I.F.T.S. 2021 - 2022

DISPENSE DIDATTICHE

Modulo n°: 11

Titolo modulo: PROGETTAZIONE E GESTIONE DATABASE

Docente: Claudio Rossi

Coordinatrice del Corso: Sara Forlivesi

Fondazione En.A.I.P. S. Zavatta Rimini
Viale Valturio, 4 47923 Rimini
Tel. 0541.367100 – fax. 0541.784001
www.enaiprimini.org; e-mail: info@enaiprimini.org

DBMS e database

Un database è un archivio strutturato di dati, i dati contenuti nel database sono strutturati e collegati fra loro secondo un modello logico (il più diffuso è quello relazionale, nel seguito del documento faremo riferimento sempre a questo modello) in modo da consentire una gestione efficiente dei dati ed il supporto a linguaggi strutturati (query languages) per la gestione dei dati.

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).

Tabelle

Una tabella di un database è un contenitore di dati fortemente tipizzati e soggetti a vincoli di integrità ed è costituita da due insiemi:

- campi (o attributi), che definiscono la struttura della tabella
- record (o tuple), che costituiscono il contenuto della tabella

Ogni campo presente in una tabella specifica una serie di informazioni strutturali che consentono al database di gestire al meglio i dati che sono presenti o verranno inseriti, le principali informazioni sono:

- nome del campo
- tipo di dato
- obbligatorietà del dato

Il nome del campo è solamente un'etichetta che serve agli operatori umani per identificare ed interpretare correttamente i dati presenti nei record, è possibile in qualsiasi momento modificare questo attributo senza conseguenze per l'integrità del database.

Il tipo di dato è un attributo fondamentale sia per l'integrità del database che per le prestazioni del medesimo, scelte sbagliate di questo attributo possono avere gravi conseguenze durante l'utilizzo.

Il tipo di dato vincola i dati che possono essere inseriti nel campo (ad esempio non è possibile inserire il valore "ciao" in un campo di tipo numerico) eventuali modifiche al tipo di dato possono creare problemi all'integrità del database perché questa operazione richiede la conversione di tutti i valori inseriti.

L'obbligatorietà di un dato è un criterio concettualmente semplice ma fondamentale per il buon funzionamento di un database relazionale: garantisce che i valori strategici siano sempre presenti.

Chiave primaria

Uno degli attributi più importanti per una tabella è la chiave primaria: si definisce chiave primaria un insieme di campi che consente di individuare univocamente un record all'interno della tabella, per ogni tabella può essere definita al più una chiave primaria.

I campi che fanno parte della chiave primaria sono soggetti ad alcuni vincoli:

- vincolo di obbligatorietà , non possono mai essere lasciati vuoti
- vincolo di unicità, non possono mai contenere valori già presenti nello stesso campo di un altro record

Esempi tipici di chiave primaria sono il codice fiscale per le persone e la partita iva per le aziende.

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 (anche se 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 quindi conviene 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.

Chiave importata

Un attributo importante da non confondere con la chiave primaria (per via della somiglianza dei nomi) è la chiave importata.

Una chiave importata (o chiave esterna) è un insieme di campi di una tabella che referenziano un insieme di campi (composto dallo stesso numero di elementi) presente all'interno di un'altra tabella: questo definisce una *relazione* fra le tabelle.

Le relazioni fra tabelle possono essere di tre tipi:

- 1:1, quando i campi in corrispondenza sono univoci in entrambe le tabelle
- 1:n, quando i campi di una tabella (detta *padre* o *testa*) sono univoci mentre non lo sono quelli dell'altra (detta *figlia* o *corpo*)
- n:m, quando i campi di entrambe le tabelle non sono univoci

Indici e vincoli

Per ogni tabella è possibile indicare uno o più indici ed uno o più vincoli.

Gli indici servono principalmente per motivi prestazionali: forniscono un sistema veloce per accedere ad un record a partire dal valore di uno o più campi.

Esistono due tipi di indice: *indice cluster* e *indice non cluster*.

