano selezioni su un'unica tabella, se il loro unico utilizzo fosse questo, il loro interesse sarebbe limitato, la vera potenza dell'SQL sta nella possibilità di estrarre dati da più tabelle combinandone i contenuti.

All'interno dei database (normalmente) ci sono più tabelle, ognuna contenente i dati relativi ad una parte del problema preso in esame (es.: clienti e venduto sono normalmente tenuti in tabelle separate); specificando più di un nome dopo la clausola From di una query si realizza un join (ossia un prodotto cartesiano) fra le tabelle specificate.

Esempio

Date le seguenti tabelle:

Clienti		
Codice	Cognome	Nome
A01	Bianchi	Mario
A02	Bianchi	Ettore
B01	Casadei	Mario

Prenot		
Cod_cli	Camera	Data
A01	15	21/08/98
B01	28	22/08/98
B01	05 25	04/09/98

L'operazione di join darà come risultato:

Codice	Cognome	Nome	Cod_cli	Camera	Data
A01	Bianchi	Mario	A01	15	21/08/98
A01	Bianchi	Mario	B01	05 25	04/09/98
A01	Bianchi	Mario	B01	28	22/08/98
A02	Bianchi	Ettore	A01	15	21/08/98
A02	Bianchi	Ettore	B01	05 25	04/09/98
A02	Bianchi	Ettore	B01	28	22/08/98
B01	Casadei	Mario	A01	15	21/08/98
B01	Casadei	Mario	B01	05 25	04/09/98
B01	Casadei	Mario	B01	28	22/08/98

In ogni riga del risultato compare un record per ogni possibile combinazione delle righe della prima e della seconda tabella.

È intuibile che un'operazione siffatta non è di nessuna utilità dato che genera una tabella contenente una mole di record di cui solo pochi sono realmente significativi, per dare un senso al join è necessario eliminare i dati di troppo che compaiono nel prodotto cartesiano delle tabelle, questo risultato si ottiene specificando una condizione nella clausola where.

Esempio

```
SELECT COGNOME, NOME, CAMERA, DATA
FROM CLIENTI, PRENOT
WHERE CLIENTI.CODICE=PRENOT.COD_CLI;
```

NB: Il codice serve solo a ricostruire la struttura completa, non è necessario che compaia anche nella Select.

Codice	Cognome	Nome	Cod_cli	Camera	Data
A01	Bianchi	Mario	A01	15	21/08/98
B01	Casadei	Mario	B01	28	22/08/98
B01	Casadei	Mario	B01	05 25	04/09/98

Per unire due tabelle è necessario che esse contengano un campo "comune" in modo da poter ricostruire la struttura completa, nell'esempio il campo che lega le due tabelle è il codice del cliente chimato rispettivamente *Cod_cli* nella tabella *Prenot* e *Codice* nella tabella *Clienti* (il nome non conta, l'importante è il contenuto).

Se più tabelle hanno campi con lo stesso nome è necessario specificare anche il nome della tabella assieme a quello del campo con una struttura del tipo *<nome tabella>.<nome campo>*, è possibile usare questa struttura anche quando i nomi non sono uguali (come nell'esempio precedente) per migliorare la leggibilità della query.

È possibile usare più condizioni utilizzando la clausola *and* per unire più di due tabelle o per creare filtri complessi.

Esempio

```
SELECT COGNOME, NOME
FROM CLIENTI, PRENOT
WHERE CLIENTI.CODICE=PRENOT.COD_CLI
AND COGNOME="CASADEI";
```

Restituisce tutti i dati relativi alle prenotazioni del signor Casadei.

Autojoin

In alcuni casi è necessario effettuare il join di una tabella con sé stessa, questo crea dei problemi di ambiguità sui nomi usati.

Esempio

Data la seguente tabella:

Clienti			
Cognome	Nome	Salario	Spese
Bianchi	Mario	L. 1.000	L. 1.000
Bianchi	Ettore	L. 5.000	L. 4.000
Casadei	Mario	L. 3.000	L. 2.500
Rossi	Mario	L. 1.500	L. 1.500

Come è possibile avere la lista delle persone che hanno il salario uguale alle spese?

