OpenOffice.org 2.0 e i Database

Introduzione all'uso dei Database con OpenOffice.org 2.0

Versione 0.99 – Dicembre 2005

Soft.Com

© 2005 **Filippo Cerulo** – Soft.Com Sas

www.softcombn.com - email: filippo.cerulo@softcombn.com

OpenOffice, MySql e *PostgreSQL* sono Marchi Registrati dai rispettivi proprietari.

Quest'opera è rilasciata sotto la licenza *Creative Commons*

"Attribuzione - Non commerciale - Non opere derivate 2.0 Italia."

Per visionare una copia di questa licenza visita il sito web http://creativecommons.org/licenses/by-nc-nd/2.0/it/ o richiedila per posta a Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, Usa.

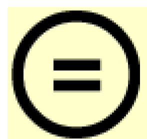
Tu sei libero:

- di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire o recitare l'opera

Alle seguenti condizioni:

Attribuzione. Devi riconoscere il contributo dell'autore originario.

Non commerciale. Non puoi usare quest'opera per scopi commerciali.



Non opere derivate. Non puoi alterare, trasformare o sviluppare quest'opera.

- In occasione di ogni atto di riutilizzazione o distribuzione, devi chiarire agli altri i termini della licenza di quest'opera.
- Se ottieni il permesso dal titolare del diritto d'autore, è possibile rinunciare ad ognuna di queste condizioni.

Le tue utilizzazioni libere e gli altri diritti non sono in nessun modo limitati da quanto sopra

2. Il mondo dei Database

2.1 Primi passi....

Questa sezione serve ad introdurre alcuni concetti indispensabili per le persone che non sono molto esperte di Database. Se la cosa vi annoia, saltatela pure.

La traduzione letterale di ***Database*** è ***Base di dati***, e serve almeno per farsi un'idea dell'argomento che andiamo ad affrontare. Più semplicemente, potremo indicare un Database con la generica parola **archivio**. Ora, un archivio è una idea semplice, nulla di astratto, usiamo ogni tipo di archivio molto più spesso di quanto si creda. Basta un elenco telefonico per capire cos'è un archivio, basta guardare la rubrica del cellulare....

Potremmo definire un a*rchivio* (*database*) come un insieme di informazioni, organizzate in una struttura logica, spesso ordinate secondo una propria necessità, con uno o più caratteristiche in comune. Guardiamo appunto una rubrica telefonica: la caratteristica in comune è "archivio dei numeri della rubrica di Giuseppe", la struttura comprende due informazioni (il nome ed il numero), l'ordine è di solito quello alfabetico.

Nome	**Numero**
Carla	340 1234
Elisa	06 54678
Giuditta	02 987456

Questa, nel gergo dei Database, è una **Tabella** (***table***), che comprende due **Campi** *(**field**)*, *Nome* e *Numero*. La Tabella comprende tre **Righe** chiamate anche **Schede** *(**row, record**)*. Un **Database** è di solito un ***insieme di Tabelle*** che possono essere messe in ***relazione*** tra loro tramite connessioni logiche (presenza in più tabelle della stessa informazione). Da questa definizione deriva il nome di **Database Relazionale** *(RDBMS, Relational Database Management System)* assegnato alla tipologia di prodotti che stiamo esaminando. Attenzione, spesso si indica con lo stesso nome (*Database*) sia il *motore* cioè il software che mi permette di gestire le informazioni, sia le *informazioni* stesse. Questo, in generale, non è corretto.

Tecnica

Esiste una *definizione* molto precisa dei database relazionali. Quindi un Database, per poter essere correttamente definito *relazionale*, deve soddisfare una serie di regole che qui sarebbe lungo ed improduttivo descrivere. Secondo alcuni, ad esempio, *MySql*, siccome non possiede TUTTE queste caratteristiche, NON E' a rigore un Database relazionale. Questo però a noi interessa davvero poco....

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

2.2 Progettiamo un Database

Abbiamo quindi definito la **Tabella** come ***l'unità logica di un Database***. Ovviamente possiamo avere Database composti da una sola Tabella, ma direi che sono casi particolari. Ora siamo perciò di fronte al nostro amato PC e vogliamo cominciare a creare il nostro primo Database; la prima cosa da fare è quindi... *un bel passo indietro*.

Infatti l'approccio più sbagliato che possa esistere è partire con la struttura di un Db senza averci prima ben riflettuto... magari davanti ad un bel foglio bianco e con una cara e vecchia penna in mano. Voglio dire che in generale è fondamentale "progettare" il Database *PRIMA* di mettere la manine sulla tastiera, ed quindi il buonsenso consiglia di porsi in anticipo le seguenti domande:

1. di quante e quali unità logiche (*tabelle*) deve essere composto il mio Db ?
2. per ogni tabella, quali informazioni (*campi*) devo comprendere ?
3. per ogni campo di ogni tabella, che tipo di informazioni devo archiviare ?
4. quali sono i campi su cui sarà necessario eseguire un *ordinamento* ?
5. che *relazioni* ci sono tra le varie tabelle ?
6. che *operazioni* desidero eseguire su ogni tabella ?

Avrete quindi capito che, siccome "chi ben comincia etc. etc.", rispondere a queste domande all'inizio del lavoro vi eviterà problemi nel seguito. Questo non significa che in corso d'opera non si potranno fare variazioni, ma cambiare la struttura di Db complessi quando già è iniziato il caricamento dei dati è davvero complicato. Inoltre, non prevedere qualche piccolo dettaglio può portare a risultati pericolosi, vi ricordate l'affare dell'anno 2000 ?

2.3 La Tabella

Una **Tabella**, abbiamo visto, è composta da **Campi**. Possiamo pensare al *Campo* come l'intestazione di colonna di una lista, ma in realtà è molto di più di una descrizione. In genere, infatti un *Campo* incorpora numerose *proprietà*, cioè *caratteristiche* che, una volta impostate, determineranno in modo preciso il tipo di informazione che quel *Campo* può contenere.

Esaminiamo un po' più in dettaglio queste caratteristiche:

- Il **nome** del Campo è in genere la *descrizione* dell'informazione (ad es. *numero di telefono*). Non ci sono molti vincoli sul nome, vi consiglio però di non sceglierlo troppo lungo, ma allo stesso tempo abbastanza esplicativo. Nel nostro caso andrebbe ad es. bene *NumTel*. Se il Db comprende più Tabelle in cui appare lo stesso campo (se ho, ad esempio, la Tabella Clienti e quella Fornitori), e non sono in relazione tra loro, può essere utile anteporre al nome del campo un indicativo della Tabella stessa, ad es. "CNumTel" e "FNumTel". In genere si possono usare gli *underscore* (Num_Tel) ma io lo trovo poco pratico.