Gli indici cluster ordinano e archiviano fisicamente i record della tabella in base ai valori di chiave, ovvero alle colonne incluse nella definizione dell'indice.

Per ogni tabella è disponibile un solo indice cluster, poiché alle righe di dati è possibile applicare un solo tipo di ordinamento.

Gli indici non cluster presentano una struttura distinta dai record, un indice non cluster contiene i valori del campo (o dei campi) che lo compone, ciascuno dei quali dispone di un puntatore alla riga di dati che contiene il valore di chiave.

I vincoli consentono di limitare gli inserimenti di dati errati specificando le regole di validità per ogni campo, in base al tipo di dato specificato per il campo cambiano i vincoli applicabili.

Un vincolo particolarmente importante è il *not null* che significa che il campo non può essere lasciato vuoto, questo si applica a qualsiasi tipo di dato.

Null è un valore speciale, diverso da stringa vuota, zero o qualsiasi altro valore, pertanto è possibile (ad esempio) inserire una stringa vuota in un campo che possiede il vincolo non null.

SQL

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.

Per operare sulla struttura delle tabelle e sui dati contenuti al loro interno si usano le query, tramite una query SQL è possibile effettuare qualsiasi operazione sul database, sulle tabelle e sui dati.

Esistono due tipi fondamentali di query:

- query di selezione, consentono di interrogare il database per cercare dei dati
- query di comando, consentono di eseguire qualsiasi tipo di operazione sul database e sui dati (creazione, modifica, inserimento, cancellazione)

Query di selezione

Il linguaggio SQL permette di programmare in più DBMS (Access, SQL server, Oracle, ecc.) allo stesso modo, alcuni DBMS usano delle estensioni del linguaggio che aggiungono potenzialità alle istruzioni di base.

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.

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.

Esempio

Data la tabella Clienti:=<Cognome, Nome, Telefono, Citta> la query che segue restituisce tutti i dati dei clienti il cui cognome è Bianchi, ordinandoli per nome:

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

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 oggetto della query.

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.

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

Select

Specifica i campi da visualizzare come risultato dell'interrogazione, utilizzando il simbolo * si visualizzano 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.

From

Specifica le tabelle o le query origine dei dati.

Distinct

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

Esempio

Clienti			
Cognome	Nome	Telefono	Citta[2]
Bianchi	Mario	0541254825	Rimini
Bianchi	Ettore	0514875978	Bologna
Casadei	Mario	0554578659	Firenze

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

Where

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

Esempio

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

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

---

[2] in SQL non è consigliabile usare lettere accentate né nomi contenenti spazi bianchi per identificare tabelle o campi.

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 insieme

È possibile estrarre dalle tabelle anche valori d'insieme, 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.

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

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

questa query è sbagliata!!!

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[3]
Bianchi	Mario	1.000.000
Bianchi	Ettore	5.000.000
Casadei	Mario	3.000.000

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).

[3] 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.

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)[4] >= 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="21/07/2010");
```

L'esecuzione della query proposta restituisce i nomi dei clienti che hanno una prenotazione per il 21/07/2010.

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
```

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

```
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* ed il carattere jolly ‘%’ che indica una sequenza di caratteri di qualsiasi lunghezza (anche nulla).

Esempio

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

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

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.

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);
```

Union

La clausola UNION consente di unire i risultati di due query di selezione all’interno di un’unica tabella.

Esempio

```
SELECT NOME, CODICE_FISCALE
```

FROM CLIENTI

UNION

SELECT RAGIONE_SOCIALE, PARTITA_IVA

FROM FORNITORI

La clausola UNION richiede che entrambe le query restituiscano lo stesso numero di campi e che i tipi di dato relativi ai campi della seconda query siano compatibili con i tipi di dato relativi ai campi della prima query. Non è necessario che i tipi di dato siano uguali perché alcuni tipi possono contenerne altri (es: il tipo "numero con la virgola" può contenere anche dati di tipo "numero intero).

Se la seconda query produce alcuni record identici ad alcuni già prodotti dalla prima, questi record non vengono accodati.

