Capitolo 6

Esercizio 6.1

Si supponga di avere le tabelle:

Magazzino(Prodotto, QtaDisp, Soglia, QtaRiordino)
OrdineInCorso(Prodotto, Qta)

Scrivere una procedura SQL che realizza il prelievo dal magazzino accettando 2 parametri, il prodotto prod e la quantità da prelevare QtaPrelievo. La Procedura deve verificare inizialmente che QtaPrelievo sia inferiore al valore di QtaDisp per il prodotto indicato. QtaPrelievo viene quindi sottratta al valore di QtaDisp. A questo punto la procedura verifica se per il prodotto QtaDisp risulta minore di Soglia, senza che in OrdineInCorso compaia già una tupla relativa al prodotto prelevato; se si, viene inserito un nuovo elemento nella tabella OrdineInCorso. Con i valori di Prod e del corrispondente attributo QtaRiordino.

Soluzione:

```
#include<stdlib.h>
main()
{
exec sql begin declare section;
        char Prodotto, prod;
        int QtaDisp, Soglia, QtaRiordino, Qta, Qta1, i;
exec sql end declare section;

exec sql declare MagazzinoCursore cursor for
        select  Prodotto, QtaDisp, Soglia, QtaRiordino
        from Magazzino;
exec sql open MagazzinoCursore;

prod = sceltaProdotto();
Qta= sceltaQta();
i=0;

do{
exec sql fetch MagazzinoCursore into
        :Prodotto, :QtaDisp, :Soglia, :QtaRiordino;
if (Prodotto == prod) i=1;
}while(Prodotto == prod || sqlca.sqlcode==0 )

if (i=1){
        printf("ERRORE – Prodotto non trovato");
        exit(1);
        }
if (QtaDisp < Qta){
        printf("ERRORE - Quantità non disponibile");
        exit(1);
        }
QtaDisp = QtaDisp – Qta;

if (QtaDisp < Soglia){
        Qtariordino = QtaRiordino + QtaDisp - Soglia;
        }
```

```
exec sql update magazzino
         set QtaDisp = :QtaDisp
         set QtaRiordino = :QtaRiordino
         where current of MagazzinoCursore;

exec sql declare OrdineCursore cursor for
         select  Prodotto, Qta
         from OrdineInCorso;
exec sql open OrdineCursore ;

i=0;
do{
exec sql fetch  OrdineCursore into
         :Prodotto, :Qta1;
if (Prodotto == prod) i=1;
}while(Prodotto == prod || sqlca.sqlcode==0 )

if (i=1){
         exec sql update OrdineInCorso
                  set Qta = :QtaRiordino;
                  where current of OrdineCursore;
         }
else exec sql insert into OrdineInCorso
         values(:prod,:QtaRiordino)";

}
```

Esercizio 6.2

Realizzare una procedura in un linguaggio di programmazione di alto livello che tramite SQL Embedded elimina dalla tabella DIPARTIMENTO l'elemento che ha il nome che viene fornito come parametro alla procedura.

Soluzione:

Soluzione in C:

```
#include<stdlib.h>
main()
{
char Nome1;
char Nome2;
begin declare section;
         char Nome;
exec sql end declare section;

exec sql declare DipCursore for
         select Nome
         from Dipartimento;
exec sql open DipCursore;

do {
exec sql fetch DipCursore into
         :Nome1;
if (Nome1 == Nome2)
         exec sql delete from Dipartimento
         where nome = :Nome1;
}while (sqlca.sqlcode==0)
}
```

Esercizio 6.3

Realizzare un programma in un linguaggio di programmazione di alto livello che tramite SQL Embedded costruisce una videata in cui si presentano le caratteristiche di ogni dipartimento seguito dall'elenco degli impiegati che lavorano nel dipartimento, ordinati per cognome