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

- Il **tipo** indica la tipologia della informazione da archiviare nel campo. Le tre categorie principali sono ***stringa***, ***numero*** e ***data/ora***. All'interno di ogni categoria possiamo scegliere molte ulteriori tipologie, ed inoltre, come vedremo, esistono *tipi* particolari (come *binario* o *timestamp*) non riconducibili facilmente agli altri.
- La **lunghezza** misura la dimensione massima che vogliamo assegnare all'informazione contenuta nel campo. Per le *stringhe* si indicano i caratteri (ad es max 30 caratteri per il campo *nome*), per i *valori numerici* il discorso è un po' più complesso (lo vedremo in seguito), per *data/ora* esistono formati standard con diverse occupazioni di memoria a seconda dell'intervallo dei valori che si vuole archiviare.
- **Ammetti null** è un segnalatore che indica la possibilità di archiviare nel campo anche valori *nulli* (a prima vista può sembrare inutile, invece la scelta è assai importante).
- **Valore predefinito** è il valore che dovrà assumere in automatico il campo all'immissione di una nuova riga se non viene modificato dall'utente. Ad esempio, se la maggior parte dei nostri clienti è della provincia di Milano, potremmo volere che il campo Provincia sia riempito dalla stringa *MI*. Se nel valore predefinito non si specifica niente, di solito il campo assume il valore *null*. Per i campi numerici è particolarmente importante assegnare come valore predefinito lo *zero* (come vedremo tra poco).
- **Chiave** oppure i**ndice** stabilisce se il campo debba essere indicizzato e se l'indice è univoco.

Queste *proprietà del campo* si ritrovano quasi identiche in praticamente tutte le varietà di Db Server. Ogni prodotto, poi, implementa *aggiunte* magari non standard, ma egualmente importanti. Ora descriveremo più in dettaglio le principali tipologie di Dati, in modo da capire meglio come strutturare le nostre Tabelle.

2.4 Indici

Alla domanda: *data una Tabella, quali campi è necessario indicizzare?* la risposta è *dipende...* . Cioè in pratica non si risponde, perché non c'è una regola generale universalmente condivisa. Ognuno, in base alla propria esperienza, decide in modo autonomo. Inoltre gestire poche centinaia oppure alcune migliaia di record sono cose ben diverse, come pure usare da soli il proprio Db oppure far parte di una rete con decine di macchine. Questo non significa che non vi posso dare qualche buon consiglio....

Innanzi tutto cerchiamo di capire **cos'è un indice**. Una volta definita la struttura della nostra tabella, ogni volta che aggiungiamo una riga, il nostro motore di Db *accoda* sul disco le informazioni all'archivio aperto. Tornando al nostro semplice esempio della Rubrica del cellulare, quando aggiungiamo i nomi, di certo non lo facciamo in ordine alfabetico; ma comunque, nella consultazione, l'ordine alfabetico ci è molto comodo. Quindi il software del nostro cellulare crea un *indice* sul campo *nome*, in modo da poterci fornire le informazioni nell'ordine più logico.

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

Quindi un *indice* è, in generale, un *ordinamento* creato su uno o più *campi* in modo che il reperimento delle informazioni contenute nell'indice stesso sia molto rapido. Non è il caso di spiegare in dettaglio come questo avvenga a livello di programmazione, accontentiamoci dei risultati. Allora....

Consiglio 1: i motori di Db funzionano meglio se in ogni tabella esiste un *indice univoco*, cioè un valore *specifico* e *diverso* per ogni riga. Questo campo di solito si chiama ***Chiave Primaria*** (***Primary Key***). Possiamo perciò definire la "*chiave primaria*" come *un campo che assume un valore diverso per ogni riga della tabella, e quindi identifica univocamente la riga stessa*.

Consiglio 2: i campi *Interi ad incremento automatico* sono (come vedremo) i candidati ideali per le *Chiavi Primarie*. Infatti i motori di Db hanno prestazioni ottimali sugli indici numerici interi.

Consiglio 3: create un indice sui campi che desiderate siano ordinati sui prospetti di stampa (report). Se ho un archivio clienti, e mi serve una stampa in ordine alfabetico, dovrò creare un indice sulla *Denominazione*.

Consiglio 4: create un indice sui campi che userete per cercare delle informazioni. Se ho un elenco di libri, una buona idea è creare un indice sul campo *autore* (cercherò sicuramente i libri per autore)

Consiglio 5: se avete due tabelle correlate (messe in relazione) tra loro, create indici sui campi comuni (questo, se non vi è chiaro, lo capirete più avanti)

Consiglio 6: troppi indici rallentano il sistema. Molti utenti ancora inesperti sono portati ad indicizzare tutto l'indicizzabile. Sbagliato. Ogni operazione di modifica dei dati, comporta, se il campo è indicizzato, anche l'aggiornamento degli indici stessi. Inoltre gli indici occupano memoria e spazio su disco. Quindi, mi raccomando, parsimonia.

Consiglio 7: l'efficacia di un indice è inversamente proporzionale alla lunghezza del campo. Più il campo è piccolo, più l'indice è efficiente. Se avete, nella vostra tabella, un campo *note* di 200 caratteri, creare un indice non è una buona idea.

Consiglio 8: evitate gli indici su campi che assumono solo pochi valori. Se ho un campo che può contenere ad esempio solo "*S*" per Si e "*N*" per No, un indice peggiora le prestazioni e non serve a niente.

Consiglio 9: i campi scelti come indice non dovrebbero contenere il valore *null*, perché i motori Db gestiscono male questa situazione. Ovviamente neppure la chiave primaria deve ammettere valori *null*.

TIP

Spesso, negli ambiti più vari, può risultare impossibile gestire Tabelle che non dispongono di una chiave primaria. Vi consiglio, quindi, di utilizzarla sempre nella progettazione dei vostri Db.

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

2.5 Principali Tipi di Dati

Ogni Server di Database possiede un lungo elenco di tipologie di Dati gestibili. In realtà i tipi più comuni (numeri, testo e date) sono molto simili, come sarà chiaro più avanti, in tutti i *motori* di Db.

2.5.1 Campi di tipo Stringa

Una **Stringa** è in generale una informazione alfanumerica di lunghezza variabile (un nome, una via, un titolo etc.). Questa informazione può essere archiviata in due modi:

1. posso stabilire un numero massimo di caratteri, e riservare *sempre* un numero di byte fisici equivalenti nel mio archivio (tipo ***char***)
2. posso stabilire un numero massimo di caratteri, ma archiviare solo quelli *effettivamente digitati* dall'utente (tipo ***varchar***)

Nel primo caso saranno aggiunti sempre tanti spazi quanto servono a raggiungere la lunghezza specificata, nel secondo invece lo spazio occupato è in funzione dei caratteri digitati. Ad esempio supponiamo di avere il campo *nome* di lunghezza max 20 caratteri; se scrivo nel campo la stringa *Carlo*, nel primo caso avrò usato 20 byte, nel secondo solo 5.