La clausola UNION prevede la variante UNION ALL che consente di accodare anche eventuali record prodotti dalla seconda query e identici ad alcuni già prodotti dalla prima query.

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	21/08/2018
B01	28	22/08/2018
B01	05 25	04/09/2018

L'operazione di join darà come risultato:

Codice	Cognome	Nome	Cod_cli	Camera	Data
A01	Bianchi	Mario	A01	15	21/08/2018
A01	Bianchi	Mario	B01	05 25	04/09/2018
A01	Bianchi	Mario	B01	28	22/08/2018

A02	Bianchi	Ettore	A01	15	21/08/2018
A02	Bianchi	Ettore	B01	05 25	04/09/2018
A02	Bianchi	Ettore	B01	28	22/08/2018
B01	Casadei	Mario	A01	15	21/08/2018
B01	Casadei	Mario	B01	05 25	04/09/2018
B01	Casadei	Mario	B01	28	22/08/2018

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	21/08/2018
B01	Casadei	Mario	B01	28	22/08/2018
B01	Casadei	Mario	B01	05 25	04/09/2018

La query dell'esempio precedente può anche essere scritta utilizzando la seguente sintassi:

```
SELECT COGNOME, NOME, CAMERA, DATA
FROM CLIENTI
INNER JOIN PRENOT ON CLIENTI.CODICE=PRENOT.COD_CLI;
```

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 chiamato rispettivamente *Cod_cli* nella tabella *Prenot* e *Codice* nella tabella *Clienti* (il nome non è rilevante, 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
INNER JOIN PRENOT ON CLIENTI.CODICE=PRENOT.COD_CLI
WHERE NOME="MARIO"
AND COGNOME="CASADEI";
```

Restituisce tutti i dati relativi alle prenotazioni del signor Mario Casadei.

Autojoin

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

Esempio

Data la seguenti tabella:

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

Il modo più semplice per avere la lista delle persone che hanno il salario uguale alle spese 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
FROM CLIENTI PRIMA, CLIENTI SECONDA
WHERE PRIMA.CODICE = SECONDA.CODICE
AND PRIMA.SALARIO=SECONDA.SPESE;
```

- selezione dei campi del risultato[5]
- attribuzione degli Alias
- selezione sul prodotto cartesiano
- selezione del parametro cercato

Outer join

In alcuni casi è necessario mantenere tutti i dati presenti in una delle tabelle unite tramite Join, anche quando i record non hanno corrispondenza nell'altra tabella.

Per ottenere questo risultato si utilizza l'operatore di Outer Join, questo consente di specificare che una delle tabelle (o entrambe) dovrà essere restituita interamente.

Esempio

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

Prenot		
Cod_cli	Camera	Data
A01	15	21/08/2018
B01	28	22/08/2018
B01	12	04/09/2018

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

C01	35	04/09/2018

Unendo le tabelle con l'operatore Inner Join, nel risultato non comparirebbe né il cliente A02 (Bianchi Ettore) perché non ha effettuato prenotazioni, né la prenotazione C01 perché manca il cliente che l'ha effettuata.

Utilizzando Left Outer Join, Right Outer Join o Full Outer Join è possibile decidere quali record devono comparire nel risultato anche se la loro relazione risulta mancante.

```
SELECT *
FROM CLIENTI
LEFT OUTER JOIN PRENOT ON CLIENTI.CODICE=PRENOT.COD_CLI;
```

Restituisce tutti i clienti e le sole prenotazioni che hanno un record corrispondente nella tabella Clienti:

Risultato					
Codice	Cognome	Nome	Cod_cli	Camera	Data
A01	Bianchi	Mario	A01	15	21/08/2018
A02	Bianchi	Ettore	NULL	NULL	NULL
B01	Casadei	Mario	B01	28	22/08/2018
B01	Casadei	Mario	B01	12	04/09/2018

```
SELECT *
FROM CLIENTI
RIGHT OUTER JOIN PRENOT ON CLIENTI.CODICE=PRENOT.COD_CLI;
```

