Весь процесс обработки запроса можно разделить на три этапа:

1. Соединение с БД и получение объекта DbLinker;
2. Инициализация ORM;
3. Загрузка файла IRP и извлечение сведений о полях запроса;
4. Передача полученных сведений о полях запроса в метод ICSMQuery

Ниже я привел участки кода, которые показывают порядок создания и вызова объектов прослойки доступа к данным (В существующем варианте исходников WebQuery эти вызовы проходят через ряд методов, но для того, чтобы увидеть сразу всю цепочку инициализаций я вынес все в представленную ниже структуру вызовов).

**Соединение с БД (Oracle или MS SQL)**

//Инициализация прослойки доступа к БД (передаем значение параметра System.Data.SqlClient для БД MS SQL)

DatalayerCs.ANetDb db = DatalayerCs.ANetDb.New(“System.Data.SqlClient”);

// указываем строку соединения с БД

db.ConnectionString = “Connection String”;

// отрываем соединение

db.Open();

//здесь имя схемы БД:

string SchemaDbx = “dbo.”;

//здесь путь к файлу ICSM.schema (это файл конфигурации, который содержит описание всех таблиц и их структур для последующего обращения к ним через объекты ORM-а):

string SchemaDbx = “Path to file ICSM.schema”;

// Инициализация схемы БД

OrmSchema.InitIcsmSchema(db, SchemaDbx, PathDbx);

// Создаем объект DbLinker

DbLinker Connect\_ = new DbLinker(dbx, SchemaDbx);

**Инициализация ORM**

// Инициализация объекта ORM-а

OrmCs.OrmRs orm = new OrmCs.OrmRs();

// передаем объект DbLinker

orm.Init(Connect\_);

orm.m\_allFetched = true;

orm.AllFetched();

**Загрузка файла IRP, извлечение метаданных**

//Создаем объект для работы с репортом - (IRP-файлом)

IcsmReport ics = new IcsmReport();

// связываем с объектом orm, который был создан выше

ics.m\_records.LinkTo(orm);

//указываем имя файла IRP,который следует загрузить (к примеру “Name IRP-file.irp”)

string gd\_name = “Name IRP-file.irp”;

//загрузка файла Irp

ics.Load(gd\_name);

// если загрузка файла прошла усшено, то:

if (ics!=null)

{

string name\_table = ics.m\_dat.m\_tab; // здесь имя основной таблицы запроса

Query name\_table = ics.m\_dat.m\_list[0].m\_query; // здесь вся структура запроса Query (m\_list – может состоять из нескольких запросов, но мы принимаем, то запрос для одного IRP-файла всегда один)

QueryItem[] items = ics.m\_dat.m\_list[0].m\_query.lq – набор всех полей запроса;

string title = ics.m\_dat.m\_list[0].m\_query.lq[0].title – заголовок отдельно взятого поля запроса;

bool isCustExpr = ics.m\_dat.m\_list[0].m\_query.lq[0].m\_isCustExpr – признак является ли это поле вычисляемым значением;

string path = ics.m\_dat.m\_list[0].m\_query.lq[0].path – наименование отдельно взятого поля запроса (именно по этим именам впоследствии ОРМ будет выполнять построение SQL – запроса, т.е. эти наименования должны соответствовать тем, что предоставляет и отображает ICS Manager через QueryEditor F2);

string format = ics.m\_dat.m\_list[0].m\_query.lq[i].format – формат отдельно взятого поля (здесь имеется ввиду специфический тип поля, т.е. к примеру - MHz, dB, dBw, Latitude, Longitude);

string CustExpr = ics.m\_dat.m\_list[0].m\_query.lq[i].m\_CustExpr – получаем код подзапроса (custom expression);

}

// пример вызова метода ICSMQuery.

// данный метод ICSMQuery выполнит запрос для таблицы MICROWA и сделает выборку по полям // {“ID”, “StationA.Position.ADDRESS” , “StationA.Position.CITY”, “StationA.POWER”, при // этом результирующая выборка будет отсортирована по убыванию для столбца ID и общее

// число записей результирующей выборки не должно превышать 1000:

ICSMQuery(“MICROWA”, new string[] {“ID”, “StationA.Position.ADDRESS” , “StationA.Position.CITY”, “StationA.POWER”}, “[ID]>0”, “[ID] DESC”, 1000, null, Connect\_, true);

Таким образом, файл IRP нам необходим только для извлечения метаданных (сведений о полях запросов и подзапросов), которые впоследствии должны быть отправлены в функцию ICSMQuery

Ниже приведена основная функция - ICSMQuery, выполняющая запрос.