L'unico modo è effettuare il prodotto cartesiano della tabella Clienti con sé stessa, per risolvere le ambiguità che si creerebbero nella query si ricorre agli *Alias*, cioè a sinonimi che permettono di risolvere le ambiguità:

```
SELECT PRIMA.COGNOME, PRIMA.NOME          - selezione dei campi del risultato[1]
FROM CLIENTI PRIMA, CLIENTI SECONDA       - attribuzione degli Alias
WHERE PRIMA.CODICE = SECONDA.CODICE       - selezione sul prodotto cartesiano
AND PRIMA.SALARIO=SECONDA.SPESE;          - selezione del parametro cercato
```

Query annidate

I dati di origine di una query (quelli specificati nella clausola from) non sono necessariamente tabelle ma possono anche essere query, questo permette di realizzare interrogazioni complesse ad un database spezzandole in più query semplici.

Esempio

Clienti		
Codice	Cognome	Nome
A01	Bianchi	Mario
A02	Bianchi	Ettore
B01	Casadei	Mario

[1] Non è importante quale dei due alias viene specificato, è però necessario specificarne uno per evitare ambiguità.

Prenot		
Cod_cli	Camera	Data
A01	15	21/08/98
B01	28	22/08/98
B01	05 25[1]	04/09/98

A:

```
SELECT CODICE
FROM PRENOT
WHERE CAMERA=“28”;
```

B:

```
SELECT COGNOME, NOME
FROM CLIENTI
WHERE CODICE IN A;
```

L’esecuzione della query B restituisce i nomi e cognomi dei clienti che hanno prenotazioni per la camera 28, una formulazione alternativa potrebbe essere:

```
SELECT COGNOME, NOME
FROM CLIENTI, PRENOT
WHERE CLIENTI.CODICE=PRENOT.COD_CLI
AND CAMERA=“28”;
```

Il risultato è uguale, però la prima soluzione è più semplice da realizzare perché permette di scomporre il problema in sottoproblemi più semplici da risolvere.

Oppure si può risolvere il problema con la query:

[1] In questo caso il valore può contenere degli spazi perché non è il nome di un campo o di una tabella ma il valore di un campo.

```
SELECT COGNOME, NOME
FROM CLIENTI
WHERE CODICE IN (SELECT CODICE
        FROM PRENOT
        WHERE CAMERA=“28”);
```

È possibile anche utilizzare *not in* per estrarre tutti i valori tranne quelli specificati.

```
A:
SELECT CODICE
FROM PRENOT
WHERE CAMERA=“28”;

B:
SELECT COGNOME, NOME
FROM CLIENTI
WHERE CODICE NOT IN A;
```

In questo caso il risultato sarà l’elenco dei nomi e cognomi dei clienti che non hanno prenotazioni per la camera 28.

È fondamentale sapere che le query annidate vengono risolte a partire da quella più interna, l’ordine di esecuzione delle query può cambiare il risultato.

Exists

La clausola EXISTS è soddisfatta se la query che segue genera almeno una riga.

Esempio

```
SELECT COGNOME,NOME FROM CLIENTI
WHERE EXISTS
        (SELECT COD_CLI
        FROM PRENOT,CLIENTI
        WHERE COD_CLI=CODICE);
```

L’utilizzo della clausola Exists può essere evitato usando la clausola In.

```
SELECT COGNOME,NOME FROM CLIENTI
WHERE CODICE IN
(SELECT COD_CLI FROM PRENOT);
```

Appendice

I nomi

In SQL standard i nomi delle tabelle e delle colonne devono essere composti esclusivamente da sequenze di caratteri alfabetici e numerici (sono esclusi spazi e lettere accentate), il cui primo carattere sia alfabetico, non è importante il case (maiuscole/minuscole).

In alcunti DBMS è consentito l'uso di spazi, lettere accentate e caratteri speciali nei nomi, per poterli usare all'interno delle query è necessario scrivere il nome fra parentesi quadre o fra altri caratteri speciali.

Cenni di progettazione di basi di dati

Nel corso degli anni dall'inizio dello sviluppo dei DBMS ad oggi sono state elaborate alcune regole di base per lo sviluppo di database "ben funzionanti", vediamo nel seguito le più importanti:

Valori Atomici

I valori contenuti all'interno dei campi non devono essere composti, non è cioè ammissibile inserire più valori all'interno di uno stesso campo, ogni campo deve contenere un unico valore non scomponibile.