Allora, vi chiederete, visto che è più conveniente, dal lato dell'occupazione di spazio, usare il tipo *varchar*, il *char* che ci sta a fare ? In effetti il *char* si usa quando l'informazione è formata da pochi caratteri e magari deve essere anche indicizzata. Quindi il *char*, per esempio, va bene per archiviare il campo *CAP* di un indirizzo, oppure un *Codice Cliente* di 5 caratteri (che sicuramente andrà indicizzato). Il fatto è che i campi a lunghezza fissa come *char* sono più veloci da manipolare da parte del motore Db, e quindi anche più indicati nella creazione di indici.

TIP

La lunghezza di un campo stringa andrebbe attentamente valutata; da una parte, se si imposta a troppo pochi caratteri, si rischia di non poter archiviare informazioni lunghe, dall'altra ogni carattere in più significa spazio sprecato e velocità abbassata. La mia regola (ovviamente opinabile) è che se la lunghezza è inferiore a 10 caratteri uso *char*, altrimenti *varchar*. Per i campi di denominazione (nome e cognome, nome di aziende etc.) direi che 50 caratteri sono sufficienti. Negli altri casi un po' di sperimentazione non guasta.

In generale il massimo numero di caratteri archiviabile in un campo ***char*** o ***varchar*** è di 255 (meglio, la lunghezza totale del campo non può eccedere 255 byte). Se abbiamo la necessità di archiviare stringhe più lunghe è possibile usare campi di tipo ***text*** che sono molto più *capienti* (anche fino a 2^32, cioè 4.294.967.295 o 4GB).

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

2.5.2 Campi di tipo Numerico

I campi di tipo numerico si dividono in due grandi sotto categorie: **interi** e **decimali**.

I campi di tipo ***numerico intero***, a seconda dell'intervallo di valori che possono contenere, si dividono in varie classi. In generale i tipi più comuni occupano due (***smallint***), quattro (***integer*** o ***int***) oppure otto byte (***bigint***) di spazio. Il numero *Intero* classico occupa *quattro* byte e di solito può contenere un range numerico abbastanza ampio. Ad esempio in *MySql* un campo ***int*** può contenere da -2.147.483.648 a 2.147.483.647. Questi numeri *possono* (***signed***) oppure *non possono* (***unsigned***) contenere il segno, e quindi assumere valori negativi. Ogni motore di Db indica gli interi in modo leggermente diverso, quindi documentarsi non fa certo male.

I numeri **decimali**, a loro volta, possono dividersi in ulteriori due classi: il tipo ***numeric*** (chiamato anche ***decimal***), che comporta calcoli esatti in quanto il numero di cifre decimali è fisso, ed il tipo ***real*** o ***double precision*** che comporta calcoli arrotondati perché eseguiti in virgola mobile. Questi ultimi sono una implementazione dello standard IEEE 754 per l'aritmetica in virgola mobile. In generale è sempre opportuno usare, per i numeri decimali, il tipo *numeric.* Nella definizione di questi tipologie di dati di solito si specificano *due* valori : la *precisione* e la *scala*. La *scala* è il numero di cifre decimali da considerare, mentre la *precisione* è il numero totale di cifre che il numero può contenere (contando la parte intera più la parte decimale). Così, ad esempio, il numero 123,4567 ha una *scala* di quattro ed una *precisione* di sette. Alcuni motori di Db hanno anche un campo di tipo ***currency*** o ***money*** per l'archiviazione dei valori di *valuta*. Dove manca o dove non è utilizzabile si può usare ad esempio un tipo ***numeric(16,4)***. Se si vuole arrotondare i calcoli a due cifre decimali, va bene anche ***numeric(14,2)***.

TIP

> Sarebbe sempre meglio controllare bene l'intervallo dei valori archiviabili nei tipi di dati numerici del motore di database che vogliamo usare. Ad esempio il tipo *Intero* di Ms Access (Jet) può contenere valori tra -32.728 e 32.767, e quindi equivale a *smallint* di MySql. Allo stesso modo, il tipo *currency.* o *valuta.* di Ms Access corrisponde al tipo *Decimal* con precisione 19 e scala 4 di MySql. La differenza è che mentre il campo di Access occupa sempre 8 byte, quello di MySql occupa *un numero di byte uguale alla precisione* (nel nostro caso quindi 19).

2.5.3 Campi di Tipo Data/Ora

Tutti i motori di Db hanno tipologie specifiche di campi per la manipolazione di date ed orari. I tipi più comuni si chiamano ***Time***, ***Date***, ***DateTime*** e possono, come è facilmente intuibile, contenere rispettivamente un *Orario*, una *Data* ed una *Data/Orario* completa.

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

Ogni prodotto però implementa diverse varianti, dall'intervallo di date che è possibile archiviare, a vari sottogruppi di tipologie (*SmallDateTime*), al formato di memorizzazione ed all'uso di *Timezone*. Quindi documentarsi è indispensabile.

2.5.4 Campi di Tipo Booleano

Questa tipologia di campi può contenere solo due "stati" logici, di solito indicati come ***true*** (vero) o ***false*** (falso). Altre rappresentazioni possibili per *true* sono *1, t, y, yes*, e per *false*: *0, f, n, no*. La rappresentazione interna è un numero intero che può assumere il valore di 0 o 1 (oppure -1). Uno stato non definito si indica col valore *null*.

2.5.5 Campi di tipo Binario

Un campo ***Binario*** può, appunto, contenere dati di tipo binario, cioè sequenze di byte. Sono usati per archiviare informazioni di tipo "grezzo" (ad esempio immagini o documenti in formato nativo), che occupano molto spazio. Non sono indicizzabili, ed andrebbero usati con cautela.

2.5.6 Campi particolari: Intero ad incremento automatico

Come abbiamo visto nel paragrafo dedicato agli indici, può essere a volte utile che il valore di un campo sia stabilito in automatico dal motore Db, secondo una progressione numerica. Questo campo può, ad esempio, rappresentare un codice univoco da assegnare alla scheda senza che noi dobbiamo preoccuparci di fare niente. Campi di questo tipo sono i candidati migliori per la definizione di *Chiavi Primarie*. In generale si tratta di un campo di tipo ***Integer*** associato alla proprietà di ***auto_increment*** (incremento automatico) o ***Identità***. Alcuni Server definiscono questo tipo di campo come ***Serial***, ma la sostanza non cambia.