Soluzione:

```
#include<stdlib.h>
main()
{

exec sql begin declare section;
         char Nome[20], Città[20], CognomeImpiegato[20],
                  NomeImpiegato[20];
         int NumeroDip;
exec sql end declare section;

exec sql declare DipCursore cursor for
         select Nome, Citta, NumDip
         from DIPARTIMENTO;

exec sql declare ImpCursore cursor for
         select CognomeImpiegato, NomeImpiegato
         from Impiegato join Dipartimento on
         Impiegato.Dipartimento=Dipartimento.:Nome
         order by CognomeImpiegato;
exec sql open DipCursore;
do
{
exec sql fetch DipCursore into
         :Nome, :Città, :NumeroDip;
printf("Dipartimento: ", Nome, " situato in: ",Città,
         " Numero Dipendenti: ",NumeroDip." "Dipendenti:");
         exec sql open ImpCursore;
         do
         {
         exec sql fetch ImpCursore into
                  :CognomeImpiegato, :NomeImpiegato;
         printf( CognomeImpiegato," ", NomeImpiegato);
         }while(sqlca.sqlcode==0);
         exec sql close cursor ImpCursore;
}while(sqlca.sqlcode==0);
exec sql close cursor DipCursore;
}
```

Esercizio 6.4

Realizzare l'esercizio precedente usando ADO

Soluzione:

```
#include<stdlib.h>
main()
{

conn ADODB.Connection;
impiegati ADODB.Recordset;
dipartimenti ADODB.Recordset;
comandoSQL ADODB.Command;
buffer string;

conn = new ADODB.connection;
conn.open(“Server”,”Utente”,”Password”);

dipartimenti = New ADODB.Recordset;
buffer=”select Nome, Citta, NumDip from DIPARTIMENTO”;
comandoSQL.CommandText = buffer;
dipartimenti.Open ComandoSQL(conn);




do{
        printf(“Dipartimento: “, dipartimenti!Nome, “ situato in:
        “,dipartimenti.Città,” Numero Dipendenti:
        “,dipartimenti.NumeroDip.” “Dipendenti:”);

do
{
        impiegati = New ADODB.Recordset;
        buffer = “select CognomeImpiegato, NomeImpiegato
                from Impiegato join Dipartimento on
                Impiegato.Dipartimento=Dipartimento.”,
                dipartimenti!Nome,
                “order by CognomeImpiegato”;
        comandoSQL.CommandText = buffer;
        impiegati.Open ComandoSQL(conn);

        printf( impiegati.CognomeImpiegato,” “,
                impiegati.NomeImpiegato);
        impiegati.movenext;
        }while(impiegati.EOF);
dipartimenti.movenext;
}while(dipartimenti.EOF);
}
```

Esercizio 6.5

Realizzare un programma java che scandisce gli impiegati ordinati per cognome e inserisce ogni impiegato che si trova in una posizione che è un multiplo di 10 in una tabella IMPIEGATIESTRATTI

Soluzione:

```
import java.sql.*;
public class ImpiegatiEstratti {
public static void main(String[] arg){

connection conn = null;
try{
        // carica driver e inizializza connessione
        Class.forName(“sun.jdbc.odbc.jbdcObdcDriver”);
        conn = DriverManager.GettConnection(“jdbc:odbc:impiegati)
        }
try{
        Statement interrogazione = conn.createStatement();
        ResultSet risultato = interrogazione.executeQuery(
                “select *
                 from IMPIEGATI”




        Statement interrogazione1 = conn.createStatement();
        ResultSet risultato1 = interrogazione1.executeQuery(
                “create table IMPIEGATIESTRATTI
                        (
                        NomeImpiegato char(20),
                        CognomeImpiegato char(20),
                        dipartimento char(20),
                        stipedio integer,
                        primary key(NomeImpiegato,CognomeImpiegato)
                        )
        i=0;
        While(risultato.next()) {

        if(i=10)
        {
        string NomeImpiegato = risultato.getString(“NomeImpiegato”);
        string CognomeImpiegato =
risultato.getString(“CognomeImpiegato”);
        string Dipartimento= risultato.getString(“Dipartimento”);
        int Stipendio = risultato.getString(“Stipendio”);
```

```
        Statement interrogazione2 = conn.createStatement();
        ResultSet risultato2 = interrogazione2.executeQuery(
                "insert into IMPIEGATIESTRATTI
                        values(“ NomeImpiegato”,”
                        CognomeImpiegato”,”
                        dipartimento”,”
                        stipedio”,
                        )”;
        i=0;
}}}}
```

Esercizio 6.6

Realizzare un programma che accede al contenuto di una tabella

Capitolo(Numero, Titolo, Lunghezza)

che descrive i capitoli di un libro, con il numero e la dimensione delle pagine. Il programma quindi popola una tabella