Restituisce tutte le prenotazioni e i soli clienti che hanno un record corrispondente nella tabella Prenot:

Risultato					
Codice	Cognome	Nome	Cod_cli	Camera	Data
A01	Bianchi	Mario	A01	15	21/08/2018
B01	Casadei	Mario	B01	28	22/08/2018
B01	Casadei	Mario	B01	12	04/09/2018
NULL	NULL	NULL	C01	35	04/09/2018

```
SELECT *
FROM CLIENTI
FULL OUTER JOIN PRENOT ON CLIENTI.CODICE=PRENOT.COD_CLI;
```

Restituisce tutte le prenotazioni e tutti i clienti:

Risultato					
Codice	Cognome	Nome	Cod_cli	Camera	Data
A01	Bianchi	Mario	A01	15	21/08/2018
A02	Bianchi	Ettore	NULL	NULL	NULL
B01	Casadei	Mario	B01	28	22/08/2018
B01	Casadei	Mario	B01	12	04/09/2018
NULL	NULL	NULL	C01	35	04/09/2018

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 spezzandole in più query semplici.

Esempio

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

Prenot		
Cod_cli	Camera	Data
A01	15	21/08/2018
B01	28	22/08/2018
B01	05 25[6]	04/09/2018

A:

```
SELECT COD_CLI
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:

```
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:

---

[6] 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 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.

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.

Query di creazione

Le query di creazione servono a creare le tabelle definendone la struttura.

Esempio[7]

```
CREATE TABLE nuova
    (
    nome NVARCHAR(50) NULL,
    cognome NVARCHAR(50) NULL,
    dataNascita DATE NULL,
    altezza FLOAT NULL
    )
```

Crea una nuova tabella di nome *nuova* contente solamente i campi *nome* di tipo stringa con lunghezza massima di 50 caratteri*, cognome* di tipo stringa con lunghezza massima di 50 caratteri*, dataNascita* di tipo data, *altezza* di tipo numero decimale.

Query di inserimento

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

[7] La query mostrata utilizza la sintassi SQL Server, altri DBMS utilizzano sintassi diverse per specificare i tipi di dato

È possibile inserire dati copiandoli da una tabella esistente o indicandoli direttamente nella query.

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.

Esempio

```
INSERT INTO NUOVA ( CODICE, COGNOME, TELEFONO )
VALUES (11, "Rossi", "12345678");
```

Aggiunge alla tabella *nuova* un record nei quale il campo *codice* vale 11, il campo *cognome* vale "Rossi" e il campo *telefono* vale "12345678".

NB: in entrambi gli esempi 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 in *Ettore* il nome di tutti i clienti il cui cognome è *Bianchi*.

Query di eliminazione record

Le query di eliminazione record servono ad eliminare da una tabella tutti i dati che corrispondono ad un determinato criterio.

Esempio

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

Cancella tutti i clienti presenti nella tabella il cui cognome è *Bianchi*.

Query di eliminazione tabelle

In maniera analoga alle query di eliminazione record, le query di eliminazione tabella consentono di distruggere interamente (vengono eliminati sia dati che la struttura) una tabella.

Esempio

```
DROP TABLE CLIENTI;
```

Cancella la tabella Clienti.

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

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
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

Cliente	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

Chiavi primarie efficienti

Le chiavi primarie dovrebbero sempre essere compatte, ossia costituite da un solo campo di tipo numerico intero

La compattezza delle chiavi consente di avere database più piccoli (i dati da scrivere nelle chiavi importate richiedono pochi byte) ma soprattutto di avere join molto efficienti dato che più piccolo è il tipo di dato utilizzato nei campi in join, meno risorse (memoria e capacità di calcolo della CPU) sono necessarie per effettuare l'operazione.

Tutti i DBMS forniscono a tale scopo un tipo di dato detto *enumerativo,* ossia un numero intero che viene automaticamente incrementato all'inserimento di un nuovo record in modo da fornire una comoda chiave primaria generata in maniera automatica.

Sql Server

Sql Server è un DBMS commerciale prodotto da Microsoft, ne esistono diverse versioni con prezzi variabili fra gratuito e decine di migliaia di euro.