2.5.7 Campi particolari: Timestamp

Il ***Timestamp*** è in generale un tipo di campo ***Data/Ora*** oppure ***Float*** aggiornato dal sistema. In ambienti multiutente è particolarmente utile per gestire le modifiche concorrenti sulla stessa tabella. Siccome alcuni *front end* per database funzionano male se in ogni tabella non è presente almeno un campo *Timestamp*, noi lo aggiungeremo sempre, tanto non costa niente. Ad esempio, *Ms Access* si rifiuta di funzionare bene su Tabelle *MySql* che non hanno il *Timestamp*. Questo, in funzione di OOo potrebbe essere ininfluente, ma certamente non vogliamo che il nostro Db in futuro non sia leggibile da qualsiasi applicazione, giusto ?

2.6 Vincoli (Constraint)

Ad un singolo Campo, oppure ad un'intera tabella, è di solito possibile applicare dei "*vincoli*" (in inglese ***constraints***). Ogni motore di Db ha delle particolarità, ma in generale possiamo avere:

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

- **vincoli *check***: servono a controllare i valori immessi in uno o più campi di una Tabella; ad esempio potrei definire un vincolo check che mi impedisce di immettere valori negativi in un campo numerico;
- **vincolo *not null*** : impedisce l'immissione di valori nulli nel campo;
- **vincolo *unique*** : di solito collegato ad un indice, impedisce di duplicare un valore in più righe di una stessa tabella;
- **vincolo *primary key*** : individua la chiave primaria di una tabella; include le caratteristiche di unique e not null; è di solito il campo che individua in modo univoco la singola riga;
- **vincolo *foreign keys*** : indica le chiavi esterne della tabella, come vedremo nel paragrafo dedicato all'integrità referenziale;

i vincoli vengono controllati, durante le fasi di immissione e variazione dei dati, direttamente dal motore di Db, e sono un ottimo strumento di verifica della coerenza del Database.

2.7 Il nostro Database di esempio

Per meglio illustrare i concetti che andremo ad introdurre, abbiamo bisogno di un piccolo Archivio di esempio, non troppo complesso ma nemmeno banale. Dopo una lunga (e sofferta) riflessione, ho deciso che gestiremo una **Mediateca**. Ora vi chiederete che sarà mai questo oggetto sconosciuto: bene, è un impasto di Videoteca, Biblioteca, Discoteca, Emeroteca etc. In pratica tutto quello che finisce in -teca ...

Vogliamo, cioè, creare un Archivio che ci permetta di gestire in modo flessibile qualsiasi tipo di *Media* desideriamo catalogare. Inoltre visto che alle nostre cose ci teniamo, ho stabilito che potremmo anche usare un piccolo promemoria per sapere a chi e quando abbiamo (sconsideratamente) fatto un prestito.

Trattandosi di un esempio didattico, non andremo troppo per il sottile, quindi guru dei Db trattenete commenti inopportuni. Chi vuole poi potrà studiare e migliorare....

2.7.1 Struttura degli Archivi

Cominciamo con una piccola analisi preliminare delle nostre esigenze, in modo da definire le principali informazioni che andremo a gestire. L'Archivio principale destinato a contenere le informazioni sui nostri Media (***TbMedia***) potrebbe essere strutturato così:

Campo	**Descrizione**
Id	Chiave primaria – Intero ad incremento automatico
Desc	Descrizione
TipoSupp	Tipo del supporto di memorizzazione (CD, DVD, Rivista, File, etc.)
Argom	Argomento o classificazione

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

Campo	Descrizione
Ubicazione	Ubicazione (Scaffale, Scatolo, Num Rivista, Percorso su Hd etc.)
Prezzo	Prezzo (di acquisto, di vendita, di prestito...), forse non necessario, ma sicuramente didattico
Note	Annotazioni libere
Prestato	Si / No

Bene, dobbiamo ora definire il tipo di campo da assegnare ad ogni informazione della nostra Tabella. La cosa più immediata, ad esempio per il campo "*TipoSupp*"o "*Argom*", sarebbe ovviamente *stringa*, magari di lunghezza 20 caratteri. Così, ad esempio, una parte del nostro archivio potrebbe essere :

Desc	TipoSupp	Argom
Pearl Jam - Binaural	CD Audio	Rock
Fromm – Avere o Essere ?	Libro	Filosofia
Harry Potter e la camera dei segreti	DVD Film	Fantasy
Peter Gabriel – Growing Up Live	DVD Musicale	Rock
Condividere risorse con Samba 3	Rivista	Linux / Samba
Rossini – 10 Ouvertures – Chailly	CD Audio	Classica

Siccome è bene, prima di progettare qualunque Database, preparare uno schema *reale* di quello che la Tabella dovrà contenere, che cosa possono suggerirci queste *righe* di esempio ?

1. I campi *TipoSupp* e *Argom* contengono stringhe ripetitive, cioè molte righe avranno ad esempio in *TipoSupp* la stringa *DVD Musicale*; è uno spreco enorme di spazio.
2. Se scriviamo *manualmente* in *TipoSupp* le varie classificazioni, sarà facile sbagliare e quindi inserire ad esempio per errore *DVD Musocale*; se devo poi selezionare tutti i *DVD Musicali*, quella riga sarà scartata.
3. Come facciamo a ricordare a memoria tutte le classificazioni usate? Se la mediateca è ampia, è quasi impossibile.

Allora come fare ? Semplice, basta creare un'altra Tabella da usare come *classificatore*, del tipo:

TbSupporti **-**> Classificazione dei Media per Tipo di Supporto

Identificatore	Descrizione
1	DVD Film
2	DVD Musicale
3	CD Audio
4	Rivista

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

Identificatore	Descrizione
5	Libro

Così il nostro Archivio apparirà in questa forma :

Desc	TipoSupp	Argom
Pearl Jam - Binaural	3	Rock
Fromm – Avere o Essere ?	5	Filosofia
Harry Potter e la camera dei segreti	1	Fantasy
Peter Gabriel – Growing Up Live	2	Rock
Condividere risorse con Samba 3	4	Linux / Samba
Rossini – 10 Ouvertures – Chailly	3	Classica

I vantaggi di questa organizzazione sono subito evidenti :

1. risparmio di spazio (numeri interi al posto di stringhe);
2. velocità di immissione (devo solo selezionare, nel campo *TipoSupp*, un valore da una Tabella);
3. velocità operativa (un indice su "*TipoSupp*" sarà molto più efficiente);
4. riduzione della possibilità di errore.

Ho dunque eseguito, magari senza saperlo, quella che si chiama una **normalizzazione** del Database. Tutto quanto spiegato vale anche per il campo *Argom*, ed in generale per tutti i campi destinati a contenere informazioni riconducibili ad un insieme ben definito e non troppo ampio. Nella fase operativa vedremo come mettere in *relazione* le due tabelle, e quindi sfruttare al meglio il Db normalizzato. In Appendice troverete lo schema completo del nostro Database di esempio.