Indice(Numero, Titolo, NumPagine)

in cui si presenta il numero di pagina nel quale inizia il capitolo, supponendo che il capitolo 1 inizia sulla prima pagina e che i capitoli devono iniziare su pagine dispari (eventualmente introducendo una pagina bianca alla fine del capitolo)

Soluzione:

```
#include<stdlib.h>
main()
{
exec sql begin declare section;
        char Titolo[50];
        int Numero, Lunghezza;
exec sql end declare section;

exec sql declare CapCursore cursor for
        select Numero, Titolo, Lunghezza
        from Capitolo;
exec sql open CapCursore;
exec sql create table Indice
        (
        Numero integer primary key,
        Titolo char(50),
        NumPagine integer
        );
do{
exec sql fetch CapCursore into
        :Numero, :Titolo, :Lunghezza;
if (Lunghezza%2 != 0)
        Lunghezza = Lunghezza + 1; // Pagina Bianca
exec sql insert into Indice
        values(:Numero,:Titolo,:Lunghezza)";
}while(sqlca.sqlcode == 0)

exec sql close cursor CapCursore;
}
```

Esercizio 6.7 Con riferimento al seguente schema relazionale:

- IMPIEGATI (CodiceFiscale, Cognome, Nome, DataNascita, Dipartimento, Stipendio) con vincolo di integrità referenziale fra l'attributo Dipartimento e la relazione DIPARTIMENTI
- DIPARTIMENTI (Codice, Nome, Sede)
- PROGETTI (Sigla, Titolo, Valore)
- PARTECIPAZIONE (Impiegato, Progetto, Data) con vincoli di integrità referenziale fra l'attributo Progetto e la relazione PROGETTI e fra l'attributo Impiegato e la relazione IMPIEGATI

scrivere un metodo Java con JDBC (o un frammento di programma in SQL immerso in un linguaggio o pseudolinguaggio di programmazione) che inserisca un impiegato con tutti i dati (letti da input o passati come parametri), verificando l'esistenza del dipartimento, con rifiuto dell'operazione in caso negativo. Assumere, per semplicità, che il sistema non supporti i vincoli di riferimento.
Soluzione

```
static void inserimento(Connection con, String cf,
            String cognome, String nome,
            String dataNascita, String dipartimento,
            int stipendio){
    try {
      // Verifica codice dipartimento
      PreparedStatement pquery =
          con.prepareStatement(
                      "select * " +
                      "from Dipartimenti " +
                      "where Codice = ?");
      pquery.setString(1,dipartimento);
      ResultSet result = pquery.executeQuery();
      if (result.next()){
          pquery.close();
          // Il dipartimento esiste
          PreparedStatement pupdate =
            con.prepareStatement(
              "insert into Impiegati" +
              "(CodiceFiscale,Cognome,Nome," +
              "DataNascita,Dipartimento,Stipendio)" +
              "values (?,?,?,?,?,?)");
          pupdate.setString(1,cf);
          pupdate.setString(2,cognome);
          pupdate.setString(3,nome);
          pupdate.setString(4,dataNascita);
          pupdate.setString(5,dipartimento);
          pupdate.setInt(6,stipendio);
          pupdate.executeUpdate();
```

```
            pupdate.close();
          }
          else {
            pquery.close();
            System.out.println(
                      "Non esiste dipartimento " +
                       dipartimento);
          }
        }
        catch (SQLException e){
            System.out.println("Errore");
            System.out.println(e.getErrorCode() + " " +
                     e.getSQLState() +  e.getMessage() );
        }
    }
```

Esercizio 6.8 Dato lo schema relazionale seguente:

- IMPIEGATI (Codice, Dati, Telefono) con vincolo di integrità referenziale fra l'attributo Dati e la relazione DATIIMPIEGATO
- DATIIMPIEGATI (CodiceDati, Cognome, Nome)

definire il metodo Java (o un frammento di programma in SQL immerso in un linguaggio o pseudolinguaggio di programmazione) getImpiegatoPerNome(String cognome, String nome) che, dato il nome di un impiegato, restituisce un Oggetto Impiegato in cui i campi (Codice, Nome, Cognome, Telefono) sono opportunamente valorizzati.