Esempio

Consideriamo la tabella Clienti vista in precedenza, in questa nuova versione ogni persona può ricevere un diverso salario nei vari mesi:

Clienti		
Cognome	Nome	Salario
Bianchi	Mario	Gennaio 1.000 Febbraio 2.000 Marzo 1.500
Bianchi	Ettore	Gennaio 5.000 Febbraio 3.000 Marzo 5.000

La soluzione proposta è errata perché il campo Salario contiene valori composti, per risolvere la situazione si può operare in due modi

a) si crea un campo per ogni mese

b) si crea un record per ogni mese

Appendice

la soluzione **b** è sempre applicabile, la soluzione **a** è applicabile solo se si sa a priori quanti sono i valori distinti di cui ci si dovrà occupare (come nel caso dell'esempio).

Soluzione a:

Clienti				
Cognome	Nome	Sal_Gen	Sal_Feb	Sal_Mar
Bianchi	Mario	1.000	2.000	1.500
Bianchi	Ettore	5.000	3.000	5.000

Soluzione b:

Clienti			
Cognome	Nome	Salario	Mese
Bianchi	Mario	1.000	Gennaio
Bianchi	Mario	2.000	Febbraio
Bianchi	Mario	1.500	Marzo
Bianchi	Ettore	5.000	Gennaio
Bianchi	Ettore	3.000	Febbraio
Bianchi	Ettore	5.000	Marzo

Ridondanza dei dati

È importante limitare al minimo la ridondanza dei dati perché e una possibile fonte di errori e provoca uno spreco di spazio nelle tabelle.

Esempio

La tabella dell'esempio precedente è limitatamente ridondante:

Clienti			
Cognome	Nome	Salario	Mese
Bianchi	Mario	1.000	Gennaio
Bianchi	Mario	2.000	Febbraio
Bianchi	Mario	1.500	Marzo
Bianchi	Ettore	5.000	Gennaio

Appendice

Bianchi	Ettore	3.000	Febbraio
Bianchi	Ettore	5.000	Marzo

Si vede chiaramente che le coppie formate da nome e cognome sono ripetute più volte, questo può portare ad errori durante l'uso del database: se per un errore di digitazione si scrive Marco al posto di Mario in un record questo fa perdere una parte delle informazioni relative alla persona in questione e genera una nuova persona che non dovrebbe esistere.

Per ovviare a questa ridondanza è necessario spezzare la tabella in due tabelle separate:

Clienti		
Cognome	Nome	Codice
Bianchi	Mario	01
Bianchi	Ettore	02

Salari		
Cod_Cli	Salario	Mese
01	1.000	Gennaio
01	2.000	Febbraio
01	1.500	Marzo
02	5.000	Gennaio
02	3.000	Febbraio
02	5.000	Marzo

In questo modo non si ha ridondanza dei dati ed è possibile ricostruire la struttura completa effettuando un join fra le due tabelle.

Chiavi

In ogni tabella deve essere possibile individuare una combinazione di colonne (al limite tutte) che identifichi univocamente un record.

Esempio

Clienti		
Cognome	Nome	Codice

Bianchi	Mario	01
Bianchi	Ettore	02

Nella tabella proposta si possono identificare due chiavi:

- Cognome + Nome

- Codice

È da preferirsi la seconda possibilità per due motivi:

1) la prima chiave impedisce l'inserimento di più persone con lo stesso nome e cognome (in alcuni contesti questo è corretto);

2) la seconda chiave è più "piccola" della prima, quindi è più efficiente.

In alcuni casi si aggiunge un campo ad una tabella al solo fine di usarlo come chiave:

Clienti			
Cognome	Nome	Telefono	Citta
Bianchi	Mario	0541254825	Rimini
Bianchi	Ettore	0514875978	Bologna
Casadei	Mario	0554578659	Firenze

Se nella tabella proposta non si pone il vincolo sull'unicità di Nome e Cognome la chiave è molto complessa, infatti l'unica scelta possibile è Cognome + Nome + Telefono, questa è indubbiamente troppo complessa, conviene quindi aggiungere un campo da usare come chiave:

Clienti				
Codice	Cognome	Nome	Telefono	Citta
1	Bianchi	Mario	0541254825	Rimini
2	Bianchi	Ettore	0514875978	Bologna
3	Casadei	Mario	0554578659	Firenze

Ora è possibile definire il campo Codice come chiave (imponendo che non sia possibile avere codici duplicati) in modo da avere una chiave "piccola" e comoda da usare nei join.