TIP

In questo esempio abbiamo usato un campo di tipo *intero* per collegare le informazioni contenute nelle due Tabelle. Questo non è obbligatorio. Ci sono molti casi in cui i campi collegati sono di tipo *stringa*. La regola è che comunque tutti i campi usati nelle relazioni DEVONO essere indicizzati (come vedremo più avanti) e quindi gli *interi* sono più efficienti

Tecnica

Quella illustrata è una relazione tra Tabelle di tipo *uno a molti*, cioè un singolo valore di una riga di *TbSupporti* ("3", equivalente a "CD Audio") appare in molte righe di *TbMedia*. Altri tipi di relazione sono *uno a uno* e *molti a molti*. Comprendere come

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

> funzionano le relazioni tra Tabelle nei Database può sembrare complicato, ma alla fine i risultati sono assai soddisfacenti. Nel nostro caso il campo *TipoSupp* di *TbMedia* fa riferimento alla *chiave primaria* di un'altra Tabella, cioè *TbSupporti*, quindi *TipoSupp* è una **chiave esterna** (**foreign key**) per *TbMedia*. Una delle regole che indicano se un motore di Db può definirsi relazionale o meno riguarda proprio le chiavi esterne. Ne riparleremo più avanti nell'esaminare la gestione dell'integrità referenziale.

Ora descriveremo nel dettaglio la struttura dei nostri archivi. Notate che:

- in ogni tabella esiste un campo di tipo **id** (*identificatore*) di tipo **integer auto_increment** *assegnato come Chiave Primaria;*
- i *nomi* dei campi hanno un prefisso che indica la tabella di appartenenza (ad es. *MDes*, dove M sta per Media e Des per descrizione), salvo quelli che fanno riferimento ad altre Tabelle (*chiavi esterne*) che assumono lo stesso nome della Tabella di appartenenza, e sono indicati in corsivo; questo non è obbligatorio, ma evita di fare confusione nell'uso del linguaggio SQL;
- il *tipo di campo* viene indicato con la sintassi standard SQL; alcuni motori di Db potrebbero avere tipologie leggermente diverse;
- in ogni tabella abbiamo aggiunto un campo di tipo *timestamp*;
- l'ultima colonna indica se al campo è associato un indice; **pk** sta per *Primary Key*, cioè quel campo è la chiave primaria.

2.7.2 Tabella TbMedia -> Archivio Principale

Dunque la Tabella **TbMedia** sarà il nostro Archivio principale. In base alle considerazioni precedenti, la struttura potrebbe essere:

Campo	Tipo di Campo	Commenti	Idx
MId	Integer auto_increment	Identificatore - Chiave primaria	Pk
MDes	Varchar(100)	Descrizione max 100 caratteri	*
SuppId	Integer	Identificatore del Supporto	*
ArgId	Integer	Identificatore dell'Argomento	*
MUbic	Varchar(50)	Ubicazione max 50 caratteri	
MPrezzo	Decimal(14,2)	Prezzo in Euro (max due cifre dec)	
MTs	Timestamp	Timestamp per la tabella	

2.7.3 Tabella TbSupporti -> Tipologie dei Supporti

Questa Tabella associa ad un numero intero la descrizione del Supporto corrispondente. Struttura di **TbSupporti**:

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

Campo	Tipo di Campo	Commenti	Idx
SuppId	Integer auto_increment	Identificatore - Chiave primaria	Pk
SuppDes	Varchar(30)	Descrizione max 30 caratteri	*
SuppTs	Timestamp	Timestamp per la tabella	

2.7.4 Tabella TbArgomenti -> Tipologie degli Argomenti

Questa Tabella associa ad un numero intero la descrizione dell'Argomento corrispondente. Struttura di **TbArgomenti**:

Campo	Tipo di Campo	Commenti	Idx
ArgId	Integer auto_increment	Identificatore - Chiave primaria	Pk
ArgDes	Varchar(30)	Descrizione max 30 caratteri	*
ArgTs	Timestamp	Timestamp per la tabella	

2.7.5 Tabella TbUtenti -> Archivio degli Utenti dei Prestiti

Siccome desideriamo gestire anche eventuali prestiti per la nostra Mediateca, può essere opportuno archiviare le informazioni degli Utenti, secondo questo schema (**TbUtenti**):

Campo	Tipo di Campo	Commenti	Idx
UtId	Integer auto_increment	Identificatore - Chiave primaria	Pk
UtDen	Varchar(50)	Denominazione max 50 caratteri	*
UtVia	Varchar(50)	Indirizzo max 50 caratteri	
UtCit	Varchar(100)	Città max 100 Caratteri	
UtTel	Varchar(20)	Telefono max 20 Caratteri	
UtTs	Timestamp	Timestamp per la tabella	
UtDNas	Data	Data di Nascita	
UtImg	Immagine	Foto dell'Utente	
UtCodFis	Varchar(16)	Codice Fiscale	
UtSesso	Char(1)	Sesso (M o F)	
UtTessera	Integer(1)	Tesserato (valore Logico, 0 o 1)	

Non tutti i campi sarebbero strettamente necessari, ma alcuni sono indispensabili per comprendere meglio l'uso avanzato dei Formulari.

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

2.7.6 Tabella TbPrestiti -> Archivio dei prestiti e delle restituzioni

Questa Tabella contiene l'elenco cronologico dei prestiti e delle restituzioni, collegato con la Tabella dei Media e con la Tabella degli Utenti. Struttura di **TbPrestiti**:

Campo	Tipo di Campo	Commenti	Idx
PreId	Integer auto_increment	Identificatore - Chiave primaria	Pk
PreData	Date	Data del prestito	*
UtId	Integer	Identificativo Utente (TbUtenti)	*
MId	Integer	Identificativo Media (TbMedia)	*
PreDataR	Date	Data di restituzione	
PreTs	Timestamp	Timestamp per la tabella	

2.8 Integrità Referenziale

In questo paragrafo cercheremo di approfondire un aspetto dei motori di Db assai importante, che ogni sviluppatore (od anche un semplice utente) dovrebbe conoscere bene per evitare che il proprio Database incorra in seri problemi nella gestione in ambienti di produzione: l'*integrità referenziale*.

tbmedia.MId	tbmedia.MDes	tbmedia.ArgId	tbargomenti.ArgId	tbargomenti.ArgDes
2	Joss Stone - The Soul Session	1	1	Rock
3	Bad Boys II	18	18	Avventura
4	KDE 3.2 - Novità Major Release	16	16	Linux
5	Montalbano - Il Ladro di merendina	5	5	Gialli
6	Benni - La compagnia dei celestini	17	17	Romanzo
7	Guccini - Ritratti	3	3	Pop
454	Il Signore degli Anelli	15	15	Fantasy
455	Montalbano - La forma dell'acqua	5	5	Gialli
456	Pearl Jam - Ten	1	1	Rock
457	Cornwell - L'Ultimo Distretto	5	5	Gialli

Figura 2.8.1: Relazione tra TbMedia e TbArgomenti

Per chiarire i termini della questione, diamo un'occhiata al nostro Db di test (*Mediateca*). La Tabella *TbMedia*, che contiene l'archivio dei nostri supporti, per la classificazione per argomento fa riferimento alla Tabella *TbArgomenti*, attraverso il campo *ArgId*.