Soluzione

Si presuppone che nell'azienda un impiegato sia univocamente identificato dal suo nome e dal suo cognome.

```
public Impiegato getImpiegato(String cognome,
 String nome) throws SQLException {

    Impiegato i = null;
    PreparedStatement s = this.con.prepareStatement(
          "setect * " +
          "from Impiegati, DatiImpiegati" +
          "where Dati = CodiceDati" +
          "and cognome = ?" +
          "and nome = ?");
    s.setString(1, cognome);
    s.setString(2, nome);
    ResultSet rs = s.executeQuery();
    if (rs.hasNext()){
        i = new Impiegato (rs.getInt("Codice"),
                           rs.getString("Nome"),
                           rs.getString("Cognome"),
                           rs.getString("Telefono"));
    }
    s.close();
    return i;
}
```

Esercizio 6.9 Si consideri il seguente schema relazionale (con gli evidenti vincoli di integrità referenziale):

- VENDITE (CodArticolo, CodNegozio, CFCliente, Data, Quantità) con vincoli di integrità referenziale fra l'attributo CodArticolo e la relazione ARTICOLI, fra l'attributo CodNegozio e la relazione NEGOZI, fra l'attributo CFCliente e la relazione CLIENTI
- ARTICOLI (CodArticolo, Descrizione, CodMarca, CodCategoria, Prezzo) con vincoli di integrità referenziale fra l'attributo CodMarca e la relazione MARCHE e fra l'attributo CodCategoria e la relazione CATEGORIE
- CLIENTI (CFCliente, Cognome, Nome, Età)
- NEGOZI (CodNegozio, Nome, Indirizzo, Città, Provincia, Regione)
- MAECHE (CodMarca, Nome, CodNazione, Nazione)
- CATEGORIE (CodCategoria, Descrizione).

Scrivere un metodo Java con JDBC che inserisca un nuovo negozio con codice, nome, indirizzo e città (letti da input o passati come parametri), prelevando provincia e regione da altre della stessa tabella (nell'ipotesi che, fissata la città, provincia e regione siano univocamente determinate) e segnalando come errore (o eccezione) il caso in cui i dati sulla città non siano disponibili.
Soluzione

```
public void insertNegozio(String codice, String nome,
 String indirizzo, String citta)
 throws SQLException {

  /* query che preleva regione e provincia
     a cui appartiene una data citta */
  PreparedStatement getProvReg = con.prepareStatement(
      "select Provincia, Regione" +
      "from Negozi" +
      "where citta = ?");
  getProvReg.setString(citta);
  ResultSet rs = getProvReg.executeQuery();
  if (rs.size() == 0)
      throw new NoDataFoundException();
  String prov = rs.getString("Provincia");
  String reg = rs.getString("Regione");
  getProvReg.close();

   /* query di inserimento */
  PreparedStatement ins = con.PrepareStatement(
      "insert into negozi values (?,?,?,?,?,?);");

  ins.setString(1, codice);
  ins.setString(2, nome);
```

```
        ins.setString(3, indirizzo);
        ins.setString(5, prov);
        ins.setString(6, reg);

        ins.executeUpdate();
        ins.close();
    }
```

Esercizio 6.10 Con riferimento allo relazionale seguente:

- FARMACI (Codice, NomeFarmaco, PrincipioAttivo, Produttore, Prezzo) con vincolo di integrità referenziale fra Produttore e la relazione PRODUTTORI e fra PrincipioAttivo e la relazione SOSTANZE
- PRODUTTORI (CodProduttore, Nome, Nazione)
- SOSTANZE (ID, NomeSostanza, Categoria)

scrivere un metodo Java con JDBC che (supponendo già disponibile una connessione, passata come parametro) stampi un prospetto con tutti i farmaci, organizzati per produttore:

CodProduttore Nome Nazione
CodiceFarmaco NomeFarmaco Prezzo Sostanza
CodiceFarmaco NomeFarmaco Prezzo Sostanza
...
CodProduttore Nome Nazione
...