SQL Server è server based, questo significa che il DBMS è un servizio che rimane in attesa di comandi e svolge in autonomia le operazioni di manutenzione programmate.

I dati di ogni database vengono archiviati in uno o più file, questo consente di avere database di dimensioni superiori a quelle dei singoli hard disk del server.

Management studio

Sql server può essere amministrato tramite la shell testuale del sistema operativo, ma è più comodo utilizzare un sistema grafico che semplifichi le operazioni e che consenta di amministrare il database senza dover necessariamente operare direttamente dal computer sul quale è installato il servizio.

Il sistema tipicamente utilizzato per l'amministrazione del DBMS è *Management Studio*, questo software consente di effettuare tutte le operazioni sul database (dall'esecuzione di query di selezione all'amministrazione degli utenti).

Management Studio può essere utilizzato indifferentemente per amministrare database locali o remoti (compatibilmente con le impostazioni di sicurezza del server).

Gestione degli utenti

Sql server può gestire la sicurezza a livello di utente in due diversi modi: utilizzando la gestione proprietaria degli utenti o integrandosi con la gestione degli utenti fornita dal sistema operativo.

La gestione della sicurezza consente di specificare le operazioni ammissibili per ogni utente o per ogni gruppo di utenti.

Backup e ripristino di database

Sql server supporta diverse modalità di backup dei database, in tutti i casi è possibile effettuare backup completi o incrementali (che memorizzano solamente le differenze rispetto all'ultimo backup).

I backup vengono salvati su file con codifica binaria e normalmente non sono retrocompatibili, pertanto non è possibile ripristinarli su versioni di SQL Server più vecchie di quelle utilizzate per creare il backup.

MySql

MySQL è un DBMS open source distribuito da Oracle, esistono diversi tipi di licenza con prezzi variabili fra gratuito e diverse migliaia di euro per installazione.

Normalmente MySQL crea un file per ogni tabella, questo crea un enorme problema: negli ambienti Linux i nomi di file sono case sensitive mentre negli ambienti Windows non lo sono, con conseguenti rischi in caso di passaggio del database da una piattaforma all'altra.

phpMyAdmin

MySQL può essere amministrato tramite la shell testuale del sistema operativo, ma è più comodo utilizzare un sistema assistito che semplifichi le operazioni e che consenta di amministrare il database senza dover necessariamente operare direttamente dal computer sul quale è installato il servizio.

Il sistema tipicamente utilizzato assieme al DBMS è phpMyAdmin, un'applicazione web che consente di effettuare tutte le operazioni sul database (dall'esecuzione di query di selezione all'amministrazione degli utenti).

La caratteristica principale di phpMyAdmin è che consente di lavorare da remoto come se si fosse in locale, è sufficiente che il server web che ospita phpMyAdmin risieda sullo stesso computer che ospita il servizio MySQL, l'utilizzo del database con connessione locale consente di scavalcare i limiti imposti da molti server che non consentono la gestione del database da remoto.

HeidiSQL

Una delle alternative a phpMyAdmin è HeidiSQL, un'applicazione locale che si connette al database remoto e ne consente l'amministrazione.

Tramite HeidiSQL è possibile effettuare qualsiasi operazione sul database, compatibilmente con i permessi dati dal server alle connessioni remote.

A differenza di phpMyAdmin, HeidiSQL effettua una connessione remota al server di database, questo implica che se il server non consente connessioni remote (situazione che si verifica spesso) non è possibile utilizzare HeidiSQL.

Gestione degli utenti

MySQL utilizza una gestione locale degli utenti e dei permessi, questo consente di gestire la sicurezza degli accessi in maniera semplice.

La gestione della sicurezza consente di specificare le operazioni ammissibili per ogni utente.

Backup e ripristino di database

MySQL consente di creare e ripristinare i database creando file SQL, ossia dei file di testo contenenti tutte le query necessarie a ricreare la struttura del database e ad inserire i dati contenuti nelle tabelle.

Sommario