Consideriamo, per semplicità, il nostro Archivio con alcuni dati inseriti, come nella figura precedente: i campi di *TbMedia* sono sulla sinistra, quelli di *TbArgomenti* sulla destra. Quella che vedete in *gergo* Db si chiama ***query***, e viene anche indicata nella terminologia di OpenOffice come ***ricerca***.

Nel nostro esempio se si specifica il valore “5” nel Campo *ArgId* di *TbMedia*, la riga sarà associata alla voce “Gialli” che compare in *TbArgomenti*. *Ora provate a pensare che per errore*

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

venga cancellato il Record con ArgId "5" in TbArgomenti. In questo caso tutti i Record che hanno come riferimento "Gialli" in TbMedia non avrebbero più un argomento associato, con conseguenze imprevedibili sulla coerenza dei dati di tutto il Db. Infatti *ArgId* è una **chiave esterna** (**foreign key**) per la Tabella *TbMedia*, collegata a *TbArgomenti*. Quindi *TbArgomenti* si indica anche come tabella **parent** (*genitore*) e *TbMedia* come tabella **child** (*figlio*); *questo sta anche a significare che per ogni chiave ArgId "parent" in TbArgomenti, possono esistere più chiavi "child" in TbMedia (cioè molti supporti possono essere classificati come "Gialli"), tipica relazione uno a molti*.

Sarebbe, inoltre, MOLTO opportuno che nel campo *ArgId* di *TbMedia* possano essere immessi solo valori GIA' PRESENTI in *TbArgomenti*, cioè non dovrebbe essere consentito inserire argomenti inesistenti, o, se volete, figli senza genitori.

Molti motori di Db hanno a disposizione un meccanismo di controllo che impedisce la cancellazione o la modifica delle chiavi esterne, o comunque permette di stabilire regole precise per la gestione di queste eventualità. Questi motori si occupano perciò di **preservare l'integrità referenziale del Database**.

2.9 Server di Database

A questo punto dovrebbe essere chiaro che cosa intendiamo quando si parla di *Database Relazionale*. Forse è meno chiaro come sfruttare i concetti che abbiamo appena illustrato, ovvero quale prodotto usare, casomai volessimo creare ad esempio la nostra Mediateca. In realtà quello dei Database è un mondo estremamente vasto e vario: esistono software di tutte le taglie e per tutte le tasche che, più o meno, possono ricondursi alla categoria oggetto del nostro interesse. Una prima classificazione può essere fatta in base alla "vocazione" del prodotto: **server** o **desktop**.

In generale un **Database Server** è un programma che rimane in "ascolto" su una porta del PC, pronto ad eseguire i "comandi" impartiti da altre applicazioni. Quindi, di solito, se un Server è in funzione sul PC potreste tranquillamente non notare nulla di strano, salvo magari qualche icona nella tray bar. Per usare queste tipologie di prodotti è necessario un programma *client* che permetta di interagire col *server*. Tutti i Server di Db sono dotati di una serie di tool *client,* quasi sempre a linea di comando. Alcuni si servono anche di apposite interfacce grafiche che semplificano la vita degli utenti. Sono disponibili molti prodotti commerciali di questo tipo, alcuni davvero ottimi, ma dal costo non sempre accessibile: **Oracle**, **Sybase**, **Microsoft SQL Server**. Sul versante *Open Source* le notizie sono, per una volta, molto buone: esistono almeno due prodotti che hanno poco da invidiare alle soluzioni a pagamento, **MySql** e **PostgreSql**. Recentemente si sta affermando anche un altro prodotto Open Source chiamato **Firebird**, che è una "rielaborazione" del vecchio motore *Interbase* di Borland.

Un **Database per Desktop** in genere ha un approccio diverso: per prima cosa quasi sempre include un suo motore di Db, ma spesso può collegarsi anche anche ad altri prodotti;

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

poi dispone della possibilità di sviluppare facilmente *interfacce* verso i dati (maschere, ricerche e selezioni, report etc.); infine a volte include un vero e proprio sistema di sviluppo di applicazioni, completo di linguaggio di alto livello. Esempi di questa tipologia di software possono essere Microsoft **Access**, **Filemaker**, **Foxpro**, **Paradox** ma anche il nostro **OpenOffice Base**. Questi prodotti possono anche essere usati come *ponte* verso altri motori di Db, compresi molti Server, ed è questo che vedremo nel seguito con riferimento appunto ad OpenOffice.

2.10 Modalità di accesso ai Dati

Un Database Server deve prevedere delle interfacce di accesso ai propri dati, in modo che applicazioni di terze parti possano *interagire* col server stesso. Queste interfacce sono di tipo anche molto diverso tra loro. Abbiamo già detto dei programmi a linea di comando. Esistono poi appositi *moduli*, sviluppati in vari linguaggi di programmazione, che permettono alle applicazioni l'accesso e la manipolazione dei dati. Un esempio è il Modulo di *PHP* per *MySql*, ma ne troviamo decine per gli ambienti di programmazione più diversi.

Sono inoltre disponibili dei protocolli standard, sviluppati sul concetto di *Driver* (un po' come per le Schede Grafiche). In questo momento esistono due principali classi di driver: **ODBC** e **JDBC**.

ODBC (Open DataBase Connectivity) è lo standard più usato: esistono versioni per Windows e Linux e le prestazioni sono soddisfacenti. La presenza di un *Driver ODBC* per un Server garantisce l'accesso ai Dati a qualunque applicazione sia conforme allo standard (a cominciare da OpenOffice per finire ai vari moduli di Microsoft Office compreso Access).

JDBC (Java DataBase Connectivity) è più o meno la stessa cosa, rivolta però ad ambienti di sviluppo Java. Questo non toglie che driver JBDC siano utilizzabili anche da altre applicazioni, come OOo.

Esistono ancora altre modalità di interconnessione specifiche per ambienti Windows (come *ADO Db*) di cui però mi sembra inutile ora occuparsi.