Soluzione

```
public void prontProspetto(Connection con)
 throws SQLException{

    /* query di estrazione dati produttore */
    String q1 = "select CodProduttore" +
                " Nome, Nazione" +
                "from Produttori";

    /* query di estrazione dati per i farmaci */
    String q2 = "select CodiceFarmaco," +
                " NomeFarmaco, Prezzo, Sostanza" +
                "from Farmaci join Produttore" +
                "    on PrincipioAttivo = ID" +
                "where CodProduttore = ?";

    PreparedStatement p = con.prepareStatement(q1);
    ResultSet rs, rs1;
    PreparedStatement p2 = con.prepareStatement(q2);
    rs.con.executeQuery();

    while(rs.hasNext()) {
        p1.setString(rs.getString("CodProduttore");
        rs1=p1.executeQuery();
        System.out.println(
               rs.getString("CodProduttore") +
               " " + rs.getString("Nome") +
               " " + rs.getString("Nazione"));
        while(rs1.hasNext()) {
```

```
                System.out.Println(
                   rs1.getString("CodiceFarmaco") +
                   " " + rs1.getString("NomeFarmaco") +
                   " " + rs1.getString("Prezzo") +
                   " " + rs1.getString("Sostanza"));
            }
        }
        p.close();
        p2.close()
    }
```

Esercizio 6.11 Si consideri una base di dati che contiene informazioni sugli impiegati, i progetti e le sedi di una azienda, con le partecipazioni degli impiegati ai progetti e le sedi di svolgimento dei progetti stessi; essa contiene le seguenti relazioni:

- IMPIEGATI (Matricola,Cognome,Nome,Progetto), con vincolo di integrità referenziale fra Progetto e la relazione PROGETTI
- PROGETTI (Codice,Titolo)
- SEDI (Nome,Città,Indirizzo)
- SVOLGIMENTO (Progetto,Sede), con vincoli di integrità referenziale fra Progetto e la relazione PROGETTI fra Sede e la relazione SEDI.

Formulare in Java-JDBC (o con SQL immerso in un linguaggio o pseudolinguaggio di programmazione), una classe (o un frammento di programma) che stampa tutti i progetti (con codice e titolo) e, per ciascuno di essi, gli impiegati coinvolti (mostrando matricola e cognome); in sostanza, va prodotto un prospetto del tipo seguente:

CodProgetto TitoloProgetto
MatricolaImpiegato CognomeImpiegato
MatricolaImpiegato CognomeImpiegato
...
CodProgetto TitoloProgetto
MatricolaImpiegato CognomeImpiegato
...

Soluzione

```
public void printProspetto(Connection con)
 throws SQLException {

    String codice = null;
    ResultSet rs1, rs2;

    /* query di estrazione dei progetti */
    String q1 = "select Codice, Titolo," +
                "from Progetto";

    /* query di estrazione degli impiegati */
    String q2 = "select Matricola, Cognome" +
                "from Impiegato join Progetto" +
                "    on (Progetto = Codice)" +
                "where codice = ?";

    PreparedStatement s1 = con.prepareStatement(q1);
    PreparedStatement s2 = con.prepareStatement(q2);

    rs1 = s1.executeQuery();
```

```
    while(rs1.hasNext()) {
        codice = rs1.getString("Codice");
        System.out.println( codice +
                " " + getString("Titolo"));
        s2.setString(codice);
        rs2. = s2.executeQuery();
        while(rs2.hasNext()) {
             System.out.println(
                 rs2.getString("Matricola" +
                 " " + rs2.getString("Cognome"));
        }
    }
    s1.close();
    s2.close();
}
```

Esercizio 6.12 Estendere la risposta al quesito precedente mostrando anche, per ciascun progetto, la lista delle sedi di svolgimento, costruendo quindi un prospetto come il seguente:

CodProgetto TitoloProgetto
MatricolaImpiegato CognomeImpiegato
MatricolaImpiegato CognomeImpiegato
…
NomeSede Città
NomeSede Città
…
CodProgetto TitoloProgetto
…