От базовой (которую написал Филипп), она отличается возможностью обрабатывать CustomExpression, данные о которых передаются в отдельном входящем параметре LstOrmItemExpr)

/// <param name="table">Имя основной таблицы запроса</param>

/// <param name="flds">Перечень полей запроса</param>

/// <param name="filter">Фильтр для отбора записей</param>

/// <param name="order">порядок сортировки</param>

/// <param name="nbMaxRec">Ограничение на максимальное число записей, которое должен возвратить запрос</param>

/// <param name="LstOrmItemExpr">Список полей, которые содержат результаты выполнения подзапросов (Custom Expression)</param>

/// <param name="lk">Объект OrmLinker</param>

/// <param name="isMaxRec">признак указывающий на то - учитывать или нет ограничение на максимальное число записей, возвращаемых в результате</param>

/// <returns>Список строк object[], где отдельно взятое значение – соответствует полям, которые были переданы в flds</returns>

static public List<object[]> ICSMQuery(string table, string[] flds, string filter, string order, int nbMaxRec,List<OrmItemExpr> LstOrmItemExpr , OrmLinker lk = null, bool isMaxRec = true)

{

// результирующая выборка

List<object[]> dataset = new List<object[]>();

try

{

//Инициализация объекта ORM-а

OrmRs rs = new OrmRs();

rs.Init(lk == null ? OrmSchema.Linker : lk);

int count = flds.GetLength(0);

// создаем масив OrmItem для хранения информации о полях запроса

OrmItem[] it = new OrmItem[count];

// создаем масив OrmVarType для хранения информации о типах полей запроса

OrmVarType[] vt = new OrmVarType[count];

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

{

string fld = flds[i]; if (fld.StartsWith("[") && fld.EndsWith("]")) fld = fld.Substring(1, fld.Length - 2);

//проверка есть ли подзапросы (CustomExpression)

bool isExpr = false;

if (LstOrmItemExpr != null)

{

if ((LstOrmItemExpr.Find(r => r.m\_name == fld) != null))

{

isExpr = true;

//записываем в элемент OrmItem определение для //соответсвующего ему подзапроса

//в данном случае, т.к. поле является подзапросом, тогда // добавляем его с помощью метода AddExpr

it[i] = rs.AddExpr(table, LstOrmItemExpr.Find(r => r.m\_name == fld).m\_name, LstOrmItemExpr.Find(r => r.m\_name == fld).m\_expression, LstOrmItemExpr.Find(r => r.m\_name == fld).m\_sp);

}

}

//если поле не является вычисляемым, тогда просто добавляем его с //помощью метода AddFld

if (!isExpr)

{

it[i] = rs.AddFld(table, fld, null, true);

}

// определяем типы полей

vt[i] = it[i] == null ? OrmVarType.var\_Null : it[i].m\_dataDesc.ClassType;

Type nettype;

switch (vt[i])

{

case OrmVarType.var\_Dou: nettype = typeof(double); break;

case OrmVarType.var\_Flo: nettype = typeof(float); break;

case OrmVarType.var\_Guid: nettype = typeof(Guid); break;

case OrmVarType.var\_Int: nettype = typeof(int); break;

case OrmVarType.var\_String: nettype = typeof(string); break;

case OrmVarType.var\_Tim: nettype = typeof(DateTime); break;

default: nettype = typeof(object); break;

}

}

// указываем условие отбора записей

rs.SetAdditionalFilter(table, filter);

// задаем сортировку

rs.SetExternalOrder(table, order);

for (rs.Open(); !rs.IsEOF(); rs.MoveNext())

{

// строка ниже позволяет получить доступ к итоговому SQL - коду

string SQL = rs.m\_ars\_sql;

object[] row = new object[count];

for (int j = 0; j < count; j++)

{

if (it[j] != null)

{

object c = it[j].Val;

if (vt[j] == OrmVarType.var\_Int && c is double)

{

if ((double)c == NullD) c = 0x7FFFFFFF; else c = (int)(double)c;

}

row[j] = c;

}

else row[j] = "";

}

//добавляем очередную строку в результирующий запрос

dataset.Add(row);

//здесь проверка – если признак isMaxRec = true, тогда учитывается параметр nbMaxRec и при достижении счетчика этого значения – nbMaxRec цикл обработки прекращается

if (isMaxRec)

{

if (dataset.Count == nbMaxRec) break;

}

}

rs.Clear();

}

catch (Exception) {

}

return dataset;

}

Относительно итогового SQL-кода, который генерирует ORM на основе перечня полей, фильтра и порядка сортировки, передаваемых в качестве входных параметров пользователем:

В функции ICSMQuery красным цветом выделена строка string SQL = rs.m\_ars\_sql; Данная строка позволяет получить конечный вариант SQL-кода, который передается БД на выполнение.

Здесь есть большой недостаток, т.к. в данном случае, чтобы получить доступ к этому значению необходимо открыть курсор. Данный недостаток можно устранить путем написания вспомогательной функции, которая будет генерировать код SQL на основе перечня свойств объектов OrmItem (за основу мне нужно будет взять код Филипа, который формирует значение переменной m\_ars\_sql).