2.11 Componenti di un Database

Abbiamo già descritto due elementi essenziali (anzi direi indispensabili) di ogni Database: le *Tabelle* e gli *Indici*. A questi ogni motore di Db aggiunge caratteristiche proprie che permettono una migliore gestione delle Basi di Dati. Parliamo di *Viste*, *Triggers*, *Stored Procedure*. Non è il caso qui di approfondire molto questi argomenti, ma forse è il caso di parlarne brevemente.

2.11.1 View (Vista)

Una ***Vista*** (*view*) è una particolare struttura che raccoglie ed organizza campi provenienti da una oppure più tabelle messe in relazione tra di loro. Si tratta, come vedremo,

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

sostanzialmente di una query di selezione archiviata sul Server, che però può essere usata come una Tabella reale (quindi a volte è anche aggiornabile). Una *Vista* è particolarmente utile quando è necessario organizzare i dati secondo precise regole, lasciando fare tutto il lavoro al Server (così da ottenere velocità e precisione). A titolo di esempio si potrebbe creare una *vista* che elenca tutti gli elementi di *TbMedia* completi di *Argomento* e *Supporto* ordinata per Descrizione; in questo caso quindi, tre tabelle, collegate tra loro da relazioni, che vengono raggruppate in una sola entità. Il risultato sarebbe più o meno:

MId	MDes	SuppDes	ArgDes
459	Afterhours - Ballate per piccole iene	CD Audio	Rock
3	Bad Boys II	DVD Video	Avventura
6	Benni - La compagnia dei celestini	Libro	Romanzo
457	Cornwell - L'Ultimo Distretto	Libro	Giallo
458	De Gregori - Pezzi	CD Audio	Rock
7	Guccini - Ritratti	CD Audio	Pop
454	Il Signore degli Anelli	DVD Video	Fantasy
2	Joss Stone - The Soul Session	CD Audio	Pop
4	KDE 3.4 - Novità Major Release	Rivista	Linux
463	Ligabue - Roma Stadio Olimpico	CD Audio	Rock
462	Marillion - Marbles on the Roads	DVD Video	Rock
5	Montalbano - Il Ladro di merendine	Libro	Giallo
455	Montalbano - La forma dell'acqua	Libro	Giallo
456	Pearl Jam - Ten	CD Audio	Rock
460	Springsteen - Devils & Dust	DVD Video	Rock

Figura 2.11.1: Una Vista che comprende tre Tabella

2.11.2 Trigger

Un ***Trigger*** è un'azione eseguita al verificarsi di una condizione riguardante una Tabella oppure un singolo Campo. Il *Trigger* è comunque un oggetto del Database, quindi deve possedere un nome univoco. Di solito, nella definizione, si indica, oltre all'oggetto a cui si applica, l'azione che attiva il Modificatore (***insert***, ***delete*** o ***update***) ed il momento di esecuzione (***before*** oppure ***after***). Così, ad esempio, se nella mia tabella ho un campo "*Prezzo Iva esclusa*" ed uno "*Prezzo Iva inclusa*", posso creare un *Trigger* che, ad ogni modifica del primo campo calcola immediatamente il secondo, senza ulteriori interventi da parte dell'utente.

2.11.3 Stored Procedure

Una ***Stored Procedure*** è una subroutine scritta in un linguaggio di programmazione (generalmente SQL) ed archiviata sul Server, che può essere richiamata in modo semplice da un Client. In generale una Stored Procedure può accettare parametri, ed il vantaggio (ovvio) è che tutte le elaborazioni avvengono "lato Server" senza appesantire il Client ed il traffico di rete. In alcuni casi le *Stored Procedure* sono particolarmente utili: ad esempio quando programmi client scritti in linguaggi diversi e funzionanti su piattaforme diverse devono

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

eseguire le stesse operazioni; in questo caso possono limitarsi ad invocare la Stored Procedure disponibile sul Server.

2.12 Parliamo un po' di Structured Query Language

Il Linguaggio *SQL*, nelle sue caratteristiche di base, è uno Standard ANSI/ISO. Come spesso accade, però, ogni azienda fornitrice di Db server ritiene opportuno aggiungere *estensioni* allo standard, in base alle caratteristiche del proprio prodotto. Come è facile immaginare, queste *estensioni*, non avendo una base concordata, spesso sono incompatibili tra loro, quindi un comando *SQL* valido, ad esempio, per *Sybase* non funzionerà con *MySql*, per la gioia di tutti gli sviluppatori. Quando si studia un motore di Db è quindi necessario capire bene quali sono le particolarità della sintassi *SQL* che si va ad utilizzare. In ogni caso una base comune esiste, quindi quanto scriverò nel seguito è in generale valido per tutti i motori di Db.

Come tutti i linguaggi di programmazione, anche *SQL* ha una serie di regole per quanto riguarda la sintassi, che sarebbe inutile descrivere qui.

SQL è stato inizialmente progettato per eseguire "ordini" digitati da una linea di comando. Quindi una istruzione *SQL* esegue *immediatamente* una operazione sul Database selezionato. Possiamo, per comodità, suddividere istruzioni in gruppi, a seconda delle scopo a cui sono dedicate, e quindi :

1. Amministrazione del Database
2. Definizione dei dati
3. Manipolazione dei dati
4. Gestione delle repliche
5. Gestione delle transazioni

Ai nostri fini, considerato che la parte amministrativa può essere benissimo gestita con uno qualsiasi degli strumenti grafici disponibili, e che le repliche e le transazioni certo esulano dallo scopo di questo manuale, converrà discutere solo della definizione e manipolazione dei dati.

Le istruzioni per la **definizione dei dati** (meglio sarebbe dire per la definizione della *struttura* dei dati) comprendono tutti gli strumenti adatti a ***creare***, ***modificare*** e ***cancellare tabelle***, ***indici*** e ***chiavi esterne***. Anche queste operazioni, come abbiamo già visto, possono essere eseguite tramite interfacce grafiche, quindi l'uso di questi istruzioni dirette è raro, almeno nel nostro ambito. A titolo di esempio, vi riporto l'istruzione SQL che crea la tabella *TbMedia* del nostro Db, completa di indici e di chiave esterna in *MySql*:

```
CREATE TABLE `tbmedia` (

  `MId` int(11) unsigned NOT NULL auto_increment,
  `MDes` varchar(100) NOT NULL default '',
  `SuppId` int(10) unsigned NOT NULL default '0',
```


© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

```
  `ArgId` int(10) unsigned NOT NULL default '0',
  `MUbic` varchar(100) default '',
  `MPrezzo` decimal(14,2) unsigned NOT NULL default '0.00',
  `MTs` timestamp(14) NOT NULL,

  PRIMARY KEY  (`Mid`),

  KEY `Supp` (`SuppId`),
  KEY `Des` (`MDes`),
  KEY `Arg` (`ArgId`),

  CONSTRAINT `tbmedia_ibfk_1` FOREIGN KEY (`ArgId`) REFERENCES `tbargomenti`
    (`ArgId`)

  CONSTRAINT `tbmedia_ibfk_2` FOREIGN KEY (`SuppId`) REFERENCES `tbsupporti`
    (`SuppId`)

) TYPE=InnoDB;
```

A titolo di esempio, vi riporto anche la stessa istruzione, ma relativa a **PostgreSQL**:

```
CREATE TABLE "TbMedia" (

  "MId" int4 NOT NULL DEFAULT nextval('public."TbMedia_MId_seq"'::text),
  "MDes" varchar(100),
  "SuppId" int4 NOT NULL DEFAULT 0,
  "ArgId" int4 NOT NULL DEFAULT 0,
  "MUbic" varchar(50),
  "MPrezzo" numeric(14,2) NOT NULL DEFAULT 0,
  "MTs" timestamp,

  CONSTRAINT "Pk" PRIMARY KEY ("MId"),

  CONSTRAINT "ExtKArg" FOREIGN KEY ("ArgId") REFERENCES "TbArgomenti"
    ("ArgId") ON UPDATE RESTRICT ON DELETE RESTRICT,

  CONSTRAINT "ExtKSupp" FOREIGN KEY ("SuppId") REFERENCES "TbSupporti"
    ("SuppId") ON UPDATE RESTRICT ON DELETE RESTRICT
)
WITHOUT OIDS;
```

Le differenze non sono poi molte, ma quanto basta per rendere ***incompatibili*** i due comandi. Come si può notare, un comando (statement) *SQL* è, di solito, composto da una parola chiave dal nome sufficientemente esplicativo, seguita da uno o più parametri. L'uso delle parentesi serve a "passare" informazioni, separate da virgole, al motore di Db (come se fossero i parametri di una chiamata di funzione). Le stringhe sono contenute in *virgolette singole o doppie (dette anche apici)*. L'istruzione si chiude, di solito, con un punto e virgola (;).

Così, ad esempio, se volessi modificare il tipo di dati del campo *MUbic* da *varchar(100)* a *varchar(50),* l'istruzione sarebbe, in *MySql*:

```
ALTER TABLE `mediateca`.`tbmedia` MODIFY COLUMN `MUbic` VARCHAR(50);
```

In questo caso è indicato esplicitamente il Database (*Mediateca*) che contiene la tabella. Gli statement per la definizione dei dati non sono molti, ma in alcuni casi possono raggiungere una notevole complessità. **Se possibile, usate l'interfaccia grafica.**

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

Per la manipolazione dei dati, invece, i comandi principali sono soltanto quattro e precisamente:

```
DELETE
INSERT
UPDATE
SELECT
```

Non vi tedierò con la sintassi precisa di ogni istruzione, e dal nome si intuisce facilmente lo scopo a cui sono dedicate. Voglio però farvi notare che di tutte, la sola che ritorna un risultato è ***SELECT***.

DELETE, ***INSERT*** e ***UPDATE*** servono rispettivamente a cancellare, aggiungere, modificare i dati presenti ne Db. Così ad esempio l'istruzione di *MySql*:

```
DELETE FROM tbmedia WHERE ArgId=98;
```

cancella dalla tabella TbMedia tutti i record con ArgId uguale a 98. Notate anche che, *se non ci sono ambiguità e non avete usato caratteri speciali, gli apici possono essere omessi*. Io, per comodità, uso le maiuscole per gli elementi della sintassi, e le minuscole per gli argomenti ed i parametri, ma ognuno può fare come vuole. Grande importanza in *SQL* assume la clausola ***WHERE*** ("*DOVE*"), che permette di limitare l'ambito di applicazione dell'istruzione ad un sottoinsieme di record che rispettano determinate condizioni. Se eseguiamo lo statement precedente, come vedremo, dalla linea di comando, *MySql* risponderà con qualcosa del tipo "*XX Rows Affected*", cioè col numero di Record cancellati. *Quindi in generale **DELETE**, **INSERT**, ed **UPDATE** modificano il contenuto del Db, e ritornano solo il numero di record modificati*.

SELECT, invece, è tutta un'altra storia. ***SELECT*** serve a "selezionare" una parte di record ed a mostrare solo le informazioni che ci servono. Ad esempio :

```
SELECT  MDes, ArgId, MId FROM tbmedia
WHERE ArgId=5;
```

ha come risultato :

MDes	ArgId	MId
Montalbano - Il Ladro di merendine	5	5
Montalbano - La forma dell'acqua	5	455
Cornwell - L'Ultimo Distretto	5	457

Figura 2.12.1Risultato della Query

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99

abbiamo cioè chiesto di selezionare (***SELECT***) i campi *Mdes*, *ArgId*, *MId* dalla tabella *TbMedia*, ma solo quelli che hanno *ArgId uguale a 5*. Da quanto detto, a seconda del contesto, ***SELECT*** può "ritornare" (cioè appunto selezionare) anche migliaia di record, ed è lo strumento ideale per interrogare il Db. L'esempio che abbiamo fatto è, in sostanza, banale, perché ***SELECT*** ha una sintassi abbastanza complessa; infatti, dal manuale di *MySql* :

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr, ...
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name']
    [FROM table_references
      [WHERE where_definition]
      [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_definition]
      [ORDER BY {col_name | expr | position}
        [ASC | DESC] , ...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset}]
      [PROCEDURE procedure_name(argument_list)]
      [FOR UPDATE | LOCK IN SHARE MODE]]
```

e da quello di *PostgreSQL*:

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ AS output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start ]
    [ FOR UPDATE [ OF table_name [, ...] ] ]
```

quindi, ad esempio, possiamo specificare più tabelle legate da relazioni, raggruppamenti, ordinamenti, limiti e molto altro. Vi consiglio perciò di studiare l'argomento in modo approfondito, se volete divertirvi con i server di Database.

Tecnica

In sostanza una istruzione ***SELECT*** non è che una "richiesta" al motore di Db, e quindi in inglese può indicarsi anche col termine ***QUERY***. Più precisamente una ***SELECT*** viene indicata come **QUERY DI SELEZIONE**. Per estensione, ***UPDATE***, ***DELETE*** ed ***INSERT*** vengono indicate come **QUERY DI COMANDO**. Per inciso, è questa la terminologia adottata da *MS Access*. *OpenOffice* invece indica una ***SELECT*** col termine di **RICERCA**, ed è possibile con lo stesso strumento eseguire anche *QUERY DI COMANDO*.

© 2005 Filippo Cerulo - Soft.Com Sas – Documento con Licenza Creative Commons - Rev 0.99