Soluzione

```
public void printProspetto(Connection con)
 throws SQLException {

    String codice = null;
    ResultSet rs1, rs2, rs3;

    /* query di estrazione dei progetti */
    String q1 = "select Codice, Titolo," +
                "from Progetto";

    /* query di estrazione degli impiegati */
    String q2 = "select Matricola, Cognome" +
                "from Impiegato join Progetto" +
                "    on (Progetto = Codice)" +
                "where codice = ?";

    /* query di estrazione delle sedi */
    String q3 = "S.Nome as NomeSede," +
                "S.Citta as Citta," +
                "from Progetto P join Svolgimento Sv" +
                "    on (P.Progetto = S.Progetto)" +
                "        join Sedi S " +
                "            on (S.Nome = Sv.Sede)" +
                "where Codice = ?";


    PreparedStatement s1 = con.prepareStatement(q1);
    PreparedStatement s2 = con.prepareStatement(q2);
    PreparedStatement s3 = con.prepareStatement(q3);

    rs1 = s1.executeQuery();
```

```
        while(rs1.hasNext()) {
           codice = rs1.getString("Codice");
           System.out.println( codice +
                  " " + getString("Titolo"));
           s2.setString(codice);
           rs2. = s2.executeQuery();
           while(rs2.hasNext()) {
               System.out.println(
                  rs2.getString("Matricola" +
                  " " + rs2.getString("Cognome"));
           }

           s3.setString(codice);
           rs3 = s3.executeQuery();
           while(rs3.hasNext()) {
               System.out.println(
               rs3.getString("NomeSede") + " " +
                     rs3.getString("Citta"));

           }
        }
        s1.close();
        s2.close();
        s3.close();
    }
```

Esercizio 6.13 Si ha uno schema di tabella IMPIEGATO (Nome, Indirizzo, Capo) in cui l'attributo Capo rappresenta il nome del superiore dell'impiegato, descritto a sua volta nella tabella. Definire il metodo Java (o un frammento di programma in SQL immerso in un linguaggio o pseudolinguaggio di programmazione) setCapo(Impiegato i1, Impiegato i2) che memorizza la relazione tra capo e sottoposto esistente tra due impiegati rifiutando l'inserimento se la relazione o la sua inversa già esistono. Supporre che gli oggetti Impiegato abbiano una variabile di istanza per ogni campo della corrispondente colonna nello schema relazionale con opportuna corrispondenza di tipi.

Soluzione

```
public boolean setCapo(Impiegato i, Impiegato capo)
 throws SQLException {

   /* query di verifica */
   String qV = "select * " +
               "from Impiegato" +
               "where nome = ?" +
               "and capo = ?" +
               "or" +
               "nome = ?" +
               "and capo = ?";

   /* query di aggiornamento */
   String upd = "update Impiegati" +
                "   set capo = ?" +
                "   where nome = ?";

   PreparedStatement p1 = con.prepareStatement(qV);

   p1.setString(1, i.getNome());
   p1.setString(2, capo.getNome());
   p1.setString(3, capo.getNome());
   p1.setString(4, 1.getNome());
   ResultSet rs = p1.executeQuery();
   if(rs.size() > 0) {
       p1.close();
       return false;
   }

   PreparedStatement p2 = con.prepareStatement(upd);
   p2.setString(1, capo.getName());
   p2.setString(2, i.getName());
   p2.executeUpdate();

   p1.close();
```

```
            p2.close();
            return true;
        }
```

Esercizio 6.14 Effettuare una comparazione sull'efficienza dei due approcci seguenti per ottenere tutti i corsi in cui un determinato professore insegna. Il metalinguaggio utilizzato é una ipotetica codifica in cui si pu immergere l'SQL.

1.
```
execute query1() into cursore1;

do
    loop while (not cursore1%empty);
    fetch cursore1 into Professore;
    execute query2(Professore) into cursore2;

    do
      loop while (not %cursore2%empty);
      fetch cursore2 into corso;
      print(Professore.Nome, Professore.Cognome,
            Corso);

    end loop;

end loop;


query1:

select * from Professori;

query2:

select *
from Corsi
where Professore = ?;
```

2.
```
select P.Cognome, P.Nome, Corso
from Professori P, Corsi C
where C.Professore = P.MatricolaProfessore;
```

Soluzione
L'approccio 2 è sicuramente più efficiente. La query è predicibile a priori ed interamente interpretata ed ottimizzata dal dbms, in grado di garantire prestazioni migliori di un qualsiasi compilatore di linguaggio di programmazione. Il programma 1 fornisce implicitamente una implementazione di un join mediante l'annidamento di due cicli. Tale join è sicuramente ottimizzato in maniera più avanzata attraverso la dichiarazione esplicita di voler effettuare una tale operazione a livello database. Inoltre, al punto 2 viene eseguita una sola query, mentre il punto 1 comporta l'esecuzione di una query di indagine per ogni professore.