**Вопросы к экзамену по БиБД:**

1. Сущность технологии BDE
2. Сущность технологии ADO
3. Технология клиент-сервер
4. Архитектура ADO.Net
5. Работа на основе OleDbConnection
6. Работа на основе ODBC Connection
7. Работа на основе SQL Connection
8. Строка соединения и её параметры
9. Командная строка. Запросы с параметрами
10. Вызов хранимых процедур
11. Работа с курсорами
12. Наборы данных DataSet
13. Типизированные наборы данных. Триггеры
14. Использование отношений для перемещения по таблицам
15. Визуальный интерфейс с БД
16. Связывание компонентов для отображения записей
17. Работа с Grid
18. Соединение сервера SQL через Server Explorer
19. Технология Entity Object Framework (концепция и применение)
20. Работа с XML документами
21. Обработка регулярных выражений
22. ASP скрипты
23. Соединение с БД в ASP скриптах
24. Скрипты ASP.Net
25. Web сервисы
26. Технология LINQ to Object
27. Технология LINQ to XML
28. Работа в БД через локальную сеть
29. MySQL. Системные таблицы. Работа с C#
30. Создание и вызов функций в MySQL. Примеры
31. Технология MVC. Базовые понятия и применение
32. Понятия о нормальных формах (1НФ, 2НФ, 3НФ)
33. Нормальная форма Бойса-Кодда (4НФ)
34. Аномалии БД
35. Аксиомы Армстронга
36. Определение избыточных атрибутов
37. Ациклические графы вывода
38. Проверка избыточных функциональных зависимостей
39. Минимизация числа функциональных зависимостей
40. ER диаграммы
41. Построение таблиц по ER диаграммам
42. Задача о минимизации ключа
43. Взаимодействие C# с Oracle
44. Хранимые процедуры и модули Oracle
45. Развитые типы данных Oracle (массив, таблица, объект)

**Ответы к вопросам:**

1. **Сущность технологии BDE**

BDE (Borland Database Engine) – набор драйверов (программа .dll(.exe) которая выполняется как самостоятельный процесс), т.е. программ для доступа к физическим носителям данных. Для работы с BDE обычно создается именованное соединение: Пуск – Панель управления – BDE Администратор – Object – ODBC – DSN – щпк – Build – Задаём имя и путь. Именованное соединение используется в программе для доступа к БД и указывается в строке Connection String. Механизм BDE не основан на классах и непредставляет объектов с их методами и свойствами. Технология BDE не используется начиная с windows Vista.

1. **Сущность технологии ADO**

Следующая технология после BDE была ADO (RDO) – Access Data Object (Remote Data Object) – ООП технология, позволяющая создавать клиент-серверные приложения.

Развитие объектного подхода привело к развитию технологии Ado и Ado.Net (ado = ActiveX Data Object). ActiveX – объект, написанный на «чужом» языке. Обычно такой объект зарегистрирован в реестре, иногда требуется конвертация под интерфейс, например, C#. Использование классов предоставляет богатые возможности в функциональном плане (можно наращивать функциональность путём наследования), увеличивает гибкость, позволяет использовать разные ЯП.

1. **Технология клиент-сервер**

В приложениях клиент-сервер отдельно выделяется клиентская часть (отдельное приложение) и отдельно выделяется серверная (серверное приложение). Клиенту предоставляется серверный компонент (объект), который обеспечивает его методами доступа к данным. Взаимодействие «клиент-сервер» в локальной сети организуется при помощи протоколов (TCP). Протокол предполагает определённую структуру сообщений и порт обработки.

Http – позволяют передавать информацию в локальные и глобальные сети. Развитие объектного подхода привело к развитию технологии Ado и Ado.Net (ado = ActiveX Data Object).

1. **Архитектура ADO.Net**

ADO.Net – платформа, представляющая набор базовых классов для работы с различными БД. Имеются следующие провайдеры:

* ODBC (BDE)
* OLEDB ( Обеспечивает связь с БД, не взаимодействует с MS SQL Server)
* SQL Server Compact Edition (Облегчённый вариант сервера ориентированный на создание web приложений)

ADO.Net пришла на смену технологии BDE, представляет собой набор объектных классов, таких как: Connection, DataSet, DataTable, DataAdapter, DataRows и так далее. Эти классы функционируют с Framework 4.0 (3.5, 3.0 …). Технология .Net впервые была представлена в 2002 году. Microsoft в Visual Studio интегрировала в среду ADO.Net . Также следует отметить развитие web технологий, в том числе и для работы с БД (ASP.Net, ASP.Net MVC).

1. **Работа на основе OleDbConnection**

Пакет OleDB предназначен для работы в БД в Microsoft ( FoxPro, Excel, Access). Важнейший момент связывания с исполняемым соединением – задание строки соединения.

Пример: OleDbConnection MyCon = new OleDbConnection( “Provider = Microsoft.Jet.OleDB.4.0;” + “DataSource = c:\work\my.mdf”);

При открытии соединения информация о нем помещается в pool, так что при закрытии соединения информация сохраняется в пуле для экономии времени. Если соединение открыли успешно, то необходимо подготовить команду для работы с таблицей (select, update) или подготовить команду для вызова хранимой процедуры (ExecuteQuery – Select, Execute N Query). Пример:

OleDB Command Mycmd = myCon.CreateCommand;

Mycmd.Connection = myCon;

String cmdTexts = “Select \* from Sclad”;

Mycmd.CommandText = cmd.text;

Имеются варианты так называемых параметризированных запросов, которые используют параметры конкретизирующие SQL команду, например: …”Select Tovar from sclad where price <= 500”.

Классы параметров:

* DataAdapter – предназначен для физического считывания записей из источника данных в DataSet, для этого используется метод fill. На ряду с адаптером имеется облегчённый DataReader, он работает быстрее, но не предназначен для изменения данных.
* DataSet – позволяет хранить одну и более таблиц одновременно считываемых из источника. Каждая таблица DataTable содержит записи представления в свойстве Rows. Также позволяет читать и конвертировать формат XML, обеспечивает отношения между таблицами.
* DataView – используется при поиске, сортировке, для выполнения различных действий над записями таблиц.
1. **Работа на основе ODBC Connection**

Пример:

using System;

using System.Text;

using System.Data;

using System.Data.Odbc;

namespace ConsoleApplication6

{

 class Program

 {

 static void Main()

 {

 string scon = "DSN = myAcc";

 OdbcConnection mycon = new OdbcConnection(scon);

 string sq = "Select \* from stud";

 OdbcCommand cmd = new OdbcCommand(sq);

 OdbcDataAdapter da = new OdbcDataAdapter();

 da.SelectCommand = cmd;

 DataSet ds = new DataSet();

 mycon.Open();

 da.Fill(ds, "stud");

 ds.Tables[0].RowChanged += new DataRowChangeEventHandler(RowIsChanged);

 ds.Tables[0].Rows[0]["Fam"] = "Doe";

 mycon.Close();

 }

 public static void RowIsChanged(Object sender, DataRowChangeEventArgs e)

 {

 Console.WriteLine("New row value is: " + e.Row["Fam"]);

 Console.ReadLine();

 }

 }

}

Запрос, отличный от Select:

 string sq = "Insert into stud values ('Sidorov',20)";

 OdbcCommand cmd = new OdbcCommand(sq, mycon);

 mycon.Open();

 cmd.ExecuteNonQuery();

1. **Работа на основе SQL Connection**

Пример:

using System;

using System.Text;

using System.Data;

using System.Data.SqlClient;

namespace ConsoleApplication6

{

 class Program

 {

 static void Main()

 {

 string scon = @"server = local;" + "database = stud; uid = boss;" + "password=1234; integratedsecurity = SSPI";

 string sq = "select \* from tovar";

 SqlConnection mycon = new SqlConnection(scon);

 SqlCommand cmd = mycon.CreateCommand();

 cmd.CommandText = sq;

 SqlDataAdapter da = new SqlDataAdapter();

 da.SelectCommand = cmd;

 DataSet ds = new DataSet();

 mycon.Open();

 da.Fill(ds, "stud");

 ds.Tables[0].RowChanged += new DataRowChangeEventHandler(RowIsChanged);

 ds.Tables[0].Rows[0]["Fam"] = "Doe";

 mycon.Close();

 }

 public static void RowIsChanged(Object sender, DataRowChangeEventArgs e)

 {

 Console.WriteLine("New row value is: " + e.Row["Fam"]);

 Console.ReadLine();

 }

 }

}

Значение SSPI использует для логина (вхождения в сервер) учётную запись Windows.

1. **Строка соединения и её параметры**

Имеется 3 типа соединений:

* OdbcConnection (соединение с машиной фирмы Borland)

Пример: Odbc Connection Con = new Connection();

Con.ConnectionString = “DSN = mycon”;

* OleDBConnection (соединение с БД Microsoft)

Пример: OleDbConnection MyCon = new OleDbConnection( “Provider = Microsoft.Jet.OleDB.4.0;” + “DataSource = c:\work\my.mdf”);

* SQLConnection (соединение с MS SQL Server)

Пример: SqlConnection Scon = new Sqlconnection();

Scon.ConnectionString = “DataSource = ./SQLExpress;” + “UserInstance = true;” + “User Id = stud; Password = 1234”;

Еще пример: SqlConnection MyCon = new SqlConnection(“Server = localhost;” + “DataSource =mydb.mdf;” + IntegratedSecurity = true”);

MyCon.Open();

(примечание: ./ определяет локальный компьютер, т.е. localhost)

Пример:

using System;

using System.Text;

using System.Data;

using System.Data.SqlClient;

namespace ConsoleApplication6

{

 class Program

 {

 static void Main()

 {

 //Подсчёт строк в таблице

 string scon = "Data Source=sqlexpress;Initial Catalog=123;Integrated Security=True";

 string sq = "select \* from Genre";

 SqlConnection mycon = new SqlConnection(scon);

 SqlCommand cmd = mycon.CreateCommand();

 cmd.CommandText = sq;

 SqlDataAdapter da = new SqlDataAdapter();

 da.SelectCommand = cmd;

 DataSet ds = new DataSet();

 mycon.Open();

 da.Fill(ds, "Genre");

 DataTable tbl = ds.Tables[0];

 Console.WriteLine("В таблице имеется {0} строк", tbl.Rows.Count );

 mycon.Close();

 }

 }

}

1. **Командная строка. Запросы с параметрами**

Пример запроса с параметрами:

using System;

using System.Text;

using System.Data;

using System.Data.OleDb;

namespace ConsoleApplication6

{

 class Program

 {

 static void Main()

 {

 string scon = @"provider = Microsoft.Jet.OleDB.4.0; DataSource = C:\\1\\mydb.mdb";

 string sq = "Select \* from stud where Fam = Petrov";

 OleDbConnection con = new OleDbConnection(scon);

 OleDbCommand cmd = con.CreateCommand();

 cmd.CommandText = sq;

 cmd.Parameters.Add("Par1", DbType.String).Direction = ParameterDirection.Input;

 cmd.Parameters[0].Value = "NichihuaSebe!";

 OleDbDataAdapter oda = new OleDbDataAdapter();

 oda.SelectCommand = cmd;

 DataSet ds = new DataSet();

 con.Open();

 oda.Fill(null);

 ds.Tables[0].RowChanged += new DataRowChangeEventHandler(RowIsChanged);

 ds.Tables[0].Rows[0]["Fam"] = "Doe";

 con.Close();

 }

 public static void RowIsChanged(Object sender, DataRowChangeEventArgs e)

 {

 Console.WriteLine("New row value is: " + e.Row["Fam"]);

 Console.ReadLine();

 }

 }

}

Типы параметров: char, DateTime, decimal, Image, int, varchar,variant. Если параметр объявлен как выходной(output), что имеет место при вызове ХП, то значение параметра можно получить через конструкцию: cmd.Parameters[0].Value.

1. **Вызов хранимых процедур**

Пример работы с ХП:

MySql delimiter //

MySql create procedure find cena (IN for varchar(25), oul CEN INT)

->begin

->select price into cen from sclad where Tovar = fov;

->end

->//

Вернуть обратно: MySql > delimeter;

MySql > call fing cena (‘milk’, @x);

MySql >select @x;

Текст процедуры нельзя ввести с помощью LOAD. Просмотр текста процедуры можно с помощью команды:

MySql >show create procedure find cena;

MySql >select body from mysql proc

->where name = “find cena”;

Вызов ХП:

* На удаление:

Procedure proc1( IN par1 INT)

Begin

Delete from Sklad where IO = par1

End

* На изменение:

Procedure proc2( IN par1 INT) N par2 varchar(50))

Begin

Update Sclad

Set Tovar = par2

Where ID = par1

End

* Вставка:

Procedure proc3(IN par1 INT IN par2 varchar(50))

Begin

Insert into sclad value (par1,par2)

End

1. **Работа с курсорами**

Работа с курсором на языке MySql:

MySql >delimiter //

 ->create procedure proc1(out tov varchar(25))

 ->begin

 ->DECLARE cur CURSOR for select Tovar from sclad

 ->where price = (select min(price) from sclad)

 ->open cur

 ->fetch cur into tov

 ->close cur

 ->end

 ->//

MySql >delicimer

Работа с курсором на языке T-SQL:

CREATE PROCEDURE MyProc AS

DECLARE Mycur CURSOR FOR

 SELECT tovar,price FROM Sclad WHERE price > 2000

 FOR READ ONLY

 OPEN Mycur

 DECLARE @tovar VARCHAR(40)

 DECLARE @price INT

 FETCH NEXT FROM Mycur INTO @tovar, @price

 PRINT @tovar + str(@price)

 CLOSE Mycur

 DEALLOCATE Mycur

Fetch – переход к следующей записи. Deallocate – удаляет курсор.

На ряду с NEXT используется: PRIOR (переход назад), FIRST, LAST, ABSOLUTE 4/-4 (переход к 4й записи сначала/к 4й записи с конца), RELATIVE 2 (перейти вперёд на 2 записи от текущей).

 Пример:

CREATE PROCEDURE myproc AS

 DECLARE @n INT

 DECLARE mycur CURSOR SCROLL FOR

 SELECT Tovar, price FROM sclad FOR

 READ ONLY

 OPEN mycur

 DECLARE @tovar VARCHAR(40)

 DECLARE @price INT

 SET @n = 0

 WHILE(@n < @@CURSOR\_ROWS)

 BEGIN

 SET @n = @n+1

 FETCH ABSOLUTE @n FROM mycur

 INTO @tovar, @price

 END

 PRINT Tovar + str(price)

 CLOSE mycur

DEALLOCATE mycur

1. **Наборы данных DataSet**

Для создания типизированного набора DataSet надо добавить сам DataSet как новое решение. Далее необходимо активировать соединение с сервером, чтобы подключиться к конкретной БД. (Добавляем соединение через окно Server Explorer). Далее нужно мышью перенести имена таблиц в окно DataSet, которое занимает весь экран. Кроме таблиц можно добавить и связи. Пример работы DataSet:

 private void button3\_Click(object sender, EventArgs e)

 {

 string scon = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=K:\\123.accdb";

 SqlConnection mycon = new SqlConnection(scon);

 string q = "select \* from stud";

 SqlCommand cmd = mycon.CreateCommand();

 cmd.Connection = SqlConnection(mycon); //?

 cmd.CommandText = q;

 SqlDataAdapter da = new SqlDataAdapter();

 da.SelectCommand = cmd;

 DataSet1 ds = new DataSet1();

 mycon.Open();

 da.Fill(ds, "stud");

 DataTable mytab = ds.Tables["stud"];

 mycon.Close();

 }

Можно реализовать добавление, удаление, поиск строк.

* Поиск выполняется через фильтр:

Пример:

DataTable tbl = ds.Tables[0];

 tbl.DefaultView.RowFilter = "Name = 'Petrov' ";

Пример:

DataTable tbl1 = ds.Tables[0];

 DataRow[] drs = tbl1.Select("age > 18");

 foreach (DataRow dr in drs)

 { ... }

* Добавление (редактирование) строки:

Пример:

 DataTable tbl = ds.Tables[0];

 DataRow dr = tbl.NewRow();

 dr["fam"] = "Mays";

 dr["age"] = 20;

 tbl.Rows.Add(dr);

* Удаление:

Пример:

Удалить 10 строк

DataTable tbl = ds.Tables[0];

 tbl.Rows.RemoveAt(10);

Пример:

Удалить строку dr

tbl.Rows.Remove(dr);

1. **Типизированные наборы данных. Триггеры**

Триггер – это ХП, которая реагирует на удаление, изменение и добавление записей.

Пример:

using System;

using System.Text;

using System.Data;

using System.Data.OleDb;

namespace ConsoleApplication6

{

 class Program

 {

 static void Main()

 {

 OleDbConnection mycon = new OleDbConnection( "Provider = Microsoft.Jet.OleDB.4.0;" + "DataSource = C:\\1\\mydb.mdb");

 OleDbCommand mycmd = mycon.CreateCommand();

 mycmd.Connection = mycon;

 mycon.Open();

 string sq = "Select \* from stud";

 mycmd.CommandText = sq;

 DataSet ds = new DataSet();

 OleDbDataAdapter oda = new OleDbDataAdapter(mycmd);

 oda.Fill(null);

 ds.Tables[0].RowChanged += new DataRowChangeEventHandler( RowIsChanged );

 ds.Tables[0].Rows[0]["Fam"] = "Doe";

 mycon.Close();

 }

 public static void RowIsChanged(Object sender, DataRowChangeEventArgs e)

 {

 Console.WriteLine("New row value is: " + e.Row["Fam"]);

 Console.ReadLine();

 }

 }

}

Аналогично изменение:

ds.Tables[0].RowChanging += new DataRowChangeEventHandler(MyProcForCh);

Удаление после завершения:

ds.Tables[0].RowDeleted += new DataRowChangeEventHandler(ProcForDEl);

1. **Использование отношений для перемещения по таблицам**

Пример:

При считывании таблиц в DataSet связи между таблицами теряются. Допустим, в Access есть 2 таблицы:

* Stud (содержит поля name, grp)
* Range (содержит поля name, range)

Таблицы связаны по полю name. Создадим приложение на основе формы:



Кнопка Load:

private void button1\_Click(object sender, EventArgs e)

 {

 string constr = @"provider = Microsoft.Jet.OleDb.4.0;" + @"data source = C:\Users\Violent\Documents\123.mdf";

 System.Data.OleDb.OleDbConnection myCon = new System.Data.OleDb.OleDbConnection(constr);

 string sq = "Select Name, grp from Stud";

 System.Data.OleDb.OleDbCommand cmd = myCon.CreateCommand();

 cmd.CommandText = sq;

 System.Data.OleDb.OleDbDataAdapter oda1, oda2;

 DataSet myDataSet;

 myCon.Open();

 oda1.Fill(myDataSet , "stud A" );

 sq = "Select Name, Range from Range";

 cmd.CommandText = sq;

 oda2.Fill(myDataSet, "Range");

 DataSet myDataTable1, myDataTable2;

 myDataSet.Relations.Add("ret", myDataSet.Tables["stud A"].Columns["Name"],

 myDataSet.Tables["Range A"].Columns["Name"]);

 Name :textBox1.DataBindings.Add("Text", myDataTable1, "Name");

 grp :textBox2.DataBindings.Add("Text", myDataTable1, (string)"grp");

 Range:textBox3.DataBindings.Add("Text", myDataTable2, (string) "range");

 }

Кнопка Show:

 private void button2\_Click(object sender, EventArgs e)

 {

 DataRow myDataTable1 = null;

 DataRow[] rows = myDataTable1.Table.Select("Name' " + textBox1.Text + "'");

 if (rows.Length > 0 )

 {

 textBox2.Text = rows[0]["grp"].ToString();

 var childrows = rows[0].GetChildRows("rd");

 textBox3.Text = childrows[0]["Range"].ToString();

 }

 }

Пример:

\*вот этот код точно должен работать: \*

string con = @"Provider=Microsoft.Jet.OLEDB.4.0.; data source=D:\1.mdb";

OleDbConnection myconn = new OleDbConnection(con);

string s = "Select name,groop,rate from stud INNER JOIN rate on stud.id=rate.id";

OleDbCommand cmd = new OleDbCommand();

cmd.CommandText = s;

OleDbDataAdapter ad = new OleDbDataAdapter(s, myconn.ConnectionString);

DataTable table = new DataTable();

table.Locale = System.Globalization.CultureInfo.InvariantCulture;

ad.Fill(table);

dataGridView1.DataSource = table;

1. **Визуальный интерфейс с БД**

Создадим какую-либо БД из нескольких таблиц. Например, из 4 таблиц. Будем использовать Visual Studio 2012. Добавим подключение в обозревателе баз данных.

* DataSet

Нажмём по имени проекта правой кнопкой мыши и добавим новый элемент DataSet1. Выделим и переместим в конструктор наши таблицы. Получим такой интерфейс:



* Linq to sql

Нажмём по имени проекта правой кнопкой мыши и добавим новый элемент DataClasses1. Выделим и переместим в конструктор наши таблицы. Получим такой интерфейс:



* Ado.net

Нажмём по имени проекта правой кнопкой мыши и добавим новый элемент Model1. Открывается мастер моделей -> Создать из базы данных -> Далее -> подтверждение или выбор другого соединения -> Далее -> Выбор таблиц -> Готово. Получим такой интерфейс:



1. **Связывание компонентов для отображения записей**
2. **Работа с GridView**
* Создаём приложение Windows Application.
* Открываем или создаём соединение с источником данных.
* Размещаем на форме компонент Data GridView, привязываем его к источнику данных. На ряду с Data GridView система добавляет не отображаемый компонент DataSetBindingSource. Нужно в свойстве DataMember компонента DataSetBindingSource указать и имя таблицы. Кроме этого, свойство DataSource объекта Data GridView установить равным DataSetBindingSource1. После этого можно запускать программу, наша таблица должна быть отображена.
* Data GridView можно сделать невидимой следующим образом:

Public Form1() { Data GridView1.Visible = False; }

Пример: Обработчик кнопки «Поиск»

Private void Button1.Click( … )

{

String s = textbox1.text.Tolower();

myDataSetBindingSource.filter = string.Format(“prod like ‘% {0}’ “,s);

dataGridView1.refresh();

dataGridView1.visible = true;

}

1. **Соединение сервера SQL через Server Explorer**

Установка IIS (Internet Information Services): Пуск –Панель управления – Установка и удаление программ – Компоненты Windows – Ставим галочку напротив IIS. Если галочка отсутствует, то IIS нужно установить.

Настройка и запуск сервера IIS: Мой компьютер – Управление – Службы и приложения – IIS – web-узел по умолчанию – Щелчок правой кнопкой мыши и выбор пункта «Свойства». На вкладке «Домашний каталог» задаётся значение TCP порта, также задаётся адрес домашней страницы, откуда IIS будет запускать скрипты: c:\inetpub\wwwroot.





Проверка работы: Открываем «Службы и приложения» - находим строку IIS Admin – Активизируем правой кнопкой мыши контекстное меню и нажимаем пункт Пуск.

Теперь, при вводе адреса <http://localhost:80> в строке браузера должно открываться домашнее окно IIS.

1. **Технология Entity Object Framework (концепция и применение)**

Entity Framework предназначена для работы с объектами БД (таблиц, представлений) как с экземплярами классов.

Пример работы:

Предположим, что БД уже есть (соединение установлено):

* Создаём консольный проект.
* Добавляем в проект модель данных Entity Model. (В окне обозревателя решений Solution Explorer активируем контекстное меню. ПКМ по имени проекта, из меню выбираем добавить – Новый элемент – выбираем Ado.Net entity data model – ок – запускается мастер, из него выбираем или создаём базу данных – Выбираем таблицы – ок. Если всё правильно сделано, то открывается окно со схемой бд.

Базовое понятием Entity Model является контекст. Контекст определяется именем соединения, под которым сохраняется модель в мастере соединений.

1. **Работа с XML документами**

XML (англ. eXtensible Markup Language — расширяемый язык разметки; произносится [экс-эм-э́ль]) — рекомендованный Консорциумом Всемирной паутины (W3C) язык разметки. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов привело к широкому использованию как собственно XML, так и множества производных специализированных языков на базе XML в самых разнообразных программных средствах.

Пример кода:

Создание XML документа:

private void button1\_Click(object sender, EventArgs e)

 {

 XDocument xDoc = new XDocument(

 new XElement("DataBase",

 new XElement("Stud",

 new XElement("kod",new XAttribute("Size", "7"), "1"),

 new XElement("fam", "Smirnov"),

 new XComment("Совершеннолетний")

 ),

 new XElement("Stud",

 new XElement("kod",new XAttribute("Size", "8"), "2"),

 new XElement("fam", "Sidorov"),

 new XComment("Несовершеннолетний")

 ),

 new XElement("Stud",

 new XElement("kod",new XAttribute("Size", "9"), "3"),

 new XElement("fam", "Ivanov"),

 new XComment("Совершеннолетний")

 )));

 textBox1.Text= Convert.ToString(xDoc);

 xDoc.Save("K:\\Laba2 - BiBD\\Laba2.xml");

 }

Результат:



1. **Обработка регулярных выражений**

Примеры регулярных выражений:

 [0-9][0-9] – равно 2 цифры

[0-9]\* - может быть ни одной цифры (итератор)

[0-9]+ - как минимум 1 буква(итератор)

[0-9] {4} – не менее 4 знаков

[A-Z] {,4} – не более 4 знаков

[A-Z] {2,2} – ровно 2 буквы

[a][a-z]\* - определяет все слова на «а»

(\b cap) – все слова начинающиеся на “cap”

(cap\b) – все слова заканчивающиеся на “cap”

[a-z]{3,15}\(org|com|by|ru) – если точно известно, что строка заканчивается на символы org|com|by|ru.

Примеры регулярных выражений MySQL:

>select ‘foto’ REGEXP”^fo”; - определяет, начинается ли слово с морфемы.

>select ‘foto’ REGEXP ‘^f?fo’; - ? соответсвует одиночному символу

1. **ASP скрипты**

Пример скрипта, который возвращает текущее время:

<html>

 <body bgcolor="0xaabbee">

 <h1><center>TITLE</center></h1>

 <hr />

 <p> NOW IS <%Responce.Write Date %></p>

 </body>

</html>

Пример скрипта, который получает данные из модели и создаёт новый объект типа Album:

@model MvcApplication1.Models.Album

@{ ViewBag.Title = "Создать новый"; }

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>

<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")" type="text/javascript"></script>

@using (Html.BeginForm()) {

 @Html.ValidationSummary(true)

 <fieldset>

 <legend>Альбом</legend>

 <div class="editor-label">

 @Html.LabelFor(model => model.GenreId, "Жанр")

 </div>

 <div class="editor-field">

 @Html.DropDownList("GenreId", String.Empty)

 @Html.ValidationMessageFor(model => model.GenreId)

 </div>

 <div class="editor-label">

 @Html.LabelFor(model => model.ArtistId, "Исполнитель")

 </div>

 <div class="editor-field">

 @Html.DropDownList("ArtistId", String.Empty)

 @Html.ValidationMessageFor(model => model.ArtistId)

 </div>

 <div class="editor-label">

 @Html.LabelFor(model => model.Title, "Название")

 </div>

 <div class="editor-field">

 @Html.EditorFor(model => model.Title)

 @Html.ValidationMessageFor(model => model.Title)

 </div>

 <div class="editor-label">

 @Html.LabelFor(model => model.Price, "Цена")

 </div>

 <div class="editor-field">

 @Html.EditorFor(model => model.Price)

 @Html.ValidationMessageFor(model => model.Price)

 </div>

 <div class="editor-label">

 @Html.LabelFor(model => model.AlbumArtUrl, "Обложка")

 </div>

 <div class="editor-field">

 @Html.EditorFor(model => model.AlbumArtUrl)

 @Html.ValidationMessageFor(model => model.AlbumArtUrl)

 </div>

 <div class="editor-label">

 @Html.LabelFor(model => model.AlbumRarUrl,"Архив")

 </div>

 <div class="editor-field">

 @Html.EditorFor(model => model.AlbumRarUrl)

 @Html.ValidationMessageFor(model => model.AlbumRarUrl)

 </div>

 <div class="editor-label">

 @Html.LabelFor(model => model.Description, "Описание")

 </div>

 <div class="editor-field">

 @Html.EditorFor(model => model.Description)

 @Html.ValidationMessageFor(model => model.Description)

 </div>

 <p>

 <input type="submit" value="Create" />

 </p>

 </fieldset>

}

<div>

 @Html.ActionLink("Вернуться назад", "Index")

</div>

Класс Album:

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace MvcApplication1.Models {

 public partial class Album {

 public virtual int AlbumId { get; set; }

 public virtual int GenreId { get; set; }

 public virtual int ArtistId { get; set; }

 public virtual string Title { get; set; }

 public virtual decimal Price { get; set; }

 public virtual string AlbumArtUrl { get; set; }

 public virtual string AlbumRarUrl { get; set; }

 public virtual string Description { get; set; }

 }

}

1. **Соединение с БД в ASP скриптах**

Пример:

<html>

 <body bgcolor="0xaabbee">

 <h1><center><font color ="YELLOW"> DATABASE CONNECTION EXAMPLE</font> </center></h1>

 <hr size="3" color="red" />

 <br />

 <font size="5" bold="true" color="yellow">test messaging</font>

 <%

 Dim con, rs, strFio

 Set con = Server.CreateObject("ADODB.Connection")

 Set rs = Server.CreateObject("ADODB.RecordSet")

 con.Open "Provider = Microsoft.Jet.OLEDB.4.0;" & \_"Data Source=c:\1\mydb.mdb"

 if con.State=1 then %> CONNECTED <br /> FIO = <%

 rs.Open "Select \* from stud", con

 if rs.EOF=false then

 strFIO=rs.Fields("Fio")

 else

 strFIO="???"

 end if

 %> <%=strFIO %>

 <% else %> NOT CONNECTED <%

 end if

 Set rs = Nothing

 Set con = Nothing %>

 </body>

</html>

1. **Скрипты ASP.Net**

В качестве скриптов asp.net можно использовать технологию AJAX (Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее.

Также популярны и технология JSON(JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми. Несмотря на происхождение от JavaScript, формат считается языконезависимым и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

Пример:

@model MvcApplication1.Models.ViewModels.ShoppingCartViewModel

@{

 ViewBag.Title = "Shopping Cart";

}

<script src="/Scripts/jquery-1.7.1.min.js"

type="text/javascript"></script>

<script type="text/javascript">

 $(function () {

 // Document.ready -> link up remove event handler

 $(".RemoveLink").click(function () {

 // Get the id from the link

 var recordToDelete = $(this).attr("data-id");

 if (recordToDelete != '') {

 // Perform the ajax post

 $.post("/Cart/RemoveFromCart", {"id": recordToDelete },

 function (data) {

 // Successful requests get here

 // Update the page elements

 if (data.ItemCount == 0) {

 $('#row-' + data.DeleteId).fadeOut('slow');

 } else {

 $('#item-count-' + data.DeleteId).text(data.ItemCount);

 }

 $('#cart-total').text(data.CartTotal);

 $('#update-message').text(data.Message);

 $('#cart-status').text('[ Корзина: ' + data.CartCount + ' | '+ data.CartTotal + ' ]');

 });

 }

 });

 });

</script>

<h3>

 <em>Review</em> your cart:

 </h3>

<p class="button">

 @Html.ActionLink("Checkout>>", "AddressAndPayment", "Cash")

</p>

<div id="update-message">

</div>

<table>

 <tr>

 <th>Album Name</th>

 <th>Price (each) </th>

 <th>Quantity</th>

 <th></th>

 </tr>

 @foreach (var item in

Model.CartItems)

 {

 <tr id="row-@item.RecordId">

 <td> @Html.ActionLink(item.Album.Title,

"Details", "Store", new { id = item.AlbumId }, null) </td>

 <td> @item.Album.Price</td>

 <td id="item-count-@item.RecordId"> @item.Count</td>

 <td><a href="#" class="RemoveLink" data-id="@item.RecordId">Remove from cart</a></td>

 </tr>

 }

 <tr>

 <td>Total </td>

 <td></td>

 <td></td>

 <td id="cart-total">@Model.CartTotal</td>

 </tr>

</table>

1. **Web сервисы**
2. **Технология LINQ to Object**

Language Integrated Query (LINQ) — проект Microsoft по добавлению синтаксиса языка запросов, напоминающего SQL, в языки программирования платформы .NET Framework. Представляет собой не что иное, как функциональное программирование, замаскированное под синтаксис SQL[1]. Ранее был реализован в языках C# и Visual Basic .NET. Множество концепций, которые вводит LINQ, изначально опробовали в исследовательском проекте Microsoft Cω.

Примеры:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication7

{

 class Pr

 {

 public void LA()

 {

 int[] numbers = { 1, 2, 3, 6, 3, 8, 2, 9 };

 var Nums = from n in numbers where n < 5 select n;

 foreach (var X in Nums)

 {

 Console.WriteLine(X);

 }

 Console.Read();

 }

 static void Main()

 {

 Pr z = new Pr();

 z.LA();

 }

 }

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication7

{

 class Pr

 {

 string[] fruit = { "apple", "not\_apple", "cherry", "lemon", "orange" };

 public void LA()

 {

 var z = fruit.Select(p => new { name = p, len = p.Length });

 foreach (var y in z)

 {

 Console.WriteLine("\n{0} is {1} long", y.name, y.len);

 }

 }

 static void Main()

 {

 Pr z = new Pr();

 z.LA();

 }

 }

}

1. **Технология LINQ to XML**

Пример кода: Изменение тегов:

private void button8\_Click(object sender, EventArgs e)

 {

 XElement first = new XElement("Root",

 new XElement("age", 17),

 new XElement("age", 19),

 new XElement("age", 15),

 new XElement("age", 20));

 XElement second = new XElement("Root", "ffff");

 second.ReplaceAll(

 from el in first.Elements()

 where (int)el < 18

 select new XElement("NotPass", (int)el)

 );

 textBox1.Text = Convert.ToString(second);

 second.Save("K:\\Laba2 - BiBD\\Laba2.xml");

 textBox1.Text = "\n\_\_\_\_\_\_\_До изменения\_\_\_\_\_\_\_\_\n" + Convert.ToString(first) + "\n\_\_\_\_\_\_\_\_\_\_\_После\_\_\_\_\_\_\_\_\_\_\_\_\n" + Convert.ToString(second);

 }



1. **Работа в БД через локальную сеть**

Одна из базовых технологий – Клиент-Сервер (Client-Server). Выделим сторону сервера и сторону клиента. Сервер всегда работает и прослушивает определённый порт (порт – ячейка памяти, через которую взаимодействуют клиент, ос, сервер)).

Пример:

Простое приложение сервера:

using System;

using System.IO;

using System.Net;

using System.Net.Sockets;

namespace ConsoleApplication1

{

 class TSPServer

 {

 public void send\_proc()

 {

 IPAddress ipa = Dns.Resolve("Localhost").AddressList[0];

 TcpListener tcplist = new TcpListener(ipa, 13222);

 tcplist.Start();

 System.Console.WriteLine("Waiting for client");

 Socket sock = tcplist.AcceptSocket();

 byte[] b = { (byte)'H', (byte)'E', (byte)'L', (byte)'L', (byte)'O' };

 NetworkStream ns = new NetworkStream(sock);

 ns.Write(b, 0, b.Length);

 ns.Close();

 tcplist.Stop();

 }

 static void Main()

 {

 TSPServer x = new TSPServer();

 x.send\_proc();

 }

 }

}

Простейший клиент:

using System;

using System.IO;

using System.Net.Sockets;

namespace ConsoleApplication2

{

 class Program

 {

 // public send\_proc(){}

 static void Main()

 {

 TcpClient cls = new TcpClient("localhost", 13222);

 NetworkStream ns = cls.GetStream();

 byte[] b = new byte[100];

 ns.Read(b, 0, 20);

 char[] c = new char[20];

 for (int i = 0; i < 20; i++)

 {

 c[i] = (char)b[i];

 }

 Console.WriteLine(c);

 Console.ReadLine();

 ns.Close();

 cls.Close();

 }

 }

}

1. **MySQL. Системные таблицы. Работа с C#**

MySQL — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Составные компоненты:

* Таблицы
* Функции
* Хранимые процедуры (появились с версии 5.0)

Система общается с MySQL проходит через системное окно. Соединение с MySQL: >mysql –u username –p (-p случай, когда система требует пароль).

Пример работы:

MySQL > show database;

MySQL > create database mydb;

MySQL > use mydb;

MySQL > create table Sklad (Tovar, varchar (20), cena int, PRIMARY KEY (Tovar));

MySQL > Insert Into Sklad values (‘milk’, 5400);

Набираем в .txt таблицу вида:

Milk 5400

Bread 6800

Soup 3400

Candy 1000

Сохраняем всё как mydb.txt.

MySQL > load DATA LOCAL IN FILE “c:\1\mydb.txt” INTO TABLE Sclad;

MySQL > select \* from Sclad;

Взаимодействие с С# MySQL:

Создадим пустой консольный проект и подключим библиотечный класс MySQL data. Для этого в обозревателе решений нужно щелкнуть правой кнопкой мыши по имени проекта – добавить ссылку – СОМ – MySQL.Data – ок.

using System;

using System.Collections.Generic;

using System.Text;

using MySQL.Data.MySQLClient;

namespace ConsoleApplication3

{

 class Program

 {

 static void Main(string[] args)

 {

 string cs = @"server=localhost; userid=root; password=1234; database = mydb";

 MySqlConnection conn = null;

 MySqlDataReader rdr = null;

 try

 {

 conn = new MySqlConnection(cs);

 conn.Open;

 string stm = "Select \* from sclad";

 MySqlCommand cmd = new MySqlCommand(stm, conn);

 rdr = cmd.ExecuteReader();

 while (rdr.Read())

 {

 Console.WriteLine(rdr.GetInt32()) + "rdr.GetString"; //wtf???

 }

 }

 catch (MySqlException ex)

 {

 Console.WriteLine("Error: " + ex.Message);

 }

 finally

 {

 if (rdr == null) rdr.Close();

 }

 }

 }

}

Вызов ХП из C#:

using System;

using System.Collections.Generic;

using System.Data;

using MySQL.Data.MySQLClient;

using System.Configuration;

namespace ConsoleApplication4

{

 class Program

 {

 static void Main(string[] args)

 {

 string constr = @"Server=localhost;" + @"UID = root; password = 1234;" + @"Data source = c:\\1\\mydb.mdf";

 MySqlCommand cmd = new MySqlCommand("proc1", new SqlCommand(constr));

 cmd.CommandType = CommandType.StoredProcedure;

 cmd.Connection.Open();

 cmd.Parametr.AddNew.MySqlParameter("par1",MessageItem.INTry\_IN));

 int i = cmd.ExecuteNonQUery();

 cmd.Connection.Close();

 }

 }

}

1. **Создание и вызов функций в MySQL. Примеры**

Необходимо помнить, что текстовые строки необходимо брать в кавычки.

Неправильно: >select trim( sosiski!!!! );

Правильно: >select trim(‘ sosiski!!!! ‘);

>select ASCII(“2”); - возвращает код 2 в ascii

>select CHAR(77,121,83,81,76); - возвращает буквы

> select concat(str1,str2, …); - объединяет строки без пробелов.

> select length(s); - возвращает длину строки (любые строковые константы должны быть в кавычках).

> select locate(‘bar’,’superpuperbar’); - находит номер, с которого первый раз встречается подстрока.

> select locate(‘bar’,’gaybar’,6); - поиск начинается с символа 6.

> select instr(s1,s2); - содержится ли строка s1 в s2? 1(да):0(нет)

> select left(s,n); >right(s,n); - возвращает левые (правые) n символов в строке S.

> select substring(s,p,n) – возвращает подстроку из s, начиная с символа р, отсчитывая n символов.

> select lirim(s); > select ririm(s); - удаляет пробелы слева (справа).

> select trim(s); - удаляет крайние пробелы.

1. **Технология MVC. Базовые понятия и применение**

Model-view-controller (MVC, «Модель-представление-поведение», «Модель-представление-контроллер») — схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента так, что модификация одного из компонентов оказывает минимальное воздействие на остальные. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области.



Концепция:

Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

Модель (англ. Model). Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

Представление, вид (англ. View). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.

Контроллер (англ. Controller). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

Для реализации схемы Model-View-Controller используется достаточно большое число шаблонов проектирования (в зависимости от сложности архитектурного решения), основные из которых «наблюдатель», «стратегия», «компоновщик».

Наиболее типичная реализация отделяет вид от модели путем установления между ними протокола взаимодействия, используя аппарат событий (подписка/оповещение). При каждом изменении внутренних данных в модели она оповещает все зависящие от неё представления, и представление обновляется. Для этого используется шаблон «наблюдатель». При обработке реакции пользователя вид выбирает, в зависимости от нужной реакции, нужный контроллер, который обеспечит ту или иную связь с моделью. Для этого используется шаблон «стратегия», или вместо этого может быть модификация с использованием шаблона «команда». А для возможности однотипного обращения с подобъектами сложно-составного иерархического вида может использоваться шаблон «компоновщик». Кроме того, могут использоваться и другие шаблоны проектирования, например, «фабричный метод», который позволит задать по умолчанию тип контроллера для соответствующего вида.

1. **Понятия о нормальных формах (1НФ, 2НФ, 3НФ)**

1НФ: Требует, чтобы не использовали составные атрибуты, в противном случае обращение к частям запрещается. Таблицы должна быть связаны по общим полям – это реализация первого этапа – построение информационной модели.

2НФ: Понятие полной и частичных зависимостей. Частичная – атрибут зависит от части ключа, 2НФ запрещает частичные зависимости. Чтобы избавиться от частичной зависимости, необходимо одну таблицу расщеплять на 2. (B -> D => AB->C & B->D, была таблица с полями ABCD, стало две таблицы: ABC и BD).

3НФ: Запрет на транзитивные зависимости (зависимости от не ключевого атрибута). (A->B->C, где В – посредник и не ключевой атрибут, необходимо расщепление и объявление В ключом).

1. **Нормальная форма Бойса-Кодда, 4НФ**

На ряду с 3НФ существует особый вид НФ – Бойса-Кодда, где требуется, чтобы в таблице был только 1 ключ, в этом случае аномалии просто невозможны.

4НФ: содержание многозначной ФЗ.

Есть некоторая таблица ABC, где есть ФЗ: A ->>BC. В столбцах ВС представлены все возможные комбинации атрибутов для данного значения А, поэтому, когда имеет место ФЗ, то зависимые атрибуты образуют все возможные комбинации. Для внесения изменений в таблицу без потерь необходима нормализация.

|  |  |  |
| --- | --- | --- |
| A | B | C |
| a1  | b1  | c1  |
| a1 | b1 | c2  |
| a1 | b2  | c1  |
| a1 | b2 | c2  |
| a2  | b1 | c1  |
| a2 | b1 | c2  |
| a2 | b2 | c1  |
| a2 | b2 | c2  |

1. **Аномалии БД**

Сохранение ФЗ при изменении данных – целостность, нарушение целостности – аномалия. Нарушение целостности связано с тем, что расщепление возможно только по ключевому атрибуту. Борьба с аномалиями – формализация.

То есть, расщепление таблиц должно выполняться по принципу сохранения функций ключа.

Аномалии возникают при:

* Расщеплении таблицы
* Добавлении новых записей
* Удаление записей
* Изменение записей
1. **Аксиомы Армстронга**

Определим правила для функциональной зависимости, они позволят проверить, что одни зависимости выходят из других.

* А->A (рефлексивность)
* A -> BC (аксиома расщепления) => A->B и A->C
* A -> B (аксиома положения) => At -> B
* A -> B, B -> C (транзитивность) => A->C
* A->B & BN ->C (псевдо транзитивность) => AN -> C
* A->B & C->D (сложение) => AC -> BD

С помощью аксиом можно проверить выводимость ФЗ. Функциональная зависимость A->В это такое соотношение между А и В, что по значению А однозначно определено и В. Понятие многозначной зависимости также является видом ФЗ.

1. **Определение избыточных атрибутов**

Следующий пример показывает, как на основе аксиом Армстронга проверить избыточность некоторого атрибута:

1. A -> B
2. C -> D
3. AD ->HE

AC -> H

 I. A->B

 II. AC -> D

 AD -> HE \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 ACD -> HE

 III. ACD -> H

 AC -> H

Чтобы проверить , что некоторый атрибут избыточен, надо этот атрибут выбросить и попытаться вывести ту же зависимость. Если зависимость можно вывести, то атрибут лишний. Избыточность отсекается с помощью 2нф. Для упрощения задачи можно обратиться к ациклическим графам вывода.

1. **Ациклические графы вывода**



Поставим метки в тех вершинах, которые соответствуют левой части до правой. Далее начнём распространять метки по графу согласно правилу. Так как D помечено только из С, то распространяет метки. Помечаем и B. На следующем, 3м шаге, помечаем Н, таким образом аксиома доказана.

1. **Проверка избыточных функциональных зависимостей**

Функциональная зависимость A->В это такое соотношение между А и В, что по значению А однозначно определено и В. Понятие многозначной зависимости также является видом ФЗ. В современных СУБД понятие ФЗ является основополагающим. При проектировании БД определяются не только ключи, но и виды ФЗ. Сохранение ФЗ при изменении данных – целостность, нарушение целостности – аномалия. Основной инструмент приведения к целостности – расщепление.

1. **Минимизация числа функциональных зависимостей**

Пример: Рассмотрим следующее множество зависимостей, построим матрицу:

* A->B
* B->C
* B->H
* E->A
* EC->B
* H->C

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | F | H |
| 1 |  | 1 |  |  |  |
| 2 |  |  | 1 |  |  |
| 3 |  |  |  |  | 1 |
| 4 | 1 |  |  |  |  |
| 5 |  | 1 |  |  |  |
| 6 |  |  | 1 |  |  |

Ставим 1, если данный атрибут выпил в данной зависимости.

Если столбец полностью нулевой, то его нельзя вычислить никакой зависимостью. Зависимости 4,5 войдут в наш набор. Выбрасываем из матрицы столбцы определённые этими зависимостями. Удаляем из левых частей выбранные атрибуты. Первые 3 зависимости потеряли смысл, удаляем их и получаем сокращённую матрицу:

* E->A
* EC->B
* H->C

|  |  |  |  |
| --- | --- | --- | --- |
|  | C | F | H |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 | 1 |  |  |

Мы получили минимальное покрытие матрицы, минимально необходимое число ФЗ это <4,5,6>.

1. **ER диаграммы**
2. **Построение таблиц по ER диаграммам**
3. **Задача о минимизации ключа**

Пример: Имеется таблица:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | x4 |
| D1 | 1 | 0 | 0 | 1 |
| D2 | 0 | 1 | 1 | 1 |
| D3 | 0 | 0 | 1 | 0 |
| D4 | 1 | 1 | 0 | 0 |
| D5 | 1 | 0 | 0 | 0 |
| D6 | 0 | 0 | 1 | 1 |

Задача состоит в том, чтобы из Х найти наименьшее количество, которые образуют ключ (минимальный ключ).

Данная таблица нам не подходит, зададим другую, матрицу покрытия, назовём её Cover:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | (D1,D2) | (D1,D3) | (D1,D4) | (D1,D5) | (D1,D6) | (D2,D3) | (D2,D4) | (D2,D5) | (D2,D6) |
| х1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| х2 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| х3 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| х4 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

Таблица заполнена следующим образом: в ячейке [x1, (D1,D2)] пишем 1, если х1 различает строки D1 и D2, то есть, имеем 1 в одной строке и 0 в другой.

Теперь формулировка задачи такая: найти минимальное покрытие матрицы Cover. Под покрытием понимается множество строк, таких, что в каждом столбце хотя бы одна из этих строк содержит 1. Например, П = <x1,x2,x3> - не является минимальным покрытием. Нас интересует покрытие с минимальным числом строк. Это достаточно сложная задача, поэтому рассмотрим приближенный метод:

Если имеется столбец, где одна 1, то данная строка должна войти в решение (x2,x4). Можно вычеркнуть столбец, где эта строка содержится. Далее будем рассматривать последующие включения в решения строки с максимальным числом 1. х1 = <x2,x4,x1>. <x1,x2,x4> - минимальный ключ. Задача с нечисловыми данными решается аналогично: строится матрица покрытия, далее поиск минимального покрытия.

Замечания:

* Приближенное решение можно найти с помощью «жадного алгоритма».
* Некоторая специфика связана с непрерывным значение атрибута, в таком случае атрибуты необходимо разбивать на диапазоны (корень из N, где N является количеством точек).
1. **Взаимодействие C# с Oracle**

Oracle Database или Oracle RDBMS — объектно-реляционная система управления базами данных компании Oracle. БД Oracle используется не только как хранилище, но и как визуализация интерфейса бд. Имеется внутренний язык PLISQL, средства для администрирования SQL\*PLUS, мастер для создания БД и другое.

Пример работы с C#:

using System;

using System.Collections.Generic;

using System.Text;

using System.Data;

using System.Data.OracleClient;

namespace ConsoleApplication5

{

 class Program

 {

 static void Main(string[] args)

 {

 using (OracleConnection ocon = new OracleConnection("DataSource=OKCL; userId = stud; password = 1234"));

 OracleCommand ocmd = new OracleCommand();

 ocmd.Connection(ocon);

 ocmd.CommandText = "myProc";

 ocmd.CommandType = CommandType.StoredProcedure;

 ocmd.Parameters.Add("Derp", OracleType.Number).Value();

 ocmd.Parameters.Add("pcount", OracleType.Number).Direction() = ParameterDirection.Output;

 try

 {

 ocon.Open();

 ocmd.ExecuteNonQuery();

 System.Console.WriteLine("Number returned =" + ocmd.Parameters["pcount"].Value());

 }

 catch (Exception) { }

 ocon.Close;

 System.Console.ReadLine();

 }

 }

}

1. **Хранимые процедуры и модули Oracle**

При обращении к примеру п.43, процедура будет иметь вид:

Create or replace procedure myproc ( derpno number, count out number)

Is

Begin

Select count (x)

Into pcount

From employe

Where derp = derpno

End myproc

1. **Развитые типы данных Oracle (массив, таблица, объект)**
* Массив varray представляет собой коллекцию объектов определённого типа.

SQL> create type

 Tax\_tp as object(

 Year Date

 Tax number);

SQL>create type

 My\_arr as array(3) of Tax\_tp;

Объявление массива:

DECLARE

TYPE t\_ar IS VARRAY(250) OF VARCHAR(100);

V1 t\_ar;

i NUMBER :=0

BEGIN

 LOOP

 IF (i >= 100) THEN

 EXIT;

 END IF;

V1(i):=TO CHAR(i);

END LOOP;

END

Типовая заготовка (структура) ХП oracle может быть представлена определённым образом:

Create or replace procedure

 имя\_процедуры

Is

Declare

…объявление переменных и типов…

Begin

… исполнительный порядок…

Exception

…Обработка исключения…

End имя\_процедуры

* Таблицы:

Тип таблиц ближе к массивам, но отличие в том, что каждый элемент таблицы имеет индекс, например, V(-15):=’Mir’. Количество элементов не фиксировано, объявление таблицы:

DECLARE

TYPE t\_tab IS TABLE OF VARCHAR(10)

 INDEX BY BINARY\_INTEGER;

V1 t\_tab

BEGIN

 V1(-4) = ‘John’

 V1(0) = ‘Doe’;

END;

В отличии от массива, в таблице индексы не обязаны последовательно возрастать. Пример с заполнением таблицы:

DECLARE

TYPE t\_tab IS TABLE OF Stud %TYPE

 INDEX BY BINARY INTEGER;

V1.t\_tab

BEGIN

 SELECT \* INTO V1(-2)

 FROM stud

 WHERE Id = -2

END;

В таблице и массиве есть ряд доступных свойств: COUNT, FIRST, LAST, NEXT, DELETE(10) (удаление строки с индексом 10), DELETE(10,200) (удаление записи из диапазона от 10 до 200). Пример:

DECLARE

TYPE t\_tab IS TABLE OF stud.FN%TYPE

 INDEX BY BINARY\_INTEGER;

FNAMES t\_tab;

BEGIN

 FNAMES(1):=’John’;

 FNAMES(3):=’Doe’;

IF FNAMES.EXISTS(1) THEN FNAMES.DELETE(1)

END IF;

END

* Курсоры:

Declare

 Cursor ordc is

 Select \* from Orders;

 Cursor get\_Orditem(prom\_num orders.ord\_num % type) is

 Select quantity

 From orders where ord\_num = prod.num;

Пример работы с курсорами:

DECLARE

 V\_stud students.IO%TYPE;

 V\_FirstName students.FirstName%TYPE;

 V\_Depart students.Department%TYPE:=”Computer Science”;

 CURSOR c\_Students IS

 SELECT id, FirstName

 FROM students

 WHERE Department=V\_Depart

 Begin

 Open c\_Students;

 LOOP

 FETCH c\_Students INTO V\_stud, v\_FirstName

 EXIT WHEN

 C\_Students% NOT FOUND;

 END LOOP

CLOSE c\_Students;

END;

Пример с курсорными переменными:

DECLARE

 TYPE L\_classes IS REF CURSOR

 Return classes%ROWTYPE

 V\_c1 c\_Classes;

 Open v\_c1 FOR

 SELECT \* FROM Classes;

REF – ссылка на курсор; V\_c1 – курсорная переменная.

Пример с функцией:

CREATE OR REPLACE FUNCTION MyFun (x IN BOOLEN)

 RETURN VARCHAR2

 IS

 DECLARE

 Str VARCHAR(5);

 Begin

 IF (x) THEN

 Str:=”TRUE”;

 ELSE

 Str:=”FALSE”;

 ENDIF

RETURN(Str);

END MyFun

Также имеется большое количество встроенных функций, таких как CHR(x) (возврат символа в формате ASCII), CONCAT(s1,s2) (Объединение строк), SUBSTR(s,n1,n2) (выделение в строке S n2 символов начиная с n1) и